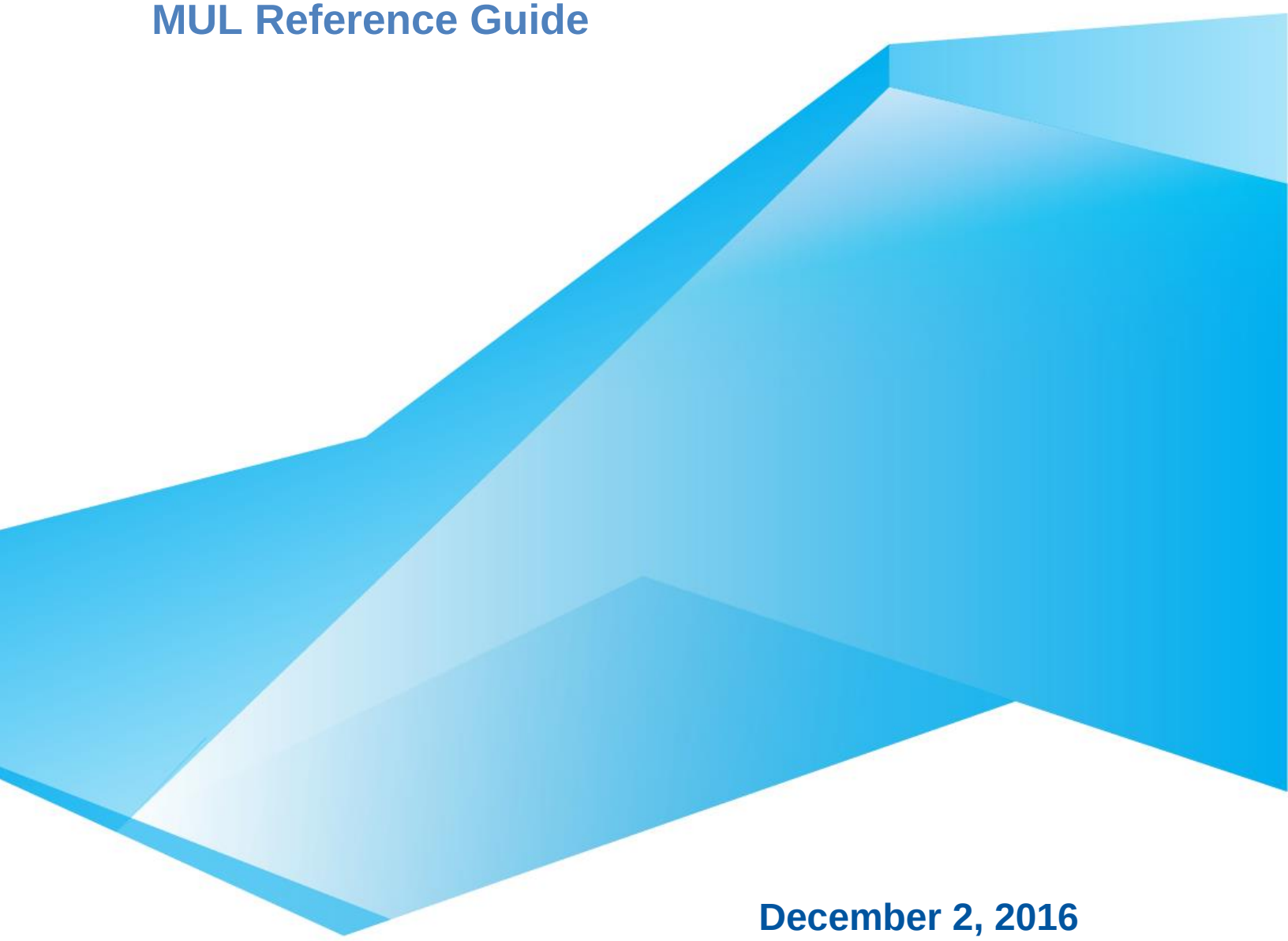
Deltek Know more. Do more.™

Deltek Maconomy 2.3 GA

MUL Reference Guide



December 2, 2016

While Deltek has attempted to verify that the information in this document is accurate and complete, some typographical or technical errors may exist. The recipient of this document is solely responsible for all decisions relating to or use of the information provided herein.

The information contained in this publication is effective as of the publication date below and is subject to change without notice.

This edition published December 2016.

Contents

Introduction 1
Separating Data Models from Queries 1
How Data Models Work 2
Reading this Manual 3
Definition 4
Universe 4
Common Syntax Elements 22
Maconomy Functions 23
Version History 24
MUL 1.3 24
MUL 1.2 24

Introduction

The Maconomy Universe Language (MUL) is used when defining a Maconomy universe. A Maconomy universe is a specification of a data model or part of a data model. A universe is used when interacting with the Maconomy database through the Maconomy Query Language (MQL).

