Preparation of a new schema definition for the aeroPerformanceMap in CPACS 3.1

Author:	Marko Alder
Report version:	1.0
GitHub issue:	–
Date:	June 18, 2019

Contents

Nomenclature

Abbreviations

APM	AeroPerformanceMap
CPACS	Common Parametric Aircraft Configuration Schema
MULDICON	Multi-Disciplinary-Configuration
`NaN`	Not a Number

1 Current Situation

The aeroPerformanceMap (APM) in CPACS 3.0 is composed of a four-dimensional tensor (`machNumber`, `altitude`, `angleOfSideslip` and `angleOfAttack`). However, the complete parameter space is not required in practice.

Example 1.0.1

Given are the lift polars for the following combinations of Mach number and altitude:

$$(\texttt{machNumber}, \texttt{altitutde}) = (0.2, 0), (0.2, 1000), (0.4, 500) \quad \text{and} \quad (0.4, 1500)\ .$$

Each polar is given for a set of angle of attacks:

$$\texttt{angleOfAttack} = (-1., 0., 1.)$$

and an angle of sideslip:

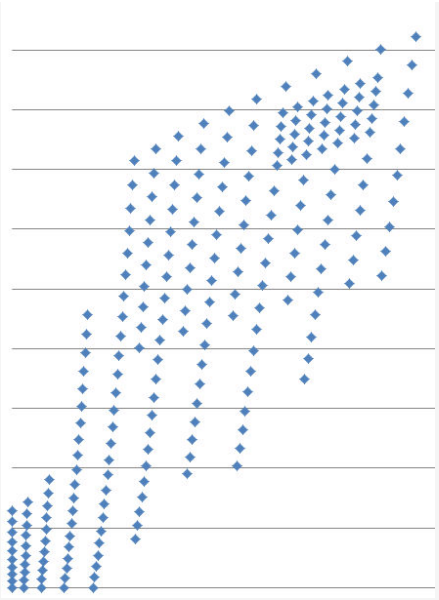
$$\texttt{angleOfSideslip} = 0\ .$$

Thus, a multidimensional matrix (tensor) with the dimensions (`machNumber` $\times$ `altitude` $\times$ `angleOfSideslip` $\times$ `angleOfAttack`), i.e. $2 \times 4 \times 1 \times 3$, must be spanned, which expects 24 values, although only 12 meaningful solutions exist:

		altitude			
		0	500	1000	1500
mach	0.2	•		•	
number	0.4		•		•

Example 1.0.2

Within the DLR project *AGILE* (`www.agile-project.eu`) a Mach-altitude-distribution as shown in the picture below was required. This highlights the need for a flexible scattering of data points.



This topic is addressed on GitHub issue #505: `https://github.com/DLR-SL/CPACS/issues/505`.

The question is how to **deal with missing** data and whether the **data can be stored more efficiently** in CPACS. In addition, there is a discussion as to whether and in what form **interpolation routines** should be available to the user for evaluating the data.

In previous discussions, four proposals for data storage in the APM have been developed, but no decision has yet been made as to which approach should be implemented. The present report is intended to shed more light on the proposals as well as to document the respective advantages and disadvantages in order to be able to determine the future procedure for dealing with the APMs.

2 Reference Case

2.1 Description of the complete reference case

As a reference a relatively large APM of the MULDICON configuration has been chosen (see Fig. 2.1). Since this data set is based on CPACS 2 the aerodynamic data is stored in the aircraft coordinate system instead of the aerodynamic coordinate system as used in CPACS 3. That's why in the following examples we have `cfx`, `cfy`, `cfz`, `cmx`, ... However, this will not influence the following schema proposals because the number of values remains the same. To publish the results on Github the data is furthermore multiplied with random factors.

