
AIMMS COM Object User's Guide and Reference - AIMMS.Project 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 $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 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 3

The Aimms.Project object

The Project object (or Aimms.Project), is the main object that you must create to access an AIMMS application. The object allows you to: *Project*

- open and close an existing AIMMS application,
- access the identifiers in your model for retrieving and/or assigning data, and
- run model procedures.

The Project object is registered as a real COM object, in the sense that you can create it via a call to a standard object creator function (for example through the New operator, or the CreateObject procedure in Visual Basic). The other three objects in the AIMMS COM interface Identifier, Set and Procedure) can only be created via specific calls to the Project object. This may seem odd at first, but these objects correspond directly to model identifiers, and thus can only exist within the context of a Project object. *Object creation*

The following example shows a small Visual Basic program, that opens an AIMMS project, runs the procedure MainExecution and then closes the project. *Example*

```
set MyProject = CreateObject( "Aimms.Project" )
MyProject.ProjectOpen "C:\Examples\Transport.prj"
MyProject.Run "MainExecution"
MyProject.ProjectClose 0
```

This implementation assumes late binding, as in VBScript. Using early binding, the example could be implemented equivalently as follows.

```
Dim MyProject as Aimms.Project

set MyProject = New Aimms.Project
MyProject.ProjectOpen "C:\Examples\Transport.prj"
MyProject.Run "MainExecution"
MyProject.ProjectClose 0
```

The Project object exposes the following methods and properties:

*Methods and
Properties*

- Project.StartupMode
- Project.User
- Project.Data
- Project.LicenseServer
- Project.ConfigDir
- Project.ProjectOpen
- Project.ProjectClose
- Project.Name
- Project.ArrayIndexOffset
- Project.DefaultTuplePassMode
- Project.DefaultElementValuePassMode
- Project.AssignArray
- Project.RetrieveArray
- Project.CreateArray
- Project.AssignElementArray
- Project.RetrieveElementArray
- Project.CreateElementArray
- Project.AssignTable
- Project.RetrieveTable
- Project.GetControl
- Project.ReleaseControl
- Project.Value
- Project.Run
- Project.GetIdentifier
- Project.GetSet
- Project.GetProcedure

Project.StartupMode

With this property of the *Project* object, you can modify the display mode of the AIMMS window. You can either set it before opening a project and thus set the initial mode of the AIMMS window, or you can modify it later on to minimize, maximize or restore the current AIMMS window. *Property*

Synopsis:

Int StartupMode

Remarks:

The property *StartupMode* is of type integer and can have any of the following values:

- *STARTUP_NORMAL* = 0 (default)
- *STARTUP_MINIMIZED* = 1
- *STARTUP_MAXIMIZED* = 2
- *STARTUP_HIDDEN* = 3

Project.User

This property is identical to the `--user` or `-U` command line option of AIMMS. *Property*
With this option you can specify the user name and optionally the password with which you want to log on to the system.

Synopsis:

String User

Remarks:

Setting the property User only has an effect before a call to ProjectOpen.

See also:

Chapter 16 of User's Guide, Calling AIMMS

Project.Data

This property is identical to the `--data` or `-D` command line option of AIMMS. *Property*
With this option you can specify the data manager file that you want to use within the project.

Synopsis:

String Data

Remarks:

Setting the property `Data` only has an effect before a call to `ProjectOpen`.

See also:

Chapter 16 of User's Guide, Calling AIMMS

Project.LicenseServer

This property is identical to the `--license-server` or `-L` command line option of AIMMS. With this option you can specify the host name of the server machine where the AIMMS License Manager is running. *Property*

Synopsis:

String LicenseServer

Remarks:

Setting the property `LicenseServer` only has an effect before a call to the method `ProjectOpen`.

See also:

Chapter 16 of User's Guide, Calling AIMMS

Project.ConfigDir

This property is identical to the `--config_dir` or `-C` command line option of AIMMS. With this option you can specify the directory where AIMMS should look for the various licensing files. By default AIMMS looks in the CONFIG directory below the top level AIMMS directory. *Property*

Synopsis:

String ConfigDir

Remarks:

Setting the property ConfigDir only has an effect before a call to the function ProjectOpen.