Separating Data Models from Queries

When writing an SQL query, the developer must know the join structure of the relations in the database. The join structure and semantic information about fields in the database, also called the data model, is very complex in the Maconomy system.

An SQL query contains an implicit data model through the selection of relations and join restrictions. But the query also contains a selection of fields, a definition of fields, and other restrictions. The consequence of hiding the data model in one query is that the data model is difficult to reuse and maintain. It is also impossible to create a query if you do not know the data model very well.

The following example shows a simple SQL query, where it is difficult to see the data model.

```
select activity.activitynumber,
       activity.activitytext,
       employee.Name1,
       SUM(timesheetline.numberofday1+ ...
+timesheetline.numberofday7) as regtime
from   timesheetline, activity, employee
where  timesheetline.employeenumber = employee.employeenumber
  and  timesheetline.periodestart   = '2001.01.01'
  and  activity.activitynumber = timesheetline.activitynumber
  and  SUM(timesheetline.numberofday1 + ... +
timesheetline.numberofday7) > 0
group by activity.activitynumber, activity.activitytext,

employee.Name1
```

By separating the data model from the query, you can reuse the data model in more than one query. The separation also makes the data model visible and therefore easier to explore and maintain. In addition, it enables a person who is not familiar with the data model to express a query using an already defined data model.

In the following example, the data model implicitly defined in the preceding SQL query is explicitly defined using a universe. The defined universe is called `timesheetUniverse`. Two different queries are defined using this universe. The first query is the same as the SQL query:

```
Universe:
<mul 1.3>
<universe timesheetUniverse>
  <join (TimeSheetLine, Employee)>
    .EmployeeNumber
  <end Join>
  <join (TimeSheetLine, Activity)>
    .ActivityNumber
  <End Join>

  <field regtime>

  SUM( timesheetline.numberofday1 + ... + timesheetline.numberofday7
)
  <end field>

<end universe>
```

Query 1:

```
mql 1.3
mselect Activity.ActivityNumber, Activity.ActivityText,
Employee.Name1, Regtime
from    timesheetUniverse
where   regtime > 0.0
  and   timesheetline.periodstart = 2001.01.01

order by Activity.ActivityNumber
```

Query 2:

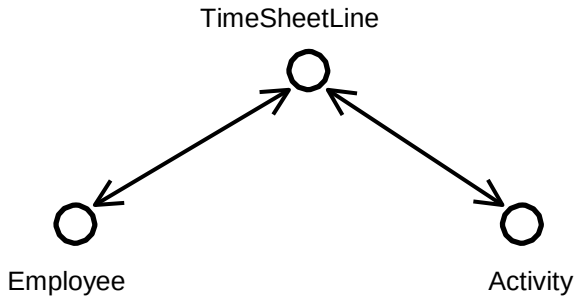
```
mql 1.3
mselect Employee.EmployeeNumber, Regtime
from    timesheetUniverse
where   timesheetline.periodstart inrange [ 2003.01.01 .. 2003.01.31
]

order by Employee.EmployeeNumber
```

How Data Models Work

In an SQL query where the data model and the query are mixed, you can adapt the data model so that it fits the given situation exactly. When separating the data model and the query, this kind of exact fit is no longer possible, because the data model is to be used in many different queries that are not available when designing the data model. When designing the data model, it is therefore necessary to have an idea of how the data model and query are combined.

The data model can be seen as a graph, where objects are vertexes, and joins are bidirectional edges. The graph defined by a universe may not contain cycles.

In the previous example, a universe was defined. The universe joins the three objects TimeSheetLine, Employee, and Activity. These joins can be illustrated as the following tree.



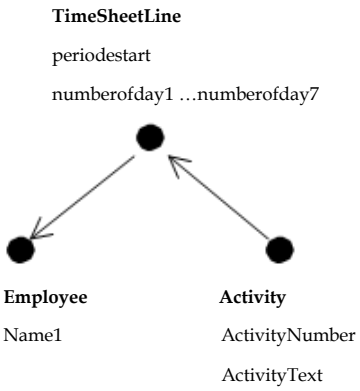
A query selects fields from the objects used in the universe, thereby selecting vertexes in the graph. The first field selected in the query selects the root of the tree. Now the least spanning tree marks the objects needed in the query.

Note the implicit root selection specified in the first selected field of the query, because it might result in different query results and unexpected behavior if two columns are switched around. In particular, this can happen when outer joins are used. Additionally, the Workspace Client does not keep the first column constant. The “first selected field” is the first field in the query, unless the first field is a constant.

After finding the least spanning tree, objects not selected are removed from the tree. The effect is that only objects needed in the query are used in the generated SQL query.

