Our common AIMMS styling guide

Develop collaboratively and make our apps ready for support

Styling cheatsheet

- We agree on universal and consistent styling rules
- We write code in English
- We give self-explanatory names to the elements of the model:
  - Prefix in the name signals type of identifier
  - Name is meaningful and spelled in CamelCase
  - Procedures are verbs, identifiers are nouns
  - Set name is plural, index name is singular
  - Unit is given in attributes, not in the name
- We use whitespaces to make our code more readable:
  - After comma and colon
  - Before and after assignment, arithmetic, conditional operators and reserved words
  - No need to align comments and separate assignments on consecutive lines
- We use line-wrapping and indents to make our code more readable:
  - Avoid scrolling horizontally
  - Use (multiple) tabs to add visual structure to long statements
- We use indents to add structure to IF/FOR/WHILE statements
- We use always parentheses with ONLYIF operator
- We use parentheses when AND, OR and NOT are used in the same statement
- We comment when necessary, in concise manner

We agree on universal and consistent styling rules

- This is a collection of simple, but hard rules on how to style AIMMS code
- The styling conventions described below are meant to be followed in all AIMMS models, unless there is good reason to do otherwise (and your co-developers and the support team agree with that)
- *Exception*: models with a lot of legacy code or third-party code. In such cases add a new piece of code in the same style as the existing code.
- Each team of developers is welcome to add own styling rules to improve readability. But make sure that the consistency is kept throughout the model.
- We recommend to customize settings of your AIMMS editor (to be found via Settings → Editor Settings…):
  - LineNumberMargin: **True**, to show line numbers
  - TabWidth: **4**, still provides a visible indent, but you run off the page less fast
  - WordWrap: **True**
  - Colouring of specific identifier types, comments, etc. as to your personal preferences

We write code in English

- We use English for everything in the back-end of AIMMS: sections, identifier names, procedures, comments, etc
- For the front-end we choose language appropriate for the client
- Consider using localization tools of AIMMS in case the front-end is required in multiple languages

We give self-explanatory names to the elements of the model (1)

- We use prefixes in the names to signal types of identifiers:
  - First 1 or 2 letters of the model identifier followed by a _ (P_, EP_, PR_, V_, …).
  - Add B in front if the identifier is binary (BP_, BV_, …). Set range binary in attributes.
  - Add GUI_ if the identifier is specifically made for the UI (P_GUI_, EP_GUI_, …).
  - Add D_ if the identifier has a definition (P_D_, EP_D_, …). If both GUI_ and D_, start with GUI_ (P_GUI_D_, …).
  - Add _ in front for locally defined identifiers in procedures and functions (_P_DepotCosts).
- Prefix is followed by a meaningful name in CamelCase (P_TotalCost). We don't use underscores '_' in this part of the name.

Natural form	CORRECT	INCORRECT
Total cost	TotalCost	Totalcost, Total_Cost, Total_cost
SKU description	SkuDescription	SKUDescription, SKU_Description, SKUdescription
Non-empty bins	NonEmptyBins	NonemptyBins, Non-emptyBins

- If you doubt how to convert a phrase into CamelCase, follow the steps:
  - Remove apostrophes (') and accents (ü, é) if any
  - Replace hyphens ('-') with spaces
  - Lowercase everything
  - Uppercase only the first character of each word
  - Join all words in a single name

We give self-explanatory names to the elements of the model (2)

- Procedures are named as verbs (PR_DoSomething)
- Identifiers are named as nouns (P_Something)
- A set is the name of its contents in plural (S_Depots)
  - Subset names start with the superset (S_DepotsInGermany, not S_GermanDepots)
  - Indices are singular and use i/j/previous/next (i_depot, j_depot, not i_depot, i_depot2). For indices we also use camelCase, without capitalizing the first letter of the first word, use of shortened name is also fine, e.g. i_assortmentGroup, i_assortGr
- Avoid using numbers in names, if needed – spell them out (BP_ItemAtTwoLocations, not BP_ItemAt2Locations). *Exception*: conventional names, such as Maasvlakte2
- Do not use units in the name of identifier, specify units in the attributes. *Exception*: identifies that need to be shown in UI in different units
- We strongly suggest to specify units as it helps to check if calculations in the model are correct and to be clear about the definitions, e.g. if productivity is defined as units per hour or minutes per unit.

