
AIMMS COM Object User's Guide and Reference - AIMMS.Identifier Object

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com

Copyright © 1993–2011 by Paragon Decision Technology B.V. All rights reserved.

Paragon Decision Technology B.V.	Paragon Decision Technology Inc.	Paragon Decision Technology Pte.
Schipholweg 1	500 108th Avenue NE	Ltd.
2034 LS Haarlem	Ste. # 1085	80 Raffles Place
The Netherlands	Bellevue, WA 98004	UOB Plaza 1, Level 36-01
Tel.: +31 23 5511512	USA	Singapore 048624
Fax: +31 23 5511517	Tel.: +1 425 458 4024	Tel.: +65 9640 4182
	Fax: +1 425 458 4025	

Email: info@aimms.com
WWW: www.aimms.com

AIMMS is a registered trademark of Paragon Decision Technology B.V. IBM ILOG CPLEX and sc CPLEX is a registered trademark of IBM Corporation. GUROBI is a registered trademark of Gurobi Optimization, Inc. KNITRO is a registered trademark of Ziena Optimization, Inc. XPRESS-MP is a registered trademark of FICO Fair Isaac Corporation. MOSEK is a registered trademark of Mosek ApS. WINDOWS and EXCEL are registered trademarks of Microsoft Corporation. $\text{T}_{\text{E}}\text{X}$, $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$, and $\text{A}_{\text{M}}\text{S}_{\text{L}}\text{A}_{\text{T}}\text{E}_{\text{X}}$ are trademarks of the American Mathematical Society. LUCIDA is a registered trademark of Bigelow & Holmes Inc. ACROBAT is a registered trademark of Adobe Systems Inc. Other brands and their products are trademarks of their respective holders.

Information in this document is subject to change without notice and does not represent a commitment on the part of Paragon Decision Technology B.V. The software described in this document is furnished under a license agreement and may only be used and copied in accordance with the terms of the agreement. The documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Paragon Decision Technology B.V.

Paragon Decision Technology B.V. makes no representation or warranty with respect to the adequacy of this documentation or the programs which it describes for any particular purpose or with respect to its adequacy to produce any particular result. In no event shall Paragon Decision Technology B.V., its employees, its contractors or the authors of this documentation be liable for special, direct, indirect or consequential damages, losses, costs, charges, claims, demands, or claims for lost profits, fees or expenses of any nature or kind.

In addition to the foregoing, users should recognize that all complex software systems and their documentation contain errors and omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors shall not be responsible under any circumstances for providing information or corrections to errors and omissions discovered at any time in this book or the software it describes, whether or not they are aware of the errors or omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors do not recommend the use of the software described in this book for applications in which errors or omissions could threaten life, injury or significant loss.

This documentation was typeset by Paragon Decision Technology B.V. using $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ and the LUCIDA font family.

Chapter 4

The Aimms.Identifier Object

The Identifier object allows you to access any scalar or multidimensional identifier in the AIMMS project. The Identifier object offers several properties and methods to retrieve and assign data and various identifier characteristics.

Because an identifier only exists within the context of an AIMMS project, you cannot create an Identifier object directly using a standard COM object creator function. Instead, you must create an Identifier object via a specific call to the Aimms.Project object, namely `Project.GetIdentifier`.

Object creation

The following example shows a small Visual Basic program, that opens an AIMMS project, creates an Identifier object, and retrieves the current values for that identifier.

Example

```
Dim MyProject As Aimms.Project
Dim Demand As Aimms.Identifier

'create project object, and open the project file
set MyProject = New Aimms.Project
MyProject.ProjectOpen "C:\Examples\Transport.prj"

'create identifier object for the AIMMS identifier 'demand'
set Demand = MyProject.GetIdentifier( "demand" )

'retrieve the current demand data
CurValues = Demand.CreateArray
```

The Identifier object exposes the following methods and properties:

Methods and Properties

- `Identifier.FullDimension`
- `Identifier.SlicedDimension`
- `Identifier.Default`
- `Identifier.Card`
- `Identifier.Name`
- `Identifier.ReadOnly`
- `Identifier.ElementValuePassMode`
- `Identifier.TuplePassMode`
- `Identifier.Ordinalsoffset`
- `Identifier.Value`

- `Identifier.AssignArray`
- `Identifier.RetrieveArray`
- `Identifier.CreateArray`
- `Identifier.Empty`
- `Identifier.Update`
- `Identifier.Cleanup`
- `Identifier.GetElementRangeSet`
- `Identifier.GetCallDomain`
- `Identifier.ValueNext`
- `Identifier.Reset`
- `Identifier.AssignSparseValues`
- `Identifier.AssignTable`
- `Identifier.RetrieveTable`

Identifier.FullDimension

This property holds the dimension of the identifier according to its declaration in the AIMMS model. Whether the identifier is sliced has no effect on this number. *Property*

Synopsis:

```
Int FullDimension
```

Remarks:

The property `FullDimension` is read-only.

