

---

**AIMMS COM Object User's Guide and Reference - AIMMS.Set 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](http://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](mailto:info@aimms.com)  
WWW: [www.aimms.com](http://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 5

### The Aimms.Set Object

The Set object allows you to access a set in the AIMMS project. The Set object offers several properties and methods to retrieve or modify the contents of the set.

Because a set only exists within the context of an AIMMS project, you cannot create a Set object directly using a standard COM object creator function. Instead, you must create a Set object via a specific call to the Aimms.Project object, namely Project.GetSet. In addition, you can create a Set object via methods of the Identifier object, namely GetCallDomain and GetElementRangeSet.

*Object creation*

The following example shows a small Visual Basic program, that opens an AIMMS project, creates a Set object, and retrieves the current contents of that set.

*Example*

```
Dim MyProject As Aimms.Project
Dim Cities As Aimms.Set

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

'create set object for the AIMMS identifier 'Cities'
set Cities = MyProject.GetSet( "Cities" )

'retrieve the current set content
AllCities = Cities.CreateElementArray
```

The Set object exposes the following methods and properties:

*Methods and Properties*

- Set.Card
- Set.Dimension
- Set.ElementPassMode
- Set.Name
- Set.Ordinalsoffset
- Set.ReadOnly
- Set.AddElement
- Set.AssignElementArray
- Set.RetrieveElementArray

- `Set.CreateElementArray`
- `Set.DeleteElement`
- `Set.ElementToName`
- `Set.ElementToOrdinal`
- `Set.OrdinalToName`
- `Set.OrdinalToElement`
- `Set.NameToElement`
- `Set.NameToOrdinal`
- `Set.RenameElement`
- `Set.CompoundAddElementTuple`
- `Set.CompoundDimension`
- `Set.CompoundElementToTuple`
- `Set.CompoundTupleToElement`
- `Set.CompoundTuplePassMode`

---

**Set.Card**

The property `Card` provides the cardinality of the set. The cardinality of a set is defined as the total number of elements in the set. *Property*

**Synopsis:**

`Int Card`

**Remarks:**

The property `Card` is read-only. Its value changes whenever elements are added to or removed from the set.

---

**Set.Dimension**

The dimension of a set is defined as follows:

*Property*

- the dimension of a simple set is 1,
- the dimension of a relation is the dimension of the Cartesian product of which the relation is a subset
- the dimension of a compound set is 1, and
- the dimension of an indexed set is the dimension of the index domain of the set plus 1.

**Synopsis:**

Int Dimension

---

## Set.ElementPassMode

The AIMMS COM interface allows you to reference set elements 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 elements as you see it in the model).

With the property `ElementPassMode` you can specify how you want to pass the elements of the corresponding set. Initially, this property inherits its value from `Project.DefaultElementValuePassMode`.

### Synopsis:

```
Long ElementPassMode
```

### Remarks:

The property `ElementPassMode` 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 Cities As Aimms.Set
Set Cities = MyProject.GetIdentifier( "Cities" )

Cities.TuplePassMode = ELEMENT_BY_NAME
Cities.AddElement "Amsterdam"
```

---

**Set.Name**

The property `Name` holds the name of the AIMMS set.

*Property*

**Synopsis:**

String `Name`

**Remarks:**

The property `Name` is read-only.

**See also:**

The property `Project.GetSet`

---

## Set.OrdinalOffset

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

### Synopsis:

```
Int OrdinalOffset
```

### Remarks:

This property has an effect on:

- the passing of elements, if `ElementPassMode` equals `ELEMENT_BY_ORDINAL`, and
- the conversion of elements between number, ordinal and string representations.

### Example:

```
Dim Cities As Aimms.Set
Dim Name0, Name1 As String
Set Cities = MyProject.GetIdentifier( "Cities" )

Cities.OrdinalOffset = 0
Name0 = Cities.OrdinalToName( 0 )

Cities.OrdinalOffset = 1
Name1 = Cities.OrdinalToName( 1 )

' Name0 and Name1 are the same
MsgBox Name0 + "=" + Name1
```

### See also:

The methods `Set.OrdinalToName`, `Set.OrdinalToElement`, `Set.NameToOrdinal`, `Set.ElementToOrdinal` and `Set.ElementPassMode`.

---

**Set.ReadOnly**

The property `ReadOnly` indicates whether you are allowed to make modifications to the contents of the set. If in the AIMMS model the set is declared with a `Definition`, or if the set 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`.

---

## Set.AddElement

With the method `AddElement` you can add single elements to the set.

### Synopsis:

```
AddElement(  
    element_name,      ! (input) String  
    [recursive]       ! (optional) Boolean  
)
```

### Arguments:

*element\_name*

The name of the element that you want to add to the set.

*recursive (optional)*

This boolean indicates whether the method should automatically add the new element to each of its super sets. If you set this to `FALSE`, then if you add an element to a subset, this element must already exist in the superset.

### Remarks:

This function will fail when the element to be added is already in the set.

### See also:

The methods [Set.DeleteElement](#)

---

## Set.AssignElementArray

With this method you can fill the set with (new) elements. Optionally you can replace the entire existing contents of the set with the new elements, or merge the new elements with the existing elements.

### Synopsis:

```
AssignElementArray(  
    element_array,    ! (input) Variant array (String/Integer)  
    [mode]            ! (optional) Long  
)
```

### Arguments:

#### *element\_array*

An array containing the names of the elements that must be added to the set.

#### *mode (optional)*

Possible values: REPLACE or MERGE. If mode is set to REPLACE (the default setting), then existing elements in the set that are not in the given array are removed from the set, and new elements are added. If mode is set to MERGE, then all the new elements in the array are added to the existing elements.

### Example:

```
Dim Cities As Aimms.Set  
Dim DutchCities(3) As String  
Set Cities = MyProject.GetSet( "Cities" )  
DutchCities(0) = "Amsterdam"  
DutchCities(1) = "Rotterdam"  
DutchCities(2) = "The Hague"  
DutchCities(3) = "Utrecht"  
Cities.AssignElementArray DutchCities
```

### See also:

The methods [Set.AddElement](#)

---

## Set.RetrieveElementArray

With this method you can retrieve an array containing all the elements in a set.

### Synopsis:

```
RetrieveElementArray(  
    element_array      ! (output) Variant array (String/Integer)  
)
```

### Arguments:

*element\_array*

An array, that on output contains references to all the elements in the set. Whether the array is of type String or Integer depends on the property `ElementPassMode`.

### Example:

```
Dim Cities As Aimms.Set  
Dim CurrCities() As String  
Set Cities = MyProject.GetSet( "Cities" )  
ReDim CurrCities( Cities.Card-1 )  
Cities.ElementPassMode = ELEMENT_BY_NAME  
Cities.RetrieveElementArray CurrCities
```

### See also:

The method [Set.CreateElementArray](#)

---

## Set.CreateElementArray

This method creates and retrieves an array with the current elements of the set. The method is similar to `RetrieveElementArray` except that `CreateElementArray` 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 CreateElementArray
```

### Arguments:

*None*

### Return value:

A newly created array of `Variants`, containing the current elements of the set. The size of the created array exactly matches with the cardinality of set. Whether the first element of the array starts at position 0 or 1 depends on the value of the property `Project.ArrayIndexOffset`.

### Example:

```
Dim Cities As Aimms.Set  
Dim CurrCities() As Variant  
Set Cities = MyProject.GetSet( "Cities" )  
Cities.ElementPassMode = ELEMENT_BY_NAME  
CurrCities = Cities.CreateElementArray
```

### See also:

The methods [Set.RetrieveElementArray](#)

---

## Set.DeleteElement

With the method `DeleteElement` you can delete an element from the set.

### Synopsis:

```
DeleteElement(  
    element_ref      ! (input) Variant (String/Integer)  
)
```

### Arguments:

*element\_ref*

A reference to the element that you want to delete. The type of this argument depends on the current value of `ElementPassMode` and is either a string or integer.

### Remarks:

Besides removing a single element using the method `DeleteElement`, you can implicitly remove elements using the method `AssignElementArray` in `REPLACE` mode.

### See also:

The methods `Set.AddElement`, `Set.AssignElementArray`, `Set.ElementPassMode`.

---

## Set.ElementToName

With the method `ElementToName` you can convert from an element number to the name of that set element.

### Synopsis:

```
String ElementToName(  
    name      ! (input) String  
)
```

### Arguments:

*number*  
The unique element number.