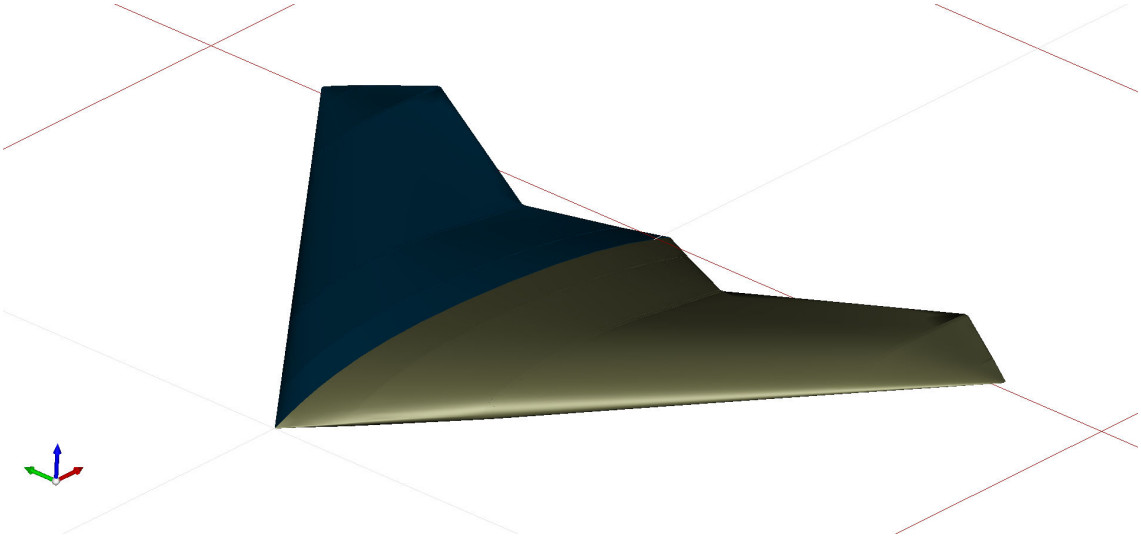


Figure 2.1: TiGL visualization of the MULDICON geometry.

The reference APM is composed of 5 Mach numbers (`machNumber`), 5 Reynolds numbers (`reynoldsNumber`), 5 angle of yaws (`angleOfYaw`) and 13 angle of attacks (`angleOfAttack`). This results in 1625 values for `cfx`, `cfy`, `cfz`, `cmx`, `cmy` and `cmz`. Additionally there are 18 damping derivatives (`dcfxdpstar` ... `dcmzdrstar`) and 6 sub-AMPs for control surface derivatives. These are varying from 2 to 5 deflection angles, resulting in 3250 to 8125 delta values (`dcfx` ... `dcmz`).

The values are arranged in CPACS matrix notation ("full factorial"). This results in **96 lines** requiring **2,23 MB** of disk space.

2.2 Reduced test case

The following test case is being set up to analyze the alternative schema proposals with respect to the issue described in Chapter 1. Figure 2.2 illustrates the assignment of data in the 4-dimensional tensor. Purple dots indicate that the corresponding parameter combination contains solutions. Yellow dots represent parameter combinations where no finite values exist.