See also:

Chapter 16 of User's Guide, Calling AIMMS

Project.ProjectOpen

With this method you can open an existing AIMMS project. Depending on the `StartupMode` the AIMMS window is created and the startup sequence of the project is executed.

Synopsis:

```
Long ProjectOpen(  
    project_name,      ! (input) String  
    [as_server]       ! (optional) Boolean  
)
```

Arguments:

project_name

The full path name of the existing AIMMS project file that you want to open.

as_server (optional)

This option is especially created for accessing AIMMS from within the script of an ASP file (Active Server Page). The option instructs AIMMS to avoid any unwanted output (windows or dialog boxes) to the screen.

Return value:

If the project is successfully opened the method returns 1, otherwise 0.

Remarks:

Before opening an AIMMS project, you can set the Project properties `User`, `Data`, `StartupMode`, `LicenseServer`, and/or `ConfigDir` to make sure that AIMMS starts in the correct manner.

Example:

```
Dim TheProject As Aimms.Project  
set TheProject = new Aimms.Project  
TheProject.StartupMode = STARTUP_MINIMIZED  
TheProject.ProjectOpen "C:\Aimms\Projects\Transport.prj"
```

Project.ProjectClose

With this method you can close the AIMMS project.

Synopsis:

ProjectClose

Arguments:

None

Project.Name

This property holds the name of the project file as you have specified it in the call to `ProjectOpen`. *Property*

Synopsis:

String Name

Remarks:

The property `Name` is read only.

See also:

The method `Project.ProjectOpen`

Project.ArrayIndexOffset

Some of the objects in the AIMMS COM interface have methods that create and allocate memory for a new array of values. Similar as in Visual Basic, these so-called SafeArrays use index numbering which starts at 0 or 1. In other words: should the first element in an array A be referenced as A(0) or A(1). By default, the arrays created by the AIMMS COM interface start at 0, the property `ArrayIndexOffset` changes this default.

Property

Synopsis:

Long `ArrayIndexOffset`

Remarks:

By default this property is equal to 0, but you can change it to any other integer value. Usually, you either use 0 or 1. The property `ArrayIndexOffset` is similar to the "Option Base" statement in Visual Basic.

Project.DefaultTuplePassMode

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 this property you can specify your default mode for passing element tuples. When you create an `Identifier` or `Set` object, the value of the property `DefaultTuplePassMode` is used as the initial value of `Identifier.TuplePassMode` or `Set.CompoundTuplePassMode`, respectively.

Synopsis:

```
Int DefaultTuplePassMode
```

Remarks:

`DefaultTuplePassMode` is of type integer, and it can have one of the following defined values:

- `ELEMENT_BY_NUMBER` = 0 (default)
- `ELEMENT_BY_ORDINAL` = 1
- `ELEMENT_BY_NAME` = 2

See also:

The properties `Identifier.TuplePassMode`, `Set.CompoundTuplePassMode`

Project.DefaultElementValuePassMode

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 this property you can specify your default mode for passing element values. When you create an Identifier or Set object, the value of the property `DefaultElementValuePassMode` is used as the initial value of the property `Identifier.ElementValuePassMode` or `Set.ElementPassMode`, respectively.

Synopsis:

```
Int DefaultElementValuePassMode
```

Remarks:

`DefaultElementValuePassMode` is of type integer, and it can have one of the following defined values:

- `ELEMENT_BY_NUMBER` = 0 (default)
- `ELEMENT_BY_ORDINAL` = 1
- `ELEMENT_BY_NAME` = 2

See also:

The properties `Identifier.ElementValuePassMode`, `Set.ElementPassMode`

Project.AssignArray

This method is a shortcut for calling the method `AssignArray` of an `Identifier` object directly from the `Project` object.