Example:

```
Dim tr As Aimms.Identifier
Set tr = MyProject.GetIdentifier( "transport('Amsterdam',)" )

' the full dimension equals 2
MsgBox tr.FullDimension
```

See also:

The property [Identifier.SlicedDimension](#)

Identifier.SlicedDimension

This property holds the sliced dimension of the identifier as it was specified in the call to `Project.GetIdentifier`. *Property*

Synopsis:

```
Int SlicedDimension
```

Remarks:

The property `SlicedDimension` is read-only.

For most methods of the `Identifier` object that involve element tuples or other dimension aspects, you must handle the object as if it is an identifier with dimension equal to `SlicedDimension`.

Example:

```
Dim tr As Aimms.Identifier
Set tr = MyProject.GetIdentifier( "transport('Amsterdam',)" )

If tr.SlicedDimension < tr.FullDimension Then
    MsgBox "The identifier is sliced."
End If
```

See also:

The property `Identifier.FullDimension`

Identifier.Default

This property holds the default value of the identifier as it is specified in the declaration of the identifier in AIMMS. When you are retrieving sparse data of the identifier, any value that is equal to this default will not be included.

Property

Synopsis:

Variant Default

Remarks:

The property `Default` is read-only and is of type `Variant`. This special data type is used frequently in a COM context and can hold various other data types such as integer, double and string. AIMMS uses this `Variant` type in a similar way to pass values of numerical identifiers, string parameters or element parameters.

Identifier type	ElementValuePassMode	Variant type
Parameter or Variable	-	Double or Integer
String Parameter	-	String
Element Parameter	ELEMENT_BY_NUMBER	Integer
	ELEMENT_BY_ORDINAL	Integer
	ELEMENT_BY_NAME	String

Table 4.1: Variant types used for various AIMMS identifiers

Example:

```
Dim tr As Aimms.Identifier
Set tr = MyProject.GetIdentifier( "transport('Amsterdam',)" )

' usually the default of an identifier equals 0
MsgBox tr.Default
```

See also:

The property `Identifier.SlicedDimension`

Identifier.Card

The property `Card` provides the cardinality of the identifier(-slice). The cardinality of an identifier is defined as the total number of nondefault data values. *Property*

Synopsis:

Int `Card`

Remarks:

The property `Card` is read-only, and its value may change whenever new values are assigned to the identifier.

See also:

The property `Identifier.Default`

Identifier.Name

The property `Name` holds the name of the AIMMS identifier. Any domain or slicing information is excluded from the name. *Property*

Synopsis:

String Name

Remarks:

The property `Name` is read-only.

See also:

The property [Project.GetIdentifier](#)

Identifier.ReadOnly

The property `ReadOnly` indicates whether you are allowed to make modifications to the data of the identifier. If in the AIMMS model the identifier is declared with a `Definition`, or if the identifier is not part of the predefined set `AllUpdatableIdentifiers`, then modifications are not allowed. *Property*

Synopsis:

`Boolean ReadOnly`

Remarks:

You cannot change the value of the property `ReadOnly`.

Identifier.ElementValuePassMode

When you assign or retrieve a value of an element parameter, you are actually passing an element of an AIMMS set. The AIMMS COM interface allows you to specify this element in three different modes:

Property

- as element number (a unique number, which remains unchanged during the AIMMS session)
- as ordinal number (the ordinal position of the element in the corresponding set), or
- as element name (the name of the element as you see it in the model).

With the property `ElementValuePassMode` you can specify how you want to pass the values for the corresponding identifier. Initially, this property inherits its value from `Project.DefaultElementValuePassMode`.