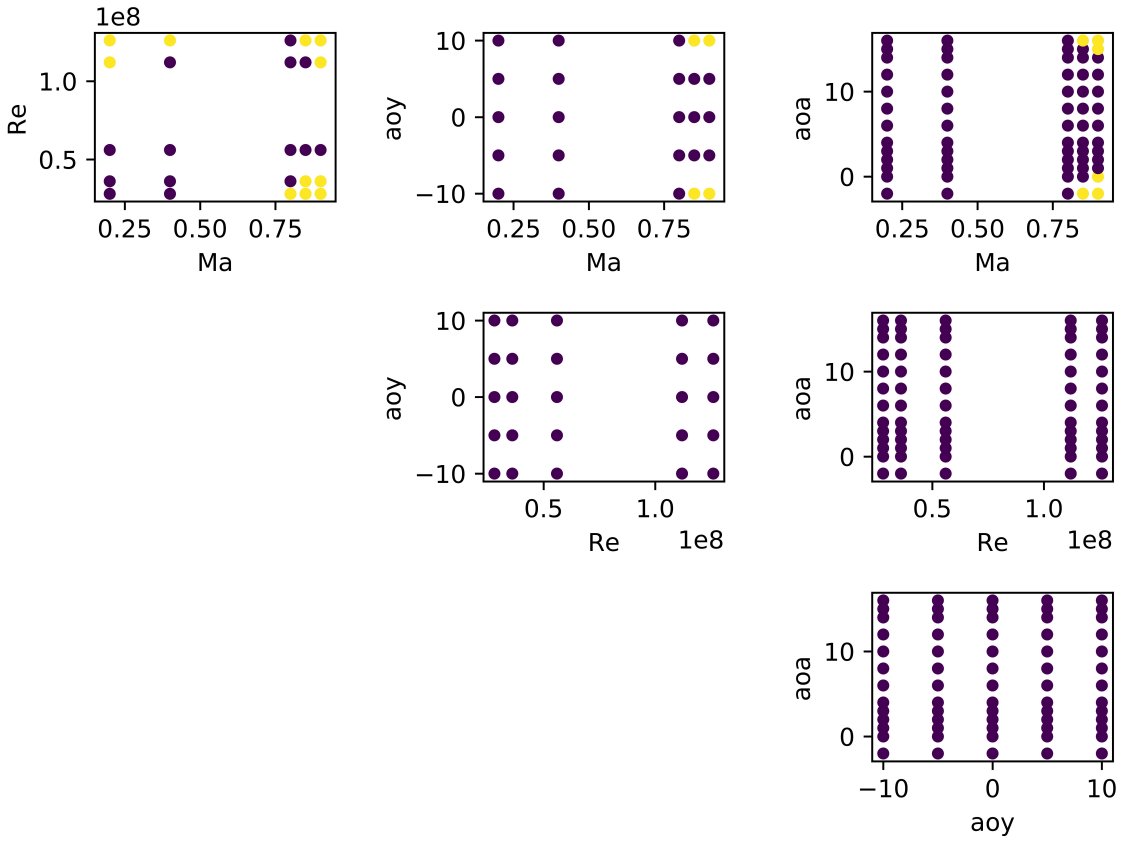


Figure 2.2: 4-dimensional assignment of the aeroPerformanceMap with empty/missing data shown in yellow.

2.3 Missing data

Even though it is intended to avoid non-finite values it must be defined how indicate these if necessary. One possibility is to use `NaN` (`"Not a Number"`) because of the following reasons:

- `NaN` is a valid value for the xml-type `xsd:double`
- The CPACS-API TiXI can read and write NaN values
- Python can handle `NaN` (e.g., `numpy.nan`)

- The string "NaN" is very short

Note 2.1*:* In this context it must be highlighted that the `vector` and `array` definition as used in CPACS is not a valid `xml` schema pattern. Here a vector is defined as an ordered sequence of repeated items of the same data type (e.g, see `https://www.w3.org/2005/07/xml-schema-patterns.html`). Because of the fact that this approach would drastically increase the number of lines in the CPACS file and decrease the readability it has been agreed to use a string with values separated by semicolons (`stringVectorBaseType` and `stringArrayBaseType`).

The drawback of relying only on `NaN` is that a program can not share the information if it failes either due to bad convergence or because of the underlying model assumptions. Thus, the workflow maintainer or tool owners may hardly distinguish if they need to adjust some input parameters to achieve convergence or if different models/tools must be included to provide the data. It has therefore been discussed weather to use on of the following strings to indicate that the required value is out of the range of the model/tool:

1. **`Na`** ("`Not available`"): a self-creation which is not known to any programming language. It must therefore be translated into another constant for API users.
2. **`Inf`** (`infinite`): is a valid value for `xsd:double` as well as a valid constant in C++ and Python (`np.inf`). However, the semantic meaning of *infinity* is different to what shall be expressed with providing no value in CPACS.
3. An **empty character**. Using no character (z.B. "`1; 2; ; ; 4; 5`") has the advantage of enabling short strings in CPACS files while maintaining the semantic meaning of "no value given". However, this is not possible for API users where some value or constant must be given in the resulting array.
4. **`None`**: is a semantically correct expression for providing no value. In CPACS files this choice would result into longer strings. In Python mixed arrays are possible. But using `numpy.ndarray` the type would be `object` and thus would permit, e.g., to perform array multiplications without further processing of the data types. Maybe definition of Null constants in C could correspond to `None`, but the author is no C-expert...

Note 2.2*:* Xml schemes provide `xsd:nil` elements to account for special characteristics of relational databases. By setting an element attribute `nillable="true"` the element is allowed to be empty. However, it appears a little far-fetched to set our CPACS-vectors/arrays `nillable="true"` because referring to single values in the array does not reflect the meaning of providing completely empty elements.

We should not further deviate from official xml-schema by inventing constants such as `Na`, i.e. option (1) is discarded. Option (4) provides a good semantic expression, but has its disadvantages both in CPACS (long character) as well for programming interfaces (no valid `double` type) and will therefore be discarded as well. Option (2) does not express the semantic meaing of not providing a value and should therefore not be used in the common language definition CPACS. Thus, it only remains option

(3) to distinguish between ”not providing a value“ and ”not possible to compute a value“ (taking the drawback into account that it must be translated into `Inf` when using the API).

Based on these discussions it will be decided that `NaN` will be used for missing values. Only `NaN` is both semantically correct (we have ”not a number“ if no value is given), a valid `xsd:double` type and a valid constant for most programming languages. If it will become necessary to distinguish between ”not providing a value“ and ”not possible to compute a value“ then option (3) can easily be introduced additionally.