We use whitespaces to make our code more readable

- We DO REQUIRE single whitespaces usage:
  - After a comma or a colon
  - Before and after assignment and arithmetic operators (:=, +, -, /, *, ^, …), also when used as lag/lead and string operators
  - Before and after comparison and logic operators (=, <>, <, >, and, or, in, …)
  - Before and after conditional operators (|, $, onlyif), when $ is used as sparsity modifier use spaces around the whole expression (/$, :=$, …)
- We DO NOT REQUIRE space usage:
  - To vertically align comments or separate assignment statements on consecutive lines, it is a trouble to maintain them. *Exception*: indentation of IF/FOR/WHILE.
- We RECOMMEND space usage:
  - In case of nested statements with parentheses if it aids readability

CORRECT	INCORRECT
`sum(i, P_X(i))`	`sum(i,P_X(i))`
`1 + 1`	`1+1`
`count(i | P_X(i) < P_Y(i))`	`count(i|P_X(i)<P_Y(i))`

We use line-wrapping and indents to make our code more readable

- As a general rule: try to avoid the need for horizontal scrolling, except for long identifier names, URLs and pathnames
- When line-wrapping, each line after the first one is indented with at least one tab
- Do not use combination of tabs and spaces for vertical alignment, it is very difficult to maintain
- There is no one-size-fits-all rule for line-wrapping
- Common guideline: present your code in such a way that the next person can recognize at one glance which parts of the statements belong together. See next slide for some good examples.

We use line-wrapping and indents to make our code more readable (2)

- For functions start each argument on the next line with a tab:

```
SetElementAdd(
    S_LongSetName,
    EP_ElementParameterName,
    "New Element");
```

- For iterative operators with binding domain (such as SUM, MAX, MIN) start expression on the next line. In case expression is very long, use the same indent for all lines:

```
P_X := sum(i | P_LongName(i),
    P_VeryLongName(i)
    + P_AnotherVeryLongName(i) );
```

Placing '+' on the next line makes understanding of the calculation quicker

- In case of line-wrapping within conditional statement, align each line with the start of the statement on the first line:

```
P_X := count(i |
    P_FirstCondition(i)
    and P_SecondCondition(i)
    and P_ValueA(i) >= P_ValueB(i) );
```

We use indents to add structure to IF/FOR/WHILE statements

- IF/ELSEIF/FOR/WHILE is always followed by whitespace and opening parenthesis, even if the logical expression or binding domain is very simple
- We always start a new line after THEN/ELSE/DO
- This new line is indented by one tab, compared to the previous line
- The statements ELSE/ELSEIF/ENDIF/ENDFOR/ENDWHILE begin at the same width as the corresponding opening statements
- We don’t have strict rules about using of lower- or uppercase and extra white lines, but try to be consistent within one model

CORRECT

```
if (condition1) then
    action1;
elseif (condition2) then
    action2;
else
    for (index) do
        action3;
    endfor;
endif;
```

INCORRECT

```
if condition1 then
  action1;
elseif condition2 then
action2;
else for (index) do
action3;
endfor;
endif;
```

We always use parentheses with ONLYIF operator

- ONLYIF operator takes precedence over other operators, so parentheses are needed for correct evaluation of conditions

STATEMENT	EVALUATION
`5 onlyif (1 = 1)`	`5 - CORRECT`
`5 onlyif 1 = 1`	`0 - INCORRECT`
`5 onlyif (2 > 1)`	`5 - CORRECT`
`5 onlyif 2 > 1`	`1 - INCORRECT`

We use parentheses when AND, OR and NOT are used in the same statement

- This is simply to ensure that the statement is evaluated correctly
- Consider also the use of indents in order to align different sections of complex conditional statements

We comment when necessary, in concise manner

- We strive to write self-documented procedures, minimizing the need for writing and the hassle of updating comments
- In general a comment is placed above the calculation. In case width allows, end-of-line comments can be used
- A comment should explain the “Why”, code should tell “What”
- If an identifier can use an explanation, use the comment block at the bottom of identifier attributes. This comment is shown when hovering over the identifier.
- Use the following commenting keywords for easy referencing/bookmarking:
  - @TODO(<developer name>)
  - @FIXME(<developer name>)
  - @TEMP(<developer name>)
- If you need to refer to a model identifier in your comments, make sure to write it between single quotes ('P_Something'). This ensures that it gets updated when the name is changed.