Synopsis:

```
AssignArray(  
    identifier_name,    ! (input) String  
    value_array,       ! (input) Variant array (Double/String/Integer)  
    [transposed]       ! (optional) Boolean  
)
```

Arguments:

The arguments of this method are a combination of the arguments from `Project.GetIdentifier` and `Identifier.AssignArray`

Examples:

The following two pieces of code have an identical effect on the AIMMS project:

```
'Assigning an array using the Identifier object  
Dim id As Aimms.Identifier  
Set id = MyProject.GetIdentifier( "Transport" )  
id.AssignArray NewValues
```

```
'Assigning an array directly from the Project object  
MyProject.AssignArray "Transport", NewValues
```

See also:

The methods `Project.GetIdentifier` and `Identifier.AssignArray`

Project.RetrieveArray

This method is a shortcut for calling the method `RetrieveArray` of an `Identifier` object directly from the `Project` object.

Synopsis:

```
RetrieveArray(
    identifier_name, ! (input) String
    value_array,    ! (output) Variant array (Double/String/Integer)
    [sparse],      ! (optional) Boolean
    [transposed]   ! (optional) Boolean
)
```

Arguments:

The arguments of this method are a combination of the arguments from `Project.GetIdentifier` and `Identifier.RetrieveArray`

Examples:

The following two pieces of code have an identical effect on the AIMMS project:

```
'Retrieving an array using the Identifier object
Dim id As Aimms.Identifier
Dim CurrentValues(4,4) As Variant
Set id = MyProject.GetIdentifier( "Transport" )
id.RetrieveArray CurrentValues
```

```
'Retrieving an array directly from the Project object
Dim CurrentValues(4,4) As Variant
MyProject.RetrieveArray "Transport", CurrentValues
```

See also:

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

Project.CreateArray

This method is a shortcut for calling the method `CreateArray` of an `Identifier` object directly from the `Project` object.

Synopsis:

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

Arguments:

The arguments of this method are a combination of the arguments from `Project.GetIdentifier` and `Identifier.CreateArray`.

Return value:

The method `CreateArray` returns a newly allocated array of `Variant` values. Whether the indices in this array start at 0 or 1 depends on the value of the property `ArrayIndexOffset`.

Examples:

The following two pieces of code have an identical effect on the AIMMS project:

```
'Creating an array using the Identifier object
Dim id As Aimms.Identifier
Set id = MyProject.GetIdentifier( "Transport" )
CurrentValues = id.CreateArray

'Creating an array directly from the Project object
CurrentValues = MyProject.CreateArray( "Transport" )
```

See also:

The methods `Project.GetIdentifier`, `Identifier.CreateArray` and the property `Project.ArrayIndexOffset`.

Project.AssignElementArray

This method is a shortcut for calling the method `AssignElementArray` of an `Set` object directly from the `Project` object.

Synopsis:

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

Arguments:

The arguments are a combination of the arguments from `Project.GetSet` and `Set.AssignElementArray`

Examples:

The following two pieces of code have an identical effect on the AIMMS project:

```
'Assigning an array using the Set object  
Dim s As Aimms.Set  
Set s = MyProject.GetSet( "Cities" )  
s.AssignElementArray MyCityNames
```

```
'Assigning an array directly from the Project object  
MyProject.AssignElementArray "Cities", MyCityNames
```

See also:

The methods [Project.GetSet](#) and [Set.AssignElementArray](#)

Project.RetrieveElementArray

This method is a shortcut for calling the method `RetrieveElementArray` of an `Set` object directly from the `Project` object.

Synopsis:

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

Arguments:

The arguments are a combination of the arguments from `Project.GetSet` and `Set.RetrieveElementArray`

Examples:

The following two pieces of code have an identical effect on the AIMMS project:

```
'Retrieving an array using the Set object  
Dim s As Aimms.Set  
Set s = MyProject.GetSet( "Cities" )  
s.RetrieveElementArray CurrentCityNames
```

```
'Retrieving an array directly from the Project object  
MyProject.RetrieveElementArray "Cities", CurrentCityNames
```

See also:

The methods [Project.GetSet](#) and [Set.RetrieveElementArray](#)

Project.CreateElementArray

This method is a shortcut for calling the method `CreateElementArray` of an `Set` object directly from the `Project` object.

