AIMMS Language Reference - Parameter Declaration

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.

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

Chapter 4

Parameter Declaration

Terminology

The word parameter does not have a uniform meaning in the scientific community. When you are a statistician, you are likely to view a parameter as an unknown quantity to be estimated from observed data. *In* AIMMS *the word parameter denotes a known quantity that holds either numeric or string-valued data.* In programming languages the term variable is used for this purpose. However, this is not the convention adopted in AIMMS, where, in the context of a mathematical program, the word variable is reserved for an unknown quantity. Outside this context, a variable behaves as if it were a parameter. The terminology in AIMMS is consistent with the standard operations research terminology that distinguishes between parameters and variables.

Why use parameters

Rather than putting the explicit data values directly into your expressions, it is a much better practice to group these values together in parameters and to write all your expressions using these symbolic parameters. Maintaining a model that contains explicit data is a painstaking task and error prone, because the meaning of each separate number is not clear. Maintaining a model in symbolic form, however, is much easier and frequently boils down to simply adjusting the data of a few clearly named parameters at a single point.

Example

Consider the set `Cities` introduced in the previous chapter and a parameter `FixedTransport(i,j)`. Suppose that the cost of each unit of transport between cities `i` and `j` is stored in the parameter `UnitTransportCost(i,j)`. Then the definition of `TotalTransportCost` can be expressed as

```
TotalTransportCost := sum[(i,j), UnitTransportCost(i,j)*FixedTransport(i,j)];
```

Not only is this expression easy to understand, it also makes your model extendible. For instance, an extra city can be added to your model by simply adding an extra element to the set `Cities` as well as updating the tables containing the data for the parameters `UnitTransportCost` and `FixedTransport`. After these changes the above statement will automatically compute `TotalTransportCost` based on the new settings without any explicit change to the symbolic model formulation.

4.1 Parameter declaration and attributes

Declaration and attributes

There are four parameter types in AIMMS that can hold data of the following four data types:

- **Parameter** for numeric values,
- **StringParameter** for strings,
- **ElementParameter** for set elements, and
- **UnitParameter** for unit expressions.

Prior to declaring a parameter in the model editor you need to decide on its data type. In the model tree parameters of each type have their own icon. The attributes of parameters are given in Table 4.1.

Attribute	Value-type	See also page
IndexDomain	*index-domain*	
Range	*range*	
Default	*constant-expression*	
Unit	*unit-expression*	
Property	NoSave, Stochastic, Uncertain, Random, *numeric-storage-property*	48
Text	*string*	19
Comment	*comment string*	19, 32
Definiton	*expression*	34
InitialData	*data enumeration*	419
Uncertainty	*expression*	48, 339
Region	*expression*	48, 336
Distribution	*expression*	49, 342

Table 4.1: Parameter attributes

Basic examples

The following declarations demonstrate some basic parameter declarations

```
Parameter Population {
     IndexDomain  :  i;
     Range        :  [0,inf);
     Unit         :  [ 1000 ];
     Text         :  Population of city i in thousands;
}
Parameter Distance {
     IndexDomain  :  (i,j);
     Range        :  [0,inf);
     Unit         :  [ km ];
     Text         :  Distance from city i to city j in km;
```

```
}
ElementParameter cityWithLargestPopulation {
     Range        :  cities;
     Definition   :  argMax( i, Population( i ) );
}
StringParameter emergencyMessage {
     InitialData :  "Warning";
}
Quantity Currencies {
     BaseUnit     :  dollar;
     Conversions  :  euro -> dollar : # -> # * 1.3;
}
UnitParameter selectedCurrency {
     InitialData :  [euro];
}
```

The IndexDomain attribute

For each multidimensional identifier you need to specify its dimensions by providing a list of index bindings at the IndexDomain attribute. Identifiers without an IndexDomain are said to be *scalar*. In the index domain you can specify default or local bindings to either simple or compound sets. The totality of dimensions of all bindings determine the total dimension of the identifier. Any references outside the index domain, either through execution statements or from within the graphical user interface are skipped.

Domain condition

You can also use the IndexDomain attribute to specify a logical expression which further restricts the valid tuples in the domain. During execution, assignments to tuples that do not satisfy the domain condition are ignored. Also, evaluation of references to such tuples in expressions will result in the value zero. Note that, if the domain condition contains references to other data in your model, the set of valid tuples in the domain may change during a single interactive session.

