
AIMMS Language Reference - Index Binding

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 © 1993–2018 by AIMMS B.V. All rights reserved.

AIMMS B.V.
Diakenhuisweg 29-35
2033 AP Haarlem
The Netherlands
Tel.: +31 23 5511512

AIMMS Inc.
11711 SE 8th Street
Suite 303
Bellevue, WA 98005
USA
Tel.: +1 425 458 4024

AIMMS Pte. Ltd.
55 Market Street #10-00
Singapore 048941
Tel.: +65 6521 2827

AIMMS
SOHO Fuxing Plaza No.388
Building D-71, Level 3
Madang Road, Huangpu District
Shanghai 200025
China
Tel.: ++86 21 5309 8733

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

AIMMS is a registered trademark of AIMMS B.V. IBM ILOG CPLEX and CPLEX is a registered trademark of IBM Corporation. GUROBI is a registered trademark of Gurobi Optimization, Inc. KNITRO is a registered trademark of Artelys. WINDOWS and EXCEL are registered trademarks of Microsoft Corporation. \TeX , \LaTeX , and $\AMS-\LaTeX$ 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 AIMMS 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 AIMMS B.V.

AIMMS 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 AIMMS 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, AIMMS 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, AIMMS 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 AIMMS B.V. using \TeX and the LUCIDA font family.

Chapter 9

Index Binding

This chapter presents the *index binding* rules implemented in AIMMS. These rules play an essential role during most repetitive set operations. For standard situations AIMMS behaves as expected. You should read this chapter if you are interested in a formal discussion of the rules of the underlying semantics.

This chapter

9.1 Binding rules

During execution, indices are used to traverse a set to repeatedly apply a specific operation on all elements of a set. These operations concern

Repetitive operations

- indexed assignment statements,
- FOR statements,
- iterative operations like summation over a domain,
- constraint generation,
- arc generation, and
- constructed set expression.

Index binding is the process by which AIMMS repeatedly couples the value of an index to elements of a specific set to execute repetitive operations.

Index binding

There are three ways in which index binding takes place:

Different types of binding

- *local* binding,
- *default* binding, and
- *context* binding.

Local binding takes place through the use of an IN modifier at the index binding position as illustrated in the following example.

Local binding

```
NettoTransport(i in SupplyCities, j in DestinationCitiesFromSupply(i)) :=  
    Transport(i,j) - Transport(j,i);
```

Instead of executing the assignment for all cities *i* and *j*, it is only executed for those combinations for which city *i* is in `SupplyCities` and city *j* is in `DestinationCitiesFromSupply(i)`.

Indices can have a *default binding*. This is the binding specified in a declaration. You can specify a default binding either via the `Index` attribute of a set, or via the `Range` attribute of an `Index` declaration. Whenever you use an index with a default binding and do not specify a local binding, AIMMS will couple this index to its default set automatically. The following example illustrates default binding.

Default binding

```
IntermediateTransportCitiesInBetween(i,j) :=
    DestinationCitiesFromSupply(i) * SupplyCitiesToDestination(j);
```

Assuming that `i` and `j` have a default binding to the set `Cities`, the assignment takes place for all tuples of cities `(i,j)`.

Whenever you use an index that has no default binding and for which you do not provide a local binding, AIMMS will try to determine a *context binding* from the context. Assume that `k` is an index without a default binding. Further assume that `LargestTransport` is an element parameter into `Cities` and indexed over `Cities`. Then the following example is an illustration of context binding.

Context binding

```
LargestTransport(k) := ArgMax( j, Transport(k,j) );
```

In this assignment AIMMS will automatically bind the index `k` to `Cities`, because the identifier `LargestTransport` has been declared with the index domain `Cities`. Note that context binding will only work in indexed assignments.

Index binding can be nested through the use of indexed element-valued parameters on the left-hand side of an assignment. The binding takes place in the way that you would expect, applying the same rules as for non-nested index binding. For example, given the declarations

Nested index binding

```
ElementParameter NextCity {
    IndexDomain : i;
    Range       : Cities;
}
ElementParameter PreviousCity {
    IndexDomain : i;
    Range       : Cities;
}
```

the following assignment, which computes the value of `PreviousCity` given the contents of `NextCity`, will bind the nested reference to the index `i`.

```
PreviousCity( NextCity(i) ) := i;
```

This binding is sparse, in the sense that the statement is only executed for those `i` for which `NextCity(i)` assumes a nonempty value.

In general, AIMMS will never accept the use of an index in references to indexed identifiers when the binding set does not have the same root set as the index domain of the identifier. This is even the case when the elements, referenced in the particular statement, have identical names in both the binding set and the index domain. Internally, AIMMS stores a set elements as a unique (integer) number with respect to its root set, and uses this number for storing data for that element in indexed identifiers. Thus, when the root sets of the binding set and the index domain are not identical, the set element numbers will be incompatible, preventing AIMMS from referencing the correct data.

*Compatible
index binding
only*

When you want to use a binding set which is incompatible with the index domain of identifier on the left-hand side of an assignment, you should manually create an element parameter which maps elements in one root to the corresponding elements the other root set. Such a mapping can be easily created using the function `ElementCast` (discussed in Section 5.2.2), as exemplified below.

*Use indirect
referencing*

```
ElementMap(i) := ElementCast( IncompatibleRootSet, i );
```

Subsequently, you can use a nested binding through the element parameter `ElementMap` to reference elements in the index domain of the identifier on the left-hand side of an assignment, while still using the index `i` as a binding index, as illustrated in the following statement.

```
IncompatibleParameter( ElementMap(i) ) := CompatibleParameter(i);
```

Conversely, when you want to use an incompatible set element in a parameter reference on the right-hand side of an assignment, there is no direct need to create a mapping parameter. In an expression on the right of an assignment, you can use the function `ElementCast` directly at any index position, as illustrated below.

*Use the
ElementCast
function*

```
CompatibleParameter(i) := IncompatibleParameter( ElementCast(IncompatibleRootSet, i) );
```

Note that you could have accomplished the same effect by creating a universal set of which all other sets are subsets. As a result, all set elements are represented as unique integer numbers with respect to the same root set, allowing the index domains of all identifiers to be referenced in a compatible manner. However, often it is not very natural to do so, and the usage of a universal set is likely to slow down the performance of AIMMS.

Universal set

For most situations the result of index binding is self-evident and the behavior of the system is as you would expect. Following are the precise rules for index binding.

*Index binding
rules*

- **Dominance rule:** Whenever index binding takes place, local binding precedes default binding, which in turn precedes context binding. If no method is applicable, a compile time error will result.
- **Intersection rule:** In indexed assignments the binding set(s) should be compatible with the index domain. The assignment will be performed for all tuples on the left-hand side that lie in the intersection of the binding set(s) and the index domain of the corresponding identifier.
- **Ordering rule:** Lag and lead operators, as well as the `Ord` and `Element` functions operate according to the order of elements in the corresponding binding set.