Synopsis:

```
VariantArray CreateElementArray(  
    set_name      ! (input) String  
)
```

Arguments:

The arguments are a combination of the arguments from `Project.GetSet` and `Set.CreateElementArray`

Return value:

The method `CreateElementArray` returns a newly allocated array containing elements.

Examples:

The following two pieces of code give the same result:

```
'Creating an array using the Set object  
Dim s As Aimms.Set  
Set s = MyProject.GetSet( "Cities" )  
CurrentCityNames = s.CreateElementArray  
  
'Creating an array directly from the Project object  
CurrentCityNames = MyProject.CreateElementArray( "Cities" )
```

See also:

The methods [Project.GetSet](#) and [Set.CreateElementArray](#)

Project.AssignTable

This method is a shortcut for calling the method `AssignTable` of an `Identifier` object directly from the `Project` object.

Synopsis:

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

Arguments:

The arguments of this method are a combination of the arguments from `Project.GetIdentifier` and `Identifier.AssignTable`

Examples:

The following two pieces of code have an identical effect on the AIMMS project:

```
'Assigning a table using the Identifier object  
Dim id As Aimms.Identifier  
Set id = MyProject.GetIdentifier( "Transport" )  
id.AssignTable Rows, Columns, CurrentValues
```

```
'Assigning a table directly from the Project object  
MyProject.AssignTable "Transport", Rows, Columns, CurrentValues
```

See also:

The methods `Project.GetIdentifier` and `Identifier.AssignTable`

Project.RetrieveTable

This method is a shortcut for calling the method `RetrieveTable` of an `Identifier` object directly from the `Project` object.

Synopsis:

```
RetrieveTable(
    identifier_name,    ! (input) String
    row_tuples,        ! (input) Variant array (String/Integer)
    column_tuples,     ! (input) Variant array (String/Integer)
    value_array,       ! (input) Variant array (Double/String/Integer)
    [sparse],          ! (optional) Boolean
    [row_mode],        ! (optional) RowColumnMode
    [column_node]     ! (optional) RowColumnMode
)
```

Arguments:

The arguments of this method are a combination of the arguments from `Project.GetIdentifier` and `Identifier.RetrieveTable`

Examples:

The following two pieces of code produce the same results:

```
'Retrieving a table using the Identifier object
Dim id As Aimms.Identifier
Set id = MyProject.GetIdentifier( "Transport" )
id.RetrieveTable Rows, Columns, CurrentValues
```

```
'Retrieving a table directly from the Project object
MyProject.RetrieveTable "Transport", Rows, Columns, CurrentValues
```

See also:

The methods `Project.GetIdentifier` and `Identifier.RetrieveTable`

Project.GetControl

Whenever an AIMMS project runs in a multi-threaded environment, synchronization of the execution and data passing requests becomes of the utmost importance. By default, the AIMMS COM interface will make sure that no two execution or data requests initiated from different threads are dealt with simultaneously. In fact, any request checks whether it can get exclusive control and if not, waits for an infinite time until another thread releases its control. This works fine except for two cases:

- You want to wait only for a specified amount of time, and if this time elapses, cancel the operation.
- You want to execute a sequence of requests, without any intervention from other threads.

In both cases you should try and get the control explicitly, execute the request(s) and then release the control.

Synopsis:

```
Long GetControl(  
    [timeout]      ! (optional) Long  
)
```

Arguments:

timeout (optional)

The number of milliseconds to wait if the control cannot be obtained immediately. If this timeout argument is omitted, then the method will wait for an infinite amount of time. A timeout value of 0 means that the method will only obtain the control if it does not have to wait for it.

Return value:

If the control is obtained successfully the method returns 1, otherwise it returns 0.