Synopsis:

```
Long ElementValuePassMode
```

Remarks:

This property only applies if the underlying identifier is an element parameter. It can have any of the following defined values:

- `ELEMENT_BY_NUMBER = 0`
- `ELEMENT_BY_ORDINAL = 1`
- `ELEMENT_BY_NAME = 2`

Example:

```
Dim CurCity As Aimms.Identifier
Dim OrdNumber As Integer
Set CurCity = MyProject.GetIdentifier( "CurrentCity" )

' assigning a value using the element name
CurCity.ElementValuePassMode = ELEMENT_BY_NAME
CurCity.Value = "Amsterdam"

' retrieving the same value as an ordinal number
CurCity.ElementValuePassMode = ELEMENT_BY_ORDINAL
OrdNumber = CurCity.Value
```

Identifier.TuplePassMode

When assigning or retrieving values from indexed identifiers you must communicate tuple(s) of elements. The AIMMS COM interface allows you to specify these tuples in three different modes:

Property

- as element numbers (unique numbers, which remain unchanged during the AIMMS session)
- as ordinal numbers (the ordinal position of the element in the corresponding set), or
- as element names (the name of the element as you see it in the model).

With the property `TuplePassMode` you can specify how you want to pass the values for the corresponding identifier. Initially, this property inherits its value from `Project.DefaultTuplePassMode`.

Synopsis:

```
Long TuplePassMode
```

Remarks:

The property `TuplePassMode` is integer valued and can have any of the following defined values:

- `ELEMENT_BY_NUMBER = 0`
- `ELEMENT_BY_ORDINAL = 1`
- `ELEMENT_BY_NAME = 2`

Example:

```
Dim CurCity As Aimms.Identifier
Dim tuple(1) As String
Set CurCity = MyProject.GetIdentifier( "Transport" )

tuple(0) = "Amsterdam"
tuple(1) = "Rotterdam"
CurCity.TuplePassMode = ELEMENT_BY_NAME
CurCity.Value( tuple ) = 25.0
```

Identifier.Ordinalsoffset

When passing set elements, the AIMMS COM allows you to reference these elements by their ordinal position within the set. With the property `Ordinalsoffset` you can indicate whether the first element in the set is referenced as ordinal number 0 or ordinal number 1.

Property

Synopsis:

```
Int Ordinalsoffset
```

Remarks:

This property has an effect on:

- the passing of element tuples, if the property `TuplePassMode` is set to `ELEMENT_BY_ORDINAL`, and
- the passing of element parameter values, if the property `ElementValuePassMode` is set to `ELEMENT_BY_ORDINAL`.

See also:

The properties `Identifier.TuplePassMode` `Identifier.ElementValuePassMode`.

Identifier.Value

With the property `Value` you can set or retrieve a single scalar value in the identifier. If the underlying identifier is not a scalar of itself, you must specify the element tuple that you want to access. *Property*

Synopsis:

```
Variant Value(
    [tuple]    ! (optional input) Variant array (String/Integer)
)
```

Arguments:

tuple (optional)

This array specifies the element tuple that you want to access (for data retrieval or assignment). The size of the array should match with the `SlicedDimension` of the identifier. Depending on the property `TuplePassMode`, this array should contain integers or strings. If `SlicedDimension` equals 0, then you may omit this argument.

Return value:

The value is given as a `Variant`. Table 4.1 illustrates how this data type is used in combination with the various AIMMS identifier types.

Example:

```
Dim CurCity As Aimms.Identifier
Dim TotCost As Aimms.Identifier
Dim tuple(1) As String

' assigning a single value in a 2-dimensional identifier
Set CurCity = MyProject.GetIdentifier( "Transport" )
tuple(0) = "Amsterdam"
tuple(1) = "Rotterdam"
CurCity.TuplePassMode = ELEMENT_BY_NAME
CurCity.Value( tuple ) = 25.0

' getting the value of a scalar identifier
Set TotCost = MyProject.GetIdentifier( "TotalCost" )
x = TotCost.Value
```

See also:

The properties `Identifier.TuplePassMode` and `Identifier.SlicedDimension`.

Identifier.AssignArray

With this method you can assign a (multi-dimensional) array of values to the identifier. The data values in the array will completely overwrite the current values of the identifier(-slice).

Synopsis:

```
AssignArray(
    value_array,      ! (input) Variant array (double/string/integer)
    [transposed]     ! (optional) Boolean
)
```

Arguments:

value_array

The array containing the new values for the identifier. The array dimensions should match the *SlicedDimension* of the identifier. The type of values in the array depend on the type of the identifier. For example: an array of string values for a string parameter, an array of doubles for a numeric parameter, and for an element parameter a type that matches the *ElementValuePassMode*. In all cases the array may be provided as an array of *Variants*. For example, for a string parameter you can either provide an array of strings, or an array of *Variants* containing strings.

transposed (optional)

This argument determines whether the method expects the values for a multi-dimensional identifiers in row-wise or column-wise order. For the default value (*FALSE*), values must be passed in row-wise order.