### Return value:

`ElementToName` returns the name of the given element.

### See also:

The methods `Set.NameToElement`, `Set.NameToOrdinal`, `Set.ElementToOrdinal`, `Set.OrdinalToElement` and `Set.OrdinalToName`.

---

## Set.ElementToOrdinal

With the method `ElementToOrdinal` you can convert from the ordinal number of a set element to its element number. In other words, you can retrieve the element number of the *n*-th element. The element number is a unique number of the set element, which remains unchanged during the AIMMS session.

### Synopsis:

```
Integer ElementToOrdinal(  
    number          ! (output) Integer  
)
```

### Arguments:

*number*  
The element number of a set element

### Return value:

`ElementToOrdinal` returns the ordinal number of the given element.

### Remarks:

The conversion from element to ordinal number uses the current value of the property `OrdinalsOffset`.

### See also:

The methods `Set.OrdinalToElement`, `Set.OrdinalToName`, `Set.ElementToName`, `Set.NameToOrdinal`, `Set.NameToElement`, and `Set.OrdinalsOffset`.

---

## Set.OrdinalToName

With the method `OrdinalToName` you can convert from the ordinal number of a set element to its element name. In other words, you can retrieve the name of the n-th element.

### Synopsis:

```
String OrdinalToName(  
    ordinal      ! (input) Integer  
)
```

### Arguments:

*ordinal*

The ordinal number of an element.

### Return value:

`OrdinalToName` returns the name of the given element.

### Remarks:

The conversion from ordinal number to name uses the current value of the property `Ordinalsoffset`.

### See also:

The methods `Set.NameToOrdinal`, `Set.NameToElement`, `Set.OrdinalToElement`, `Set.ElementToName`, `Set.ElementToOrdinal` and `Set.Ordinalsoffset`.

---

## Set.OrdinalToElement

With the method `OrdinalToElement` you can convert from the ordinal number of a set element to its element number. In other words, you can retrieve the element number of the n-th element. The element number is a unique number of the set element, which remains unchanged during the AIMMS session.

### Synopsis:

```
Integer OrdinalToElement(  
    ordinal    ! (input) Integer  
)
```

### Arguments:

*ordinal*  
The ordinal number of an element.

### Return value:

`OrdinalToElement` returns the unique element number for the given ordinal number.

### Remarks:

The conversion from ordinal to element number uses the current value of the property `OrdinalsOffset`.

### See also:

The methods `Set.ElementToOrdinal`, `Set.ElementToName`, `Set.OrdinalToName`, `Set.NameToOrdinal`, `Set.NameToElement`, and `Set.OrdinalsOffset`.

---

## Set.NameToElement

With the method `NameToElement` you can convert from the name of a set element to its element number.

### Synopsis:

```
Integer NameToElement(  
    name          ! (input) String  
)
```

### Arguments:

*name*  
The name of the set element.

### Return value:

`NameToElement` returns the unique element number of the given element name.

### See also:

The methods `Set.ElementToName`, `Set.ElementToOrdinal`, `Set.NameToOrdinal`, `Set.OrdinalToElement` and `Set.OrdinalToName`.

---

## Set.NameToOrdinal

With the method `NameToOrdinal` you can convert from the name of a set element to the ordinal number of the element.

### Synopsis:

```
Integer NameToOrdinal(  
    name          ! (input) String  
)
```

### Arguments:

*name*  
The name of the set element.

### Return value:

`NameToOrdinal` returns the ordinal number of the given element.

### Remarks:

The conversion from name to ordinal number uses the current value of the property `Ordinalsoffset`.

### See also:

The methods `Set.OrdinalToName`, `Set.OrdinalToElement`, `Set.NameToElement`, `Set.ElementToName`, `Set.ElementToOrdinal` and `Set.Ordinalsoffset`.

---

## Set.RenameElement

With the method `RenameElement` you can rename an element in the set.

### Synopsis:

```
RenameElement(  
    element_ref,    ! (input) Variant (String/Integer)  
    new_name,      ! (input) String  
)
```

### Arguments:

*element\_ref*

A reference to the element that you want to rename. The type of this argument depends on the current value of `ElementPassMode` and is either a string or integer.

*new\_name*

The new name for the element.