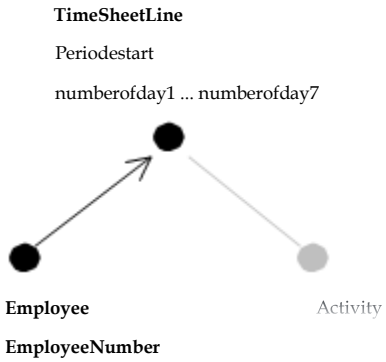
In the following examples, the previously defined universe is used in two queries. In the first query, the first selected field is ActivityNumber from the object Activity, and the least spanning tree contains all of the objects in the universe. Note that the field Regtime is defined using the fields numberofday1, ..., numberofday7 from the object TimeSheetLine.

```
mql 1.3
mselect Activity.ActivityNumber,
        Activity.ActivityText,
        Employee.Name1,
        Regtime
from    timesheetUniverse
where   regtime > 0.0
  and   timesheetline.periodstart = 2001.01.01

order by Activity.ActivityNumber
```



In the second query, the first selected field is EmployeeNumber from the object Employee. The least spanning tree does not contain the Activity object, indicating that the object is not needed in the generated SQL query.

```
mql 1.3
mselect Employee.EmployeeNumber,
        Regtime
from    timesheetUniverse
where   timesheetline.periodstart inrange
        [ 2003.01.01 .. 2003.01.31 ]

order by Employee.EmployeeNumber
```



Reading this Manual

The formal syntax of MUL is presented in BNF (Bachus Naur Form).

Definition

This section describes the current version of MUL. For a version history of MUL, see Version History.

Effort has been made to turn MUL into a programmer-friendly language. The syntax of MUL is a tag-based language, and it consists of elements and attributes just like XML. Unlike XML, every attribute in MUL has an associated type and can have a short form.

```
mul ::=
 (<MUL 1.3> | MUL 1.3)
 universe
```

Universe

The universe element defines a universe. The universe element can be used inside another universe element, also known as an internal universe.

```
universe ::=
 <universe (name = module|module) [(title = string|string)] >
   [help]

   object*
   join*
   parameter*
   field*
   restriction*
   interface*

 <end universe>
```

The following are attributes of the universe element, when defining a new universe.

Attribute	Description
name	The name of the universe. The universe name is used when referring to the universe.
[title]	The title of the universe.

In the following example, a very simple universe is defined. The universe name is reference::U001, and uses the relation employee. Users of this universe can only access the fields EmployeeNumber and EmployeeName. All of the other fields from the employee relation cannot be accessed because they are not part of the universe interface.

```
<MUL 1.3>
<universe reference::U001 "Reference Manual Universe 001">
   <object employee>
   <interface>
      .EmployeeNumber :employee.EmployeeNumber
      .EmployeeName   :employee.Name1
   <end interface>

<end universe>
```

Help

With the help element, external help information regarding the universe can be specified. Help for a universe can be specified using five levels.

1. **name** is the identifier of the universe, and is used for referring to the universe. The name is defined in the universe element.
2. **title** is a descriptive name for the universe, and is used in listings shown to the user. The title is defined in the universe element.
3. **description** is a very short description of what the universe can be used for. The description is for use in universe listings or mouse-over help.
4. **summary** is a short version of the help text.
5. **text** is an in-depth description of what the universe can be used for and how it works.

```
help ::=
 <help [(description =
   string|string)] > [summary]
   [text]
 <help end>

summary ::=
  <summary>
    [text]
  <end summary>

text ::=
  <text (text = text|text) /> | text
```

The following example defines a universe help element where all levels of help are defined.

```
...
<help "MUL Reference Manual, Universe">
  <summary>
    #This Universe is for the MUL Reference Manual#
  <end summary>
    #
    This Universe illustrates the help element, and is used as an
example
    in the MUL Reference Manual.
    #
<end help>

...
```

Objects

An object is a relation in the Maconomy database. An object element in a universe refers to an existing object, refers to an external universe, or creates a “new” object using one of the standard database set operators.

```
object ::=
  objectRef | universeRef | objectUnion
```

Object Reference

An object is a relation in the Maconomy database, defined by Maconomy or added to the system using MOL (Maconomy Object Language). See the *Deltek Maconomy MOL Language Reference*.

```
objectRef ::=
  <object (name = id|id) [basis = id] >
```

The following are attributes of the object element.

Attribute	Description
name	The name of the object used when referring to the object in the universe definition. If the name is an existing object in the Maconomy database, the basis attribute can be omitted.
[basis]	The name of an existing object in the Maconomy database.

In the following example, two references to the Maconomy object Employee are defined and named Employee and SeniorEmployee. This is useful when using the same object in different join situations.

```
...
<object Employee>
<object SeniorEmployee basis=Employee>
<join (Employee, SeniorEmployee) OuterNormal>
  .SeniorEmployee <-> .EmployeeNumber

<end join>
```

Universe Reference

The universe reference element defines an alias for an existing universe. By using the alias, an existing universe can be used just like any other object.

Using an external universe as an object can be used for modularization of a complex universe, and for reuse of already defined universes.

```
universeRef ::=
  <universe (name = id|id) basis = id />
```

The following are attributes of the universe element, when used for referring to an existing universe.