Example:

```
Dim id As Aimms.Identifier
Dim random_vals(3,3) As Variant
Set id = MyProject.GetIdentifier( "Transport" )
For i=0 To 3
    For j=0 To 3
        random_vals(i,j) = Rnd
    Next j
Next i
id.AssignArray random_vals
```

See also:

The methods [Project.AssignArray](#)

Identifier.RetrieveArray

With this method you can retrieve an array of values from the identifier.

Synopsis:

```
RetrieveArray(
    value_array,      ! (output) Variant array (double/string/integer)
    [sparse],        ! (optional) Boolean
    [transposed]     ! (optional) Boolean
)
```

Arguments:

value_array

The array, that on return contains the current values of the identifier. The array dimensions should match the (sliced) dimension of the identifier. The type of values in the array depend on the type of the identifier. For example: an array of string values for a string parameter, an array of doubles for a numeric parameter, and for an element parameter a type that matches the `DefaultElementValuePassMode`. You may also provide an array of Variants, in that case AIMMS will fill these Variants with the corresponding data type.

sparse (optional)

If this argument is set to TRUE, and the *value_array* is an array of Variant values, then when the identifier has default values, these elements are not passed as the default value, but as an 'empty' variant. This is especially usefull in spreadsheets, so that default values are not shown as zeros or empty strings, but as empty cells.

transposed (optional)

This argument determines whether the method fills the array for a multi-dimensional identifiers in row-wise or column-wise order. For the default value (FALSE), values are passed according to a row-wise order.

Example:

```
'Retrieving an array using the Identifier object
Dim id As Aimms.Identifier
Dim CurVals(3,3) As Variant
Set id = MyProject.GetIdentifier( "Transport" )
id.RetrieveArray CurVals
```

See also:

The methods [Project.RetrieveArray](#)

Identifier.CreateArray

This method creates and retrieves an array with the current values of the identifier. The method is similar to `RetrieveArray` except that `CreateArray` does not require you to provide the value array. Instead the array is allocated by the method itself, and returned as the return value of the method.

Synopsis:

```
VariantArray CreateArray(
    [sparse],           ! (optional) Boolean
    [transposed]       ! (optional) Boolean
)
```

Arguments:

sparse (optional)

If this argument is set to `TRUE`, then when the identifier has default values, these elements are not passed as the default value, but as an 'empty' variant. This is especially useful in spreadsheets, so that default values are not shown as zeros or empty strings, but as empty cells.

transposed (optional)

This argument determines whether the method fills the array for a multi-dimensional identifiers in row-wise or column-wise order. For the default value (`FALSE`), values are passed according to a row-wise order.

Return value:

A newly created array of Variants, containing the current values of the identifier. The dimension and sizes of the created array exactly match with the sliced dimension of the identifier and the cardinality of the domain sets. Whether the first element of the array starts at position 0 or 1 depends on the value of the property `Project.ArrayIndexOffset`.

Example:

```
'Retrieving an array using the Identifier object
Dim id As Aimms.Identifier
Dim CurVals() As Variant
Set id = MyProject.GetIdentifier( "Transport" )
CurVals = id.CreateArray
' Now, CurVals is a 2-dimensional array
```

See also:

The methods `Project.CreateArray` and `Identifier.RetrieveArray`

Identifier.Empty

This method empties the identifier(slice). In other words, all values are set to the default value of the identifier. Afterwards, the cardinality of the identifier is 0.

Synopsis:

Empty

Arguments:

None

See also:

The properties [Identifier.Card](#) and [Identifier.Default](#)

Identifier.Update

This method updates the values of the *entire* identifier, regardless whether the object refers to only a slice of the data.

Synopsis:

Update

Arguments:

None

See also:

The UPDATE statement

Identifier.Cleanup

This method cleans up the data of the *entire* identifier, regardless whether the object refers to only a slice of the data. The cleanup operation removes all inactive data of the identifier, i.e. the data elements that are no longer part of the domain of the identifier.

Synopsis:

Cleanup

Arguments:

None

See also:

The CLEANUP statement

Identifier.GetElementRangeSet

This method creates a new *Set* object for the range set of an element parameter.