Example

Consider the sets ConnectedCities with default index cc and DestinationCitiesFromSupply(i) from the previous chapter. The following statements illustrate a number of possible declarations of the two-dimensional identifier UnitTransportCost with varying index domains.

```
Parameter UnitTransportCost {
    IndexDomain : (i,j);
}
Parameter UnitTransportCostWithCondition {
    IndexDomain : (i,j) in ConnectedCities;
}
Parameter UnitTransportCostWithCompoundDomain {
    IndexDomain : cc;
}
Parameter UnitTransportCostWithIndexedDomain {
    IndexDomain : (i, j in DestinationCitiesFromSupply(i));
}
```

Explanation

The identifiers defined in the previous example will behave as follows.

- The identifier UnitTransportCost is defined over the full Cartesian product Cities × Cities by means of the default bindings of the indices i and j. You will be able to assign values to every pair of cities (i,j), even though there is no connection between them.
- The identifier UnitTransportCostWithCondition is defined over the same Cartesian product of sets. Its domain, however, is restricted by an additional condition (i,j) in ConnectedCities which will exclude assignments to tuples that do not satisfy this condition, or evaluate to zero when referenced.
- The identifier UnitTransportCostWithCompoundDomain is defined over the two-dimensional compound set ConnectedCities by the default binding of the index cc. Although the declaration seems equivalent to that of UnitTransportCostWithCondition, AIMMS will now produce a domain error, when you try to make assignments to tuples outside of this set.
- Finally, the identifier UnitTransportCostWithIndexedDomain is defined over a subset of the Cartesian product Cities × Cities. The second element j must lie in the subset DestinationCities(i) associated with i. AIMMS will produce a domain error if this condition is not satisfied.

The .Domain suffix

For every indexed identifier you have access to the current contents of the IndexDomain attribute through the .Domain suffix associated with the identifier at hand. The .Domain suffix behaves like a set of the dimension of its associated identifier. You can use it, for instance, in the IndexDomain of another identifier to indicate that two identifiers should have exactly the same domain.

Example

The following declaration illustrates a parameter that has exactly the same domain as the parameter UnitTransportCostWithIndexedDomain declared above.

```
Parameter UnitTransportCostWithSharedDomain {
    IndexDomain : (i,j) in UnitTransportCostWithIndexedDomain.Domain;
}
```

The Range attribute

With the Range attribute you can restrict the values to certain intervals or sets. The Range attribute is not applicable to a StringParameter nor to a UnitParameter. The possible values for the Range attribute are:

- one of the predefined ranges Real, Nonnegative, Nonpositive, Integer, or Binary,
- any one of the interval expressions $[a, b]$, $[a, b)$, $(a, b]$, or (a, b), where a square bracket implies inclusion into the interval and a round bracket implies exclusion,
- any enumerated integer set expression, e.g. $\{a \,..\, b\}$ covering all integers from a until and including b,

- a set reference, if you want the values to be elements of that set. For set element-valued parameters this entry is mandatory.

The values for *a* and *b* can be a constant number, `inf`, `-inf`, or a parameter reference involving some or all of the indices on the index domain of the declared identifier.

Consider the following declarations. *Example*

```
Parameter UnitTransportCost {
    IndexDomain  :  (i,j);
    Range        :  [ UnitLoadingCost(i), 100 ];
}
Parameter  DefaultUnitsShipped {
    IndexDomain  :  (i,j);
    Range        :  {
        { MinShipment(i) .. MaxShipment(j) }
    }
}
Set States {
    Index        :  s;
}
Set adjacentStates {
    SubsetOf     :  States;
    IndexDomain  :  s;
}
ElementParameter nextState {
    IndexDomain  :  s;
    Range        :  adjacentStates(s);
}
```

It limits the values of the identifier `UnitTransportCost(i,j)` to an interval from `UnitLoadingCost(i)` to 100. Note that the lower bound of the interval has a smaller dimension than the identifier itself. The integer identifier `DefaultUnitsShipped(i,j)` is limited to an integer range through an enumerated integer range inside the set brackets.

The `Default` attribute

In AIMMS, parameters that have not been assigned an explicit value are given a default value automatically. You can specify the default value with the `Default` attribute. The value of this attribute *must* be a constant expression. If you do not provide a default value for the parameter, AIMMS will assume the following defaults:

- 0 for numbers,
- 1 for unit-valued parameters,
- the empty string "" for strings, and
- the empty element '' for set elements.

The Definition attribute

The Definition attribute of a parameter can contain a valid (indexed) numerical expression. Whenever a defined parameter is referenced inside your model, AIMMS will, by default, recompute the associated data if (data) changes to any of the identifiers referenced in its definition make its current data out-of-date. In the definition expression you can refer to any of the indices in the index domain as if the definition was the right-hand side of an assignment statement to the parameter at hand (see also Section 8.2).