Attribute	Description
name	The alias for the external universe.
basis	The name of an existing universe

In the following example, the external universe reference::U001 is referred and given the name SeniorEmployee. After the alias definition, the fields from the universe can be used just like any other objects.

```
...
<object Employee>
<universe SeniorEmployee basis=reference::U001>
<join (Employee, SeniorEmployee) OuterNormal>
  .SeniorEmployee <-> .EmployeeNumber
<end join>

...
```

Set Operator, Union

The objectUnion element defines a “new” object using the standard database union operator on two objects. The effect of the union operator is that a new object with the union of all rows in the two objects is created.

The fields available from the object defined are the union of all of the fields defined in the two objects. If a field only exists in one of the objects, a constant null value field is automatically created.

```
objectUnion
    ::=
   <objectUnion (name = id|id) [unionAll(+|-)] > object

  object
   <end objectUnion>
```

The following are attributes of the objectUnion.

Attribute	Description
name	The alias for “new” object created.
[unionAll]	Controls the handling of duplicate rows. If unionAll- (default), duplicate rows are eliminated. If unionAll+, duplicate rows are retained.

The following example defines the union object MyUnionObject from two internally defined universes. The object MyUnionObject has four fields: employeeNumber, employeeName, fieldX, and fieldY. Note that the names of the objects X and Y are not used when referring to a field from the object MyUnionObject.

```
...
<objectUnion MyUnionObject>

  <universe X>
    <object Employee>
    <interface>
      .employeeNumber : Employee.EmployeeNumber
      .employeeName   : Employee.Name1
      .fieldX         : type=Boolean :value=true
    <end interface>
  <end universe>

  <universe Y>
    <object Employee>
    <interface>
      .employeeNumber : Employee.EmployeeNumber
      .employeeName   : Employee.Name2
      .fieldY         : type=Boolean :value=true
    <end interface>
  <end universe>

<end objectUnion>

...
```

Join Definition

The join element defines how to join two objects. Together with the objects defined by the object elements, the joins define a graph without cycles. That is, every object defined by an object element must be joined to at least one other object. It is not necessary to define an object element for a Maconomy object before using it in a join.

```
join ::=
  <join (id, id) [(joinType = joinTypeEnum |
   joinTypeEnum)] > (.id <-> .id | .id)*

  <end join>


joinTypeEnum ::=
  Normal | Required | Outer | NormalOuter | OuterNormal
```

A join between two objects contains a list of pairs of join fields. The first field in a pair must exist in the first object, and the second field must exist in the second object. The two fields in the pair must have the same type.

In the following example, the three objects MyEmployee, TimesheetLine, and JobRegistrationBudgetRelation have been joined in a graph without cycles. In the join between the objects TimeSheetLine and MyEmployee, one pair of join fields has been defined using the field .EmployeeNumber as both the first and the second element of the pair. In the join between the objects TimesheetLine and JobRegistrationBudgetRelation, two pairs of join fields are used. Note that no object element is necessary for the objects TimesheetLine and JobRegistrationBudgetRelation.

```
...
<object MyEmployee basis=Employee>

<join (TimeSheetLine, MyEmployee)>
  .EmployeeNumber
<end Join>

<join (TimeSheetLine, JobRegistrationBudgetRelation)>
  .EmployeeNumber
  .LineNumber <-> .TimeSheetLineNumber
<end join>

...
```

A join is a bidirectional edge between the two objects, and for each direction a join type is associated.

In the examples used for each join type, the following objects and contents are used.

TimeSheetLine Object

EmployeeNumber	LineNumber	ActivityNumber
100	1	200
100	2	201
100	3	201
102	1	

Activity Object

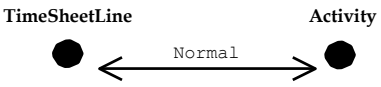
ActivityNumber	ActivityText
200	Design
201	Programming
202	Verification

Join Type, Normal

The join type Normal between the two objects X and Y defines a database normal join from X to Y and a database normal join from Y to X. A database normal join indicates that only rows from an object that has at least one matching row in the other object is selected. The join is only supplied when fields from both objects are selected. The join type Normal is the default value.

In the following example, the Normal join type is used in a universe to define the join between the two objects TimeSheetLine and Activity. The first query selects the object TimeSheetLine as root, and the second query selects the object Activity as root.