3 Loop 1: Evaluation of alternative schema proposals

3.1 Prop. 1: Hierarchical structure

The idea of this proposal is to increase the readability and accessibility of the APM data by following a typical top-down description as used for other CPACS types. This could, for example, orientate on the following schema:

Transforming the reference data into the current schema results in an APM of **35092 lines** requiring **4,01 MB** disk space. The pros and cons can be summarized as follows:

Table 3.1: Pros and cons of the schema proposal 1.

pro	con
Readability: easy to find and interpret polars, since equal length vectors for coefficients and angle of attacks.	**Very long** ASCII file (performance issues)
Parsing: easy for parsers to query information using xpaths from the schema (no TiXI modifications necessary).	**Parsing**: since the data is not stored as needed for interpolation, performance issues might arise from quering data.
CLmax and **CLmin** values can easily be defined explicitly	**File size** increases significantly; may result into performance issues when parsing and processing
Flexible size: only nodes for given data necessary; very few `NaN`	
Polar characteristics of the data is enforced	

Figure 3.1: MindMap of the APM schema.

3.2 Prop. 2: Vectors of equal length for main APM

This proposal defines four input vectors (`machNumber`, `reynoldsNumber`, `angleOfYaw`, `angleOfAttack`) and the corresponding solution vectors with equal length in the main APM. Thus, the related data can easily be accessed via a given index value:

```
<aeroPerformanceCards>
  <machNumber mapType="vector">0.2;0.2;0.2;0.2;0.2;0.2;0.2;0.2;0.2;0.2;0.2;0.2;0.2;0
  <reynoldsNumber mapType="vector">2.8e+07;2.8e+07;2.8e+07;2.8e+07;2.8e+07;2.8e+07;2
  <angleOfYaw mapType="vector">-10;-10;-10;-10;-10;-10;-10;-10;-10;-10;-10;-10;-10;-
  <angleOfAttack mapType="vector">-2;0;1;2;3;4;6;8;10;12;14;15;16;-2;0;1;2;3;4;6;8;1
  <cfx mapType="vector">0.00802427;0.00632056;0.00627971;0.0067798;0.00782033;0.0094
  <cfy mapType="vector">0.000223446;0.000477298;0.000450143;0.00031789;7.91333e-05;-
  <cfz mapType="vector">-0.126437;-0.0267957;0.023365;0.073693;0.124143;0.174669;0.2
  <cmx mapType="vector">0.0143506;0.00523489;0.000655628;-0.00393092;-0.0085175;-0.0
  <cmy mapType="vector">-0.000845044;0.00255133;0.00416545;0.00571647;0.00721333;0.0
  <cmz mapType="vector">0.000853611;0.000313143;0.000312899;0.000537445;0.000904808;
  <dampingDerivatives>
    <positiveRates>
      <dcfxdpstar mapType="vector">0.013913;0.00438083;-0.000387;-0.00515433;-0.0099
      <dcfxdqstar mapType="vector">-0.0418403;-0.0109862;0.00443197;0.019838;0.03522
      <dcfxdrstar mapType="vector">-0.000504;-0.000523133;-0.000532767;-0.000542333;
```

Figure 3.2: Example for proposal 2. The values with index 3 in the vectors belong together.

This is consistent to the enginePerformanceMap and a common approach for file formats used for large data, e.g. NetCDF.

The proposal simplifies the data handling. Once the vectors are imported with TiXI, Python users only need to generate a single `id`-vector (as `numpy.ndarray`) to access the desired main APM data (see Listing 3.1).

```
1  import numpy as np
2  ...
3  ids = np.where((mach==0.2)&
4          (reynoldsNumber==2.8e7)&
5          (angleOfYaw==-10.)&
6          (angleOfAttack==1.))
7  cfx_values = cfx[ids]
```

Listing 3.1: Python example

Even though this compact approach is very specific to Python, it is expected that the degree of accessibility also applies for other programming languages.

The input and solution vectors can easily be extended by new values since no order of the data is enforced. Although this can be seen as an advantage, it contradicts the approach of storing data in the form of polars. Furthermore, it does not reflect our typical approach of working with the aeroPerformanceMap, i.e. that we first interpolate within `machNumber` and `reynoldsNumber` in a first step and `angleOfAttack` and `angleOfYaw` in a second step.

A drawback of this approach might also be the readability of the given boundary condition vectors (redundant values for `machNumber`, `reynoldsNumber`, `angleOfYaw` and `angleOfAttack`). Without scripting (e.g., by using `np.unique` in Python) it is difficult to get an overview of which data is stored in the APM compared to the current (full factorial) approach. Although the repeating values require more disk space, the current example only increases by $\approx 1\,\%$ (see Fig. B.1). Thus, the data overhead is small. The number of lines and the maximum line length used for the APM remain constant compared to the current definition (see Fig. B.2 and B.