Remarks:

Any successful call to the method `GetControl` *must* be followed by a call to `ReleaseControl`. Failure to do so will result in a situation where other threads cannot get the control.

See also:

The method `Project.ReleaseControl`

Project.ReleaseControl

This method releases the control over AIMMS that was obtained via a call to `GetControl`. Any successful call to `GetControl` *must* be followed by a call to this method.

Synopsis:

Release

Arguments:

None

See also:

The method `Project.GetControl`

Project.Value

This property is a shortcut for referencing the `Value` property of a `Scalar` *Property* object directly from the `Project` object. Note that `Value` is implemented as a property and not as a method. In a Visual Basic environment, this makes assignment and retrieval of a scalar value much easier.

Synopsis:

```
Variant Value(  
    identifier_name      ! (input) String  
)
```

Arguments:

identifier_name

The name of the scalar identifier(-slice). This argument is directly related to the single argument of `Project.GetIdentifier`.

Return value:

The property is of type `Variant`, which contains either a double, string or integer depending on the type of the identifier.

Examples:

The following two pieces of code give an identical result:

```
'Assigning a value using the Identifier object  
Dim id As Aimms.Identifier  
Set id = MyProject.GetIdentifier( "FuelCost" )  
id.Value = 2.50
```

```
'Assigning a scalar value directly from the Project object  
MyProject.Value( "FuelCost" ) = 2.50
```

Project.Run

This method is a shortcut for calling the method `Run` of a `Procedure` object directly from the `Project` object.

Synopsis:

```
Long Run(  
    procedure_name,      ! (input) String  
    [var_args]          ! (optional) variable number of arguments  
)
```

Arguments:

The arguments of this method are a combination of the arguments from `Project.GetProcedure` and `Procedure.Run`

Examples:

The following two pieces of code give the same result:

```
'Running a procedure using the Procedure object  
Dim proc As Aimms.Procedure  
Set proc = MyProject.GetProcedure( "MainExecution" )  
proc.Run
```

```
'Running a procedure directly from the Project object  
MyProject.Run "MainExecution"
```

See also:

The methods [Project.GetProcedure](#) and [Procedure.Run](#)

Project.GetIdentifier

This method creates a new Identifier object for a given identifier in the model or slice thereof.

Synopsis:

```
Aimms.Identifier GetIdentifier(  
    id_name,      ! (input) String  
)
```

Arguments:

id_name

The name of an identifier (slice) in the model. For example: "Transport" or "transport('Amsterdam',j)". In the latter example you may omit the free index j and specify "Transport('Amsterdam',)".

Return value:

A newly created Identifier object.

Remarks:

In Visual Basic you must use the 'Set' statement to assign the object to an object variable. See the example below:

Example:

```
Dim MyIdentifier As Aimms.Identifier  
Set MyIdentifier = MyProject.GetIdentifier( "Transport" )
```

See also:

The Aimms.Identifier object.

Project.GetSet

This method creates a new Set object for a given set in the model.

Synopsis:

```
Aimms.Set GetSet(  
    set_name      ! (input) String  
)
```

Arguments:

set_name
The name of a set in the model.

Return value:

A newly created Set object.

Remarks:

In Visual Basic you must use the 'Set' statement to assign the object to an object variable. See the example below:

Example:

```
Dim MySet As Aimms.Set  
Set MySet = MyProject.GetSet( "Cities" )
```

See also:

The *Aimms.Set* object.

Project.GetProcedure

This method creates a new Procedure object for a given procedure in the model.

Synopsis:

```
Aimms.Procedure GetProcedure(  
    proc_name      ! (input) String  
)
```

Arguments:

proc_name

The name of a procedure in the model. If the procedure has arguments then you should not include these arguments in the name.

Return value:

A newly created Procedure object.

Remarks:

In Visual Basic you must use the 'Set' statement to assign the object to an object variable. See the example below:

Example:

```
Dim MyProc As Aimms.Procedure  
Set MyProc = MyProject.GetProcedure( "MainExecution" )
```

See also:

The Aimms.Procedure object.