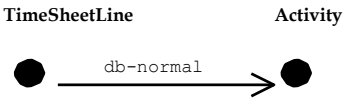
```
<mul 1.3>
<universe reference::U002 "Reference Manual Universe 002">
   <join (TimeSheetLine, Activity)>
     .ActivityNumber
   <end join>
<end universe>
```



Query 1:

```
mselect Timesheetline.EmployeeNumber,
        Timesheetline.ActivityNumber,
        Activity.ActivityText
from reference::U002
order by Timesheetline.EmployeeNumber
```

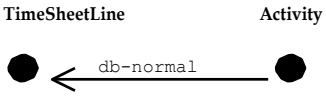
EmployeeNumber	ActivityNumber	ActivityText
100	200	Design
100	201	Programming
100	201	Programming



Query 2:

```
mselect Activity.ActivityNumber,
        Activity.ActivityText,
        Timesheetline.EmployeeNumber,
        Timesheetline.LineNumber
from reference::U002
order by Activity.ActivityNumber
```

ActivityNumber	ActivityText	EmployeeNumber	LineNumber
200	Design	100	1
201	Programming	100	2
201	Programming	100	3



Join Type, Required

The join type Required is the same as the join type Normal, but the join is always supplied, even if no fields from the objects are selected. This join type is to be used if the Normal join has a semantic meaning. This join type reduces the optimization possibilities, so use it with care.

In the following example, the Required join type is used in a universe to define the join between the two objects TimeSheetLine and Activity. The query only selects fields from the object Activity, but as the result reveals, the join of the Object TimeSheetLine is enforced.

```
<mul 1.3>
<universe reference::U003 "Reference Manual Universe 003">
   <join (TimeSheetLine, Activity) Required>
     .ActivityNumber
```

```
      <end join>
   <end universe>
```

```
mselect Activity.ActivityNumber,
        Activity.ActivityText
from reference::U003
order by Activity.ActivityNumber
```

ActivityNumber	ActivityText
200	Design
201	Programming
201	Programming

Join Type, Outer

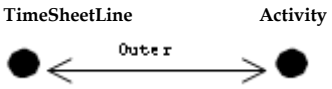
The join type Outer between the two objects X and Y defines a database outer join from X to Y and a database outer join from Y to X. A database outer join indicates that all rows from an object are selected, even if no matching rows are found in the other object. The join is only supplied when fields from both objects are selected.

If rows are found in the joined object, the database outer join behaves just like a database normal join. If no rows are found in the joined object, fields from this object are filled with the Maconomy null value matching the type of each field. Note that the null value is **not** the database null value.

Note that if one of the key fields of the outer joined object is the Maconomy null value, no rows were found in the joined object.

In the following example, the Outer join type is used in a universe to define the join between the two objects TimeSheetLine and Activity. The first query selects the object TimeSheetLine as root, and the second query selects the object Activity as root. A new field outerExists has been defined in the first query, indicating whether or not rows were found in the outer joined object.

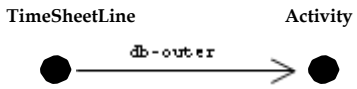
```
<mul 1.3>
<universe reference::U004 "Reference Manual Universe 004">
   <join (TimeSheetLine, Activity) Outer>
     .ActivityNumber
   <end join>

<end universe>
```



Query 1:

```
mselect Timesheetline.EmployeeNumber,
        Timesheetline.ActivityNumber,
        Activity.ActivityText,
        Activity.ActivityNumber!=string'null as
outerExists
from reference::U004
order by Timesheetline.EmployeeNumber
```

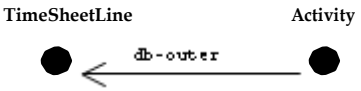
Employee Number	ActivityNumber	ActivityText	outerExists
100	200	Design	true
100	201	Programming	true
100	201	Programming	true
102	string'null	string'null	false



Query 2:

```
mselect Activity.ActivityNumber,
        Activity.ActivityText,
        Timesheetline.EmployeeNumber,
        Timesheetline.LineNumber
from reference::U004
order by Activity.ActivityNumber
```

Activity Number	ActivityText	EmployeeNumber	LineNumber
200	Design	100	1
201	Programming	100	2
201	Programming	100	3
202	Verification	string'null	integer'null

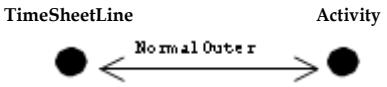


Join Type, NormalOuter

The join type **NormalOuter** is an asymmetric join between two objects X and Y. It defines a database normal join from X to Y, but a database outer join from Y to X. The join is only supplied when fields from both objects are selected.

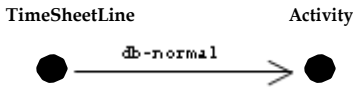
In the following example, the NormalOuter join type is used in a universe to define the join between the two objects TimeSheetLine and Activity. The first query selects the object TimeSheetLine as root, and the second query selects the object Activity as root.

```
<mul 1.3>
<universe reference::U005 "Reference Manual Universe 005">
   <join (TimeSheetLine, Activity) NormalOuter>
     .ActivityNumber
   <end join>
<end universe>
```



Query 1:

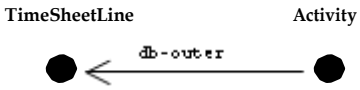
```
mselect Timesheetline.EmployeeNumber,
        Timesheetline.ActivityNumber,
        Activity.ActivityText
from reference::U005
order by Timesheetline.EmployeeNumber
```