Synopsis:

```
Aimms.Set GetElementRangeSet
```

Arguments:

None

Return value:

A newly created *Aimms.Set* object, that refers to the range set of the element parameter.

Remarks:

This method only applies if the underlying identifier is an element parameter in AIMMS.

See also:

The *Aimms.Set* object (see [Chapter 5](#)).

Identifier.GetCallDomain

This method returns a *Set* object for the *n*-th free domain index of the identifier.

Synopsis:

```
Aimms.Set GetCallDomain(  
    n      ! (input) Integer  
)
```

Arguments:

n

The sequence number of the domain index for which you want to create a *Set* object. The numbering starts at 1, and only the indices in the sliced domain are counted.

Return value:

A newly created *Aimms.Set* object, that refers to the set of the *n*-th sliced domain index.

Example:

```
Dim id As Aimms.Identifier  
Dim s1 As Aimms.Set  
Dim s2 As Aimms.Set  
  
'create a 2-dimensional slice of parameter x  
set id = MyProject.GetIdentifier( "x(i,'j0',k)" )  
  
'get a Set object for index i  
set s1 = id.GetCallDomain( 1 )  
  
'get a Set object for index k (the second dimension in the slice)  
set s2 = id.GetCallDomain( 2 )
```

See also:

The *Aimms.Set* object (see Chapter 5).

Identifier.ValueNext

This method retrieves the next non-default value of the identifier. The method `ValueNext` can be used sequentially to retrieve all the non-default values of the identifier.

Synopsis:

```
Boolean ValueNext(
    tuple_array,    ! (output) Variant array (String/Integer)
    value          ! (output) Variant (Double/String/Integer)
)
```

Arguments:

tuple_array

The size of this array must match with the sliced dimension of the identifier. The type of the array elements depend on the property `TuplePassMode` and is either a string or an integer. If the method successfully retrieves the next value, then this array is filled with the corresponding tuple.

value

If the next value exists, it is returned in this argument. The type of Variant depends on the type of the identifier. See also Table 4.1.

Return value:

`ValueNext` returns `TRUE` if there is a next value, otherwise it returns `FALSE`. In the latter case, the contents of both arguments `tuple_array` and `value` is undefined.

Remarks:

To access all non default values of an identifier, you first need to call the method `Reset` and then call `ValueNext` repeatedly, until it returns `FALSE`.

Example:

```
Dim id As Aimms.Identifier
Dim tuple(1) As Variant
Dim t As Variant
Set id = MyProject.GetIdentifier( "Transport" )
id.TuplePassMode = ELEMENT_BY_NAME
id.Reset
exists = id.ValueNext( tuple, t )
While exists
    MsgBox "(" + tuple(0) + "," + tuple(1) + ")=" + Str(t)
    exists = id.ValueNext( tuple, t )
Wend
```

See also:

The methods `Project.AssignArray`

Identifier.Reset

This method resets the cursor, that is used internally in the method `ValueNext`. After a call to `Reset`, the first call to `ValueNext` will return the first non-default value in the identifier(slice), or return `FALSE` if the identifier(slice) is empty.

Synopsis:

Reset

Arguments:

None

See also:

The method `Identifier.ValueNext`

Identifier.AssignSparseValues

With this method you can assign values for a number of explicitly given element tuples. The element tuples that are not referenced keep their current values.

Synopsis:

```
AssignSparseValues(  
    nr,                ! (input) integer  
    tuple_array,      ! (input) Variant array (String/Integer)  
    value_array       ! (input) Variant array (Double/String/Integer)  
)
```

Arguments:

nr

The number of values that you want to assign.

tuple_array

A two-dimensional array of size: $nr \times SlicedDimension$. The array should contain *nr* tuples. Depending on the value of `TuplePassMode`, the elements in these tuples are passed as integers or strings.

value_array

A array of size *nr*. For each tuple defined in *tuple_array*, *value_array* holds the corresponding new value. The type of this value depends on the type of the identifier.

See also:

The properties `Identifier.Value` and `Identifier.AssignArray`

Identifier.AssignTable

With this method you can assign values to the identifier using a tabular format of rows and columns. For both the rows and columns you specify the (tuples of) elements for which you want to assign the data.