Example

The following declaration illustrates an indexed Definition attribute.

```
Parameter MaxTransportFrom {
    IndexDomain  : i;
    Definition   : Max(j, Transport(i,j));
}
```

Care when used in loops

Whenever you provide a definition for an *indexed* parameter, you should carefully verify whether and how that parameter is used in the context of one of AIMMS' loop statements (see also Section 8.3). When, due to changes in only a slice of the dependent data of a definition during a previous iteration, AIMMS (in fact) only needs to evaluate a single slice of a defined parameter during the actual iteration, you should probably not be using a defined parameter. AIMMS' automatic evaluation scheme for defined identifiers will always recompute the data for such identifiers *for the whole domain of definition*, which can lead to severe inefficiencies for high-dimensional defined parameters. You can find a more detailed discussion on this issue in Section 13.2.3.

The Unit attribute

By associating a Unit to every numerical identifier in your model, you can let AIMMS help you check your model's consistency. AIMMS also uses the Unit attribute when presenting data and results in both the output files of a model and the graphical user interface. You can find more information on the use of units in Chapter 32.

The Property attribute

The Property attribute can hold various properties of the identifier at hand. The allowed properties for a parameter are NoSave or one of the numerical storage properties Integer, Integer32, Integer16, Integer8 or Double, in addition to the properties Stochastic, Uncertain, Random which are discussed in Section 4.1.1.

- The property NoSave indicates whether the identifier values are stored in cases. It is discussed in detail in Section 3.2.
- By default, the values of numeric parameters are stored as double precision floating point numbers. By specifying one of the storage properties Integer, Integer32, Integer16, Integer8, or Double AIMMS will store the values of the identifier as (signed) integers of default machine length, 4 bytes, 2 bytes or 1 byte, or as a double precision floating point number

respectively. These properties are only applicable to parameters with an integer range.

The Property statement

During execution you can change the properties of a parameter through the Property statement. The syntax of the Property statement and examples of its use can be found in Section 8.5.

The Text attribute

With the Text attribute you can provide one line of descriptive text for the end-user. If the Text string of an indexed parameter or variable contains a reference to one or more indices in the index domain, then the corresponding elements are substituted for these indices in any display of the identifier text.

4.1.1 Properties and attributes for uncertain data

Stochastic programming and robust optimization

The AIMMS modeling language allows you to specify both stochastic programs and robust optimization models. Both methodologies are designed to deal with models involving data uncertainty. In stochastic programming the uncertainty is expressed by specifying multiple scenarios, each of which can define scenario-specific values for certain parameters in your model. Stochastic programming is discussed in full detail in Chapter 19. For robust optimization, parameters can be declared to not have a single fixed value, but to take their values from an user-defined uncertainty set. Robust optimization is discussed in Chapter 20.

Properties

The following Parameter properties are available in support of stochastic programming and robust optimization models.

- The property Stochastic indicates that the identifier can hold stochastic event data for a stochastic model. It is discussed in detail in Section 19.2.
- The property Uncertain indicates that the identifier can hold uncertain values from an uncertainty set specified through the Uncertainty and/or Region attributes. Uncertain parameters are used in AIMMS' robust optimization facilities, and are discussed in detail in Section 20.2.
- The property Random indicates that the identifier can hold random values with respect to a distribution with characteristics specified through the Distribution attribute. Random parameters are used in AIMMS' robust optimization facilities, and are discussed in detail in Section 20.3.

The Uncertainty and Region attributes

The Uncertainty and Region attributes are available if the parameter at hand has been declared uncertain using the Uncertain property. Uncertain parameters are used by AIMMS' robust optimization framework, and are discussed in full detail in Section 20.2. With the Region attribute you can specify an uncertainty set using one of the predefined uncertainty sets Box, ConvexHull or Ellipsoid. The Uncertainty attribute specifies a relationship between the un-

certain parameter at hand, and one or more other (uncertain) parameters in your model. The `Uncertainty` and `Region` attributes are not exclusive, i.e., you are allowed to specify both, in which case AIMMS' generation process of the robust counterpart will make sure that both conditions are satisfied by the final solution.

The `Distribution` attribute

The `Distribution` attribute is available if the parameter at hand has been declared random using the `Random` property. Random parameters are used by AIMMS' robust optimization framework, and are discussed in full detail in Section 20.3. With the `Distribution` attribute you can declare that the values for the random parameter at hand adhere to one of the predefined distributions discussed in Section 20.3.