Employee Number	ActivityNumber	ActivityText	outerExists
100	200	Design	true
100	201	Programming	true
100	201	Programming	true

Query 2:

```
mselect Activity.ActivityNumber,
        Activity.ActivityText,
        Timesheetline.EmployeeNumber,
        Timesheetline.LineNumber
from reference::U005
order by Activity.ActivityNumber
```



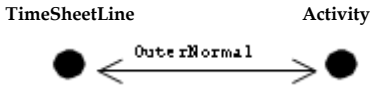
Activity Number	ActivityText	EmployeeNumber	LineNumber
200	Design	100	1
201	Programming	100	2
201	Programming	100	3
202	Verification	string′null	integer′null

Join Type, OuterNormal

The join type OuterNormal is an asymmetric join between two objects X and Y. It defines a database outer join from X to Y, but a database normal join from Y to X. The join is only supplied when fields from both objects are selected.

In the following example, the OuterNormal join type is used in a universe to define the join between the two objects TimeSheetLine and Activity. The first query selects the object TimeSheetLine as root, and the second query selects the object Activity as root.

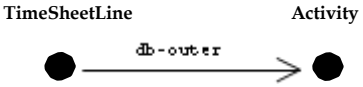
```
<mul 1.3>
<universe reference::U006 "Reference Manual Universe 006">
   <join (TimeSheetLine, Activity) OuterNormal>
     .ActivityNumber
   <end join>
<end universe>
```



Query 1:

```
Mselect Timesheetline.EmployeeNumber,
        Timesheetline.ActivityNumber,
        Activity.ActivityText
From reference::U006
Order by Timesheetline.EmployeeNumber
```

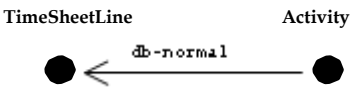
EmployeeNumber	ActivityNumber	ActivityText
100	200	Design
100	201	Programming
100	201	Programming
102	string’null	string’null



Query 2:

```
Mselect Activity.ActivityNumber,
        Activity.ActivityText,
        Timesheetline.EmployeeNumber,
        Timesheetline.LineNumber
From reference::U006
Order by Activity.ActivityNumber
```

Activity Number	ActivityText	EmployeeNumber	LineNumber
200	Design	100	1
201	Programming	100	2
201	Programming	100	3



Parameter Definition

A universe can be parameterized, allowing fields and restrictions to be dependent on values supplied at run time. The current use of parameters is subject to change and should be used with care.

```
parameter ::=
  (
     <parameter (name = id|id) (type = typeid|typeid) [(title = string|string)]
           [value = constExpressionShort|constExpressionShort]
     />
  ) |
  (    id :(type = typeid|typeid) [:(title = string|string)]
          [:(value = constExpressionShort|constExpressionShort)]
  )
```

When a formal parameter is defined, it can be used in expressions just like any other field. If no value is given to the parameter at run time, the default value is used.

Parameters defined in a universe are considered global. This means that if a universe uses an external universe, and the external universe defines a parameter, that parameter is known in the universe.

In the following example, the parameter parmEmployeeNumber is defined and used for parameterization of a restriction.

```
<mul 1.3>
<universe reference::U007 "Reference Manual Universe 007">
  <join (TimeSheetLine, Activity)>
    .ActivityNumber
  <end join>

  <parameter parmEmployeeNumber type=string>

  <restriction>
    TimeSheetLine.EmployeeNumber = parmEmployeeNumber
  <end restriction>

<end universe>
```

Field Definition

New fields can be defined in a universe using the field definition element.

```
field ::=
  fieldExp | fieldValue | fieldBasis | fieldHidden

fieldExp ::=
  <field (name = id|id) [title = string|string] [description =
    string] > expressionShort
  <end field>

fieldValue ::= (
     <field (name = id|id) [title = string|string] [description =
           string] (value =
           constExpressionShort|constExpressionShort)
     />
  ) |
 (   .id [:(title = string|string)] [:description = string]
         (:(value = constExpressionShort|constExpressionShort))
  )

fieldBa
  sis
  ::= (
     <field (name = id|id) [title = string|string] [description =
            string] (basis = qualifiedfieldid|qualifiedfieldid)
     />
  ) |
  (
     .id [:(title = string|string)] [:description =
         string] (:(basis =
         qualifiedfieldid|qualifiedfieldid))
```

The following are attributes of the field definition element.

Attribute	Description
name	The name of the new field.
[title]	The title of the new field.
[description]	A short description of the new field.

Attribute	Description
[value]	The new field is defined as a constant with this value. Used in connection with the short version of the field definition element.
[basis]	The new field is defined as an alias for this field. Used in connection with the short version of the field definition element.

A field is defined by an expression, and the type of the field is inferred from this expression. The Maconomy functions available in the expression are listed in Maconomy Functions.