Synopsis:

```
AssignTable(
    row_tuples,      ! (input) Variant array (String/Integer)
    column_tuples,  ! (input) Variant array (String/Integer)
    value_array     ! (input) Variant array (Double/String/Integer)
)
```

Arguments:

row_tuples

A two-dimensional matrix of size $m \times r$. In this matrix every row contains a tuple of elements (or a single element in case $r = 1$).

column_tuples

A two-dimensional matrix of size $c \times n$. In this matrix every column contains a tuple of elements (or a single element in case $c = 1$).

value_array

A two-dimensional array of size $m \times n$, containing the new values for the identifier.

Remarks:

In the simplest form the sliced dimension of the identifier is 2, so that both the rows and columns are represented by single elements. The first index is presented in the rows and the second index in the columns. In that case both r and c are 1 and the tuples arrays can be seen as 1-dimensional arrays. The table below illustrates this for the case where $m=3$ and $n=4$:

	(1 × 4)			
	j1	j2	j3	j4
i1	11	12	13	14
i2	21	22	23	24
i3	31	32	33	34
(3 × 1)	(3 × 4)			

In a general situation the sliced dimension is split in two parts: the first r indices are presented in the rows, and the remaining c indices in the columns. The example below illustrates this for a sliced dimension of 4, both r and c are 2, and $m=3$ and $n=4$:

(2 × 4)			
k1	k1	k3	k4
l1	l2	l3	l1

i1	j1	11.11	11.12	11.33	11.41
i2	j3	23.11	23.12	23.33	23.41
i2	j4	24.11	24.12	24.33	24.41
(3 × 2)		(3 × 4)			

If $r = 1$ or $c = 1$, the `row_tuples` resp. `column_tuples` does not have to be a 2-dimensional matrix. Instead you can specify it as a 1-dimensional array.

See also:

The methods `Project.AssignTable Identifier.RetrieveTable`

Identifier.RetrieveTable

With this method you can retrieve the values of the identifier using a tabular format of rows and columns. For both the rows and columns you pass matrices that (on return) contain the element tuples. You can either specify which rows and/or columns you want to retrieve, or let the method itself fill in the rows and columns.

Synopsis:

```
RetrieveTable(
    row_tuples,      ! (input or output) Variant array (String/Integer)
    column_tuples,  ! (input or output) Variant array (String/Integer)
    value_array,    ! (output) Variant array (Double/String/Integer)
    [sparse],       ! (optional) Boolean
    [row_mode],     ! (optional) Integer
    [column_mode]   ! (optional) Integer
)
```

Arguments:

row_tuples

A two-dimensional matrix of size $m \times r$. In this matrix every row contains a tuple of elements (or a single element in case $r = 1$).

column_tuples

A two-dimensional matrix of size $c \times n$. In this matrix every column contains a tuple of elements (or a single element in case $c = 1$).

value_array

A two-dimensional array of size $m \times n$, to hold the values of the identifier. The type of values in the array depend on the type of the identifier. For example: an array of string values for a string parameter, an array of doubles for a numeric parameter, and for an element parameter a type that matches the `DefaultElementValuePassMode`. You may also provide an array of Variants, in that case AIMMS will fill these Variants with the corresponding data type.

sparse (optional)

If this argument is set to TRUE, and the `value_array` is an array of Variant values, then when the identifier has default values, these elements are not passed as the default value, but as an 'empty' variant. This is especially useful in spreadsheets, so that default values are not shown as zeros or empty strings, but as empty cells.

row_mode

An integer that indicates how the `row_tuples` array should be interpreted. The value should be one of the table below.

column_mode

An integer that indicates how the `col_tuples` array should be interpreted. The possible values are the same as for `row_mode`.

SPARSE_OUTPUT	0	In this mode the tuples array (row or column) is regarded as output, and is thus filled by the method. The method will search for non-default values and only fill the array with tuples for which a non-default value exists.
DENSE_OUTPUT	1	In this mode the tuples array (row or column) is regarded as output, and is thus filled by the method. The method will fill the tuples array with all possible tuples, using the entire content of the domain sets.
USER_INPUT	2	In this mode the tuples array (row or column) is regarded as input, and the method will only retrieve values for the tuples that are given in this array.
NON_EXISTING	3	If the <code>row_mode</code> is set to <code>NON_EXISTING</code> , then the entire element tuple should be given in the <code>column_tuples</code> array (and vice-versa).

Table 4.2: Values for `row_mode` and `column_mode`**Remarks:**

The dimensions of the three arrays `row_tuples`, `column_tuples` and `value_array` should match in a similar way as for `Identifier.AssignTable`.

See also:

The methods `Project.RetrieveTable` `Identifier.AssignTable`