---

## Set.CompoundAddElementTuple

The method `CompoundAddElementTuple` adds a new element to a compound set. The new element is given as a tuple of elements.

### Synopsis:

```
CompoundAddElementTuple(
    tuple_array,      ! (input) Variant array (String/Integer)
    number,          ! (output) Integer
    [recursive]     ! (optional) Boolean
)
```

### Arguments:

#### *tuple\_array*

An array with a size equal to `CompoundDimension`. This array contains the tuple of individual elements that you want to add to the compound set. The type of the array should match with `CompoundTuplePassMode` and is either integer or string.

#### *number*

On return this arguments holds the element number of the newly added compound element.

#### *recursive (optional)*

If this argument is set to `TRUE`, the new compound element is automatically added to all supersets of the compound set. If set to `FALSE`, the element is only added to the set itself. In the latter case the set must either be a root compound set or the element must already exist in the superset.

### Remarks:

The method `CompoundAddElementTuple` only applies to compound sets in the AIMMS model. The method returns with an error if you call this function for other type of sets.

### Example:

```
Dim routes As Aimms.Set
Dim newtuple(1) As Variant
Dim newelement As Variant
set routes = MyProject.GetSet( "routes" )
newtuple(0) = "Amsterdam"
newtuple(1) = "Rotterdam"
routes.TuplePassMode = ELEMENT_BY_NAME
routes.ElementPassMode = ELEMENT_BY_NAME
routes.CompoundAddTupleElement newtuple, newelement
MsgBox "Added: " + newelement
' result: "Added: ('Amsterdam','Rotterdam')"
```

**See also:**

The methods `Set.CompoundTupleToElement`, `Set.CompoundDimension`

---

**Set.CompoundDimension**

The property `Dimension` of a compound set is per definition equal to 1. If you want to retrieve the dimension of a compound set as a relation, then you must use the special property `CompoundDimension`.

*Property*

**Synopsis:**

```
Int CompoundDimension
```

**Remarks:**

The property `CompoundDimension` is read-only and only applies to compound sets in the AIMMS model. For other sets, this property will always return 0.

---

## Set.CompoundElementToTuple

The method `CompoundElementToTuple` converts a compound element to the corresponding tuple of domain set elements.

### Synopsis:

```
CompoundElementToTuple(  
    elem,          ! (input) Variant (String/Integer)  
    tuple_array   ! (output) Variant array (String/Integer)  
)
```

### Arguments:

*elem*

A reference to an element in the compound set. The type of this argument depends on the property `ElementPassMode` and is either a string or integer.

*tuple\_array*

An array with a size equal to `CompoundDimension`. On output this array contains the tuple of individual domain set elements. The type of the array matches with `CompoundTuplePassMode` and is either integer or string.

### Remarks:

The method `CompoundElementToTuple` only applies to compound sets in the AIMMS model. You will get an error if you call this function for other type of sets.

### See also:

The methods [Set.CompoundTupleToElement](#)

---

## Set.CompoundTupleToElement

The method `CompoundTupleToElement` converts a tuple of elements to the corresponding element in a compound set.

### Synopsis:

```
CompoundTupleToElement(  
    tuple_array,    ! (input) Variant array (String/Integer)  
    elem,          ! (output) Variant (String/Integer)  
)
```

### Arguments:

*tuple\_array*

An array with a size equal to `CompoundDimension`. This array contains the tuple of individual elements that you want to convert. The type of the array should match with `CompoundTuplePassMode` and is either integer or string.

*elem*

On successful return this argument holds a reference to an existing element in the compound set. The type of this argument depends on the property `ElementPassMode` and is either a string or integer.

### Remarks:

The method `CompoundTupleToElement` only applies to compound sets in the AIMMS model. The method returns with an error if you call this function for other type of sets.

### See also:

The methods [Set.CompoundElementToTuple](#)

---

## Set.CompoundTuplePassMode

The AIMMS COM interface allows you to reference set elements 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 elements as you see it in the model).

With the property `CompoundTuplePassMode` you can specify how you want to pass the domain set elements of a compound tuple. Initially, this property inherits its value from `Project.DefaultTuplePassMode`.

### Synopsis:

Long `CompoundTuplePassMode`

### Remarks:

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

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

The property `CompoundTuplePassMode` only applies to compound sets in the AIMMS model. For other sets, changing this property will not have any effect.