Two short versions of a field definition are available, one defining a constant, and one defining an alias for another field.

In the following example, three fields are defined in a universe. The first field is defined using the Maconomy function if, the second field is defined using the constant 42, and the third field is defined as an alias for the field Employee.Name1.

```
<mul 1.3>
<universe reference::U008 "Reference Manual Universe 008">
  <object Employee>

  <field fieldFunction>
     if(Employee.SalesEmployee, "Sales person", "Not a sales person")
  <end field>
  .fieldConstant :value=42
  .fieldAlias    :Employee.Name1

<end universe>
```

Restriction Definition

A restriction can be defined in a universe using the restriction definition element.

```
restictionDefinition ::=
  <restriction [required+|required-] >
    expressionShort

  <end restriction>
```

A restriction specified in the universe is only in effect if a field from a selected object is used in the restriction. If the attribute required+ is used, then the restriction is always in effect.

In the following example, a restriction using only fields from the object Activity is defined. The first query only selects fields from the object TimeSheetLine. Because no fields are selected from the object Activity, the restriction is not in effect. The second query selects fields from the object Activity, and therefore the restriction is in effect. The data used in this example is defined in Join Definition.

```
<mul 1.3>
<universe reference::U009 "Reference Manual Universe 009">
  <join (TimeSheetLine, Activity) OuterNormal>
    .ActivityNumber
  <end join>

  <restriction>
```

```
      like( "%mm%", Activity.ActivityText )
    <end restriction>
  <end universe>
```

Query 1:

```
mselect Timesheetline.EmployeeNumber,
        Timesheetline.ActivityNumber
from reference::U009
order by Timesheetline.EmployeeNumber
```

EmployeeNumber	ActivityNumber
100	200
100	201
100	201
102	string′null

Query 2:

```
mselect Timesheetline.EmployeeNumber,
        Timesheetline.ActivityNumber,
        Activity.ActivityText
from reference::U009
order by Timesheetline.EmployeeNumber
```

EmployeeNumber	ActivityNumber	ActivityText
100	201	Programming
100	201	Programming

In the following example, a required restriction using only fields from the object Activity is defined. The query only selects fields from the object TimeSheetLine, but because the restriction is required, the join to the Activity is forced, and the restriction is in effect. The data used in this example is defined in Join Definition.

```
<mul 1.3>
<universe reference::U010 "Reference Manual Universe 010">
  <join (TimeSheetLine, Activity)>
    .ActivityNumber
  <end join>

  <restriction required+>
    like( "%mm%", Activity.ActivityText )
  <end restriction>

<end universe>
```

Query 1:

```
mselect Timesheetline.EmployeeNumber,
        Timesheetline.ActivityNumber
from reference::U010
order by Timesheetline.EmployeeNumber
```

EmployeeNumber	ActivityNumber
100	201
100	201

Interface Definition

An interface is a specification of how users are to see the universe. An interface can be seen as a long list of fields, where the fields can be divided into groups.

```
interface ::=
  <interface [name = id|id] [basis = qualifiedfieldidList] [sumOnAmount(+|-)] /> |
  <interface [name = id|id] [basis = qualifiedfieldidList] [sumOnAmount(+|-)] >
    field | fieldHide | interfaceGroup | interfacegroupHide
  <end interface>

fieldHide ::=
  <field (name = id|id) hidden+ />   |
  .id : hidden+

qualifiedfieldidList ::=
  qualifiedfieldid | [qualifiedfieldid (, qualifiedfieldid)* ]
```

The following are attributes of the interface elements.

Attribute	Description
[name]	The name of the interface. If a name is not assigned to an interface, the interface is considered the default interface.
[basis]	A list of object/interface group names. Fields and groups from each element in the list are added to this interface.
[sumOnAmount]	Indicates that amount fields added to the interface with the basis attribute are wrapped with the group operator SUM. The default value is sumOnAmount- indicating that amount fields are not wrapped.

A universe can contain more than one interface, but a query using a universe can only select fields from one interface at a time. If no interface is specified, the objects and their fields are visible and can be used directly in a query.

All of the fields and subgroups from an object/object-interfacegroup can be copied to the interface with the basis attribute. If two items with the same name are added to the interface, the item from the first-mentioned element in the basis attribute has precedence. If a field is defined using the group operator SUM, the field added to the interface is also defined using the group operator SUM.

Fields and subgroups can be hidden using the special hidden element, which hides a previously defined field/subgroup.

In the following example, an interface is defined taking all of the fields from the objects TimeSheetLine and Activity, except for the VATCode field, which is hidden. The field ActivityNumber exists in both objects, but is taken from TimeSheetLine because it is mentioned first in the list of basis objects.

```
<MUL 1.3>
<universe reference::U011 "Reference Manual Universe 011">
   <join (TimeSheetLine, Activity) OuterNormal>
     .ActivityNumber
   <end join>
   <interface basis = [TimeSheetLine, Activity] >
      .VATCode :hidden+

   <end interface>

   <end universe>
```

In the following example, an interface is defined taking all of the fields from the objects JobHeader and JobEntry. Because the attribute SumOnAmount+ is used, all amount fields in the two objects are added to the interface using the group operator SUM, for example, the field CostPriceInvoiced available in the universe is defined by SUM(JobEntry.CostPriceInvoiced). The field NumberHoursRegistered has the type real, and to turn the field into a SUM –field, an explicit definition is used.

```
<MUL 1.3>
<universe reference::U012 "Reference Manual Universe 012">
   <join (JobHeader, JobEntry) OuterNormal>
     .JobNumber
   <end join>
   <interface basis = [JobHeader, JobEntry] SumOnAmount+>
     <field NumberHoursRegistered>
       SUM(JobEntry.NumberHoursRegistered)
     <end field>
   <end interface>
<end universe>
```

In the following example, an interface is defined taking all of the fields from the previously defined universe reference::U012. All fields are defined using a group operator SUM, that is, all amount fields and the field NumberHoursRegistered are defined in the interface using the group operator SUM.

```
<MUL 1.3>
<universe reference::U013 "Reference Manual Universe 013">
   <Universe MyUniverse basis=reference::U012 />
   ...
   <interface basis=[MyUniverse]/>
<end universe>
```

Interfacegroup Definition

Fields in the interface can be grouped using the interfacegroup element. This can be used for logical grouping of fields.

```
interfaceGroup ::=
  <interfacegroup [name = id|id]
                  [basis = qualifiedfieldidList] [sumOnAmount(+|-)] [title
                  = string|string]
  >
    field | interfaceGroup | interfaceGroupHide
  <end interfacegroup>

interfaceGroupHide ::=
  <interfacegroup [name = id|id] (hidden+|hidden-) />
```

The following are attributes of the interfacegroup elements.

Attribute	Description
name	The name of the group.
[title]	The title of the group.
[basis]	A list of object/interface group names. Fields and groups from each element in the list are added to this group.
[sumOnAmount]	Indicates that amount fields added to the interface with the basis attribute are wrapped with the group operator SUM. The default value is sumOnAmount- indicating that amount fields are not wrapped.

Fields can be copied to the interfacegroup with the basis attribute exactly as it is done in the interface element.

In the following example, three interfaces are given to the same universe. The first interface is a long list of fields, the second simulates the same grouping of fields as defined by the objects, and the third defines a limited interface.

```
<MUL 1.3>
<universe reference::U014 "Reference Manual Universe 014">
   <join (TimeSheetLine, Activity) OuterNormal>
     .ActivityNumber
   <end join>
   <field regtime>
     SUM(timesheetline.numberofday1 + timesheetline.numberofday2)
   <end field>

   -- First interface, long list of fields
   <interface basis = [TimeSheetLine, Activity] >
     .regtime :.regtime
   <end interface>

   -- Second interface, Object grouping of fields
   <interface objectView>
     <interfacegroup TimeSheetLine basis=TimeSheetLine />
     <interfacegroup Activity basis=Activity />
     .regtime :.regtime
   <end interface>

   -- Second interface, limited interface
   <interface limitView>
     <interfacegroup TimeKeys>
       .Empoloyee :TimesheetLine.EmployeeNumber
       .Activity :Activity.ActivityNumber
     <end interfacegroup>

     <interfacegroup TimeNumbers>
       .Monday :timesheetline.numberofday1
       .Thursday :timesheetline.numberofday2
       .weekSum :.regtime
     <end interfacegroup>
   <end interface>
<end universe>
```

Common Syntax Elements

See the *Deltek Maconomy Language Reference MQL*.

Maconomy Functions

A wide range of functions is available for use in expressions. For each function, a type scheme is defined. If the types of the arguments to a function do not match the type scheme, a type error is given.

Note that even if the database is known, database functions are not available.

See the *Deltek Maconomy Language Reference MQL* for a list of functions.

Version History

MUL 1.3

Inheritance of SUM fields from sub-universe when using basis attribute on interface and interfacegroup elements.

New SumOnAmount attribute on the interface and interfacegroup tags, for automatic generation of SUM fields on all amount fields inherited using the basis attribute.

If no interface is defined in a universe, an interface is created to inherit SUM fields. Available objects are created as matching interfacegroups and fields at the top level in the interface.

MUL 1.2

Initial version.

Deltek is the leading global provider of enterprise software and information solutions for government contractors, professional services firms and other project- and people-based businesses. For decades, we have delivered actionable insight that empowers our customers to unlock their business potential. 20,000 organizations and millions of users in over 80 countries around the world rely on Deltek to research and identify opportunities, win new business, recruit and develop talent, optimize resources, streamline operations and deliver more profitable projects. Deltek – Know more. Do more.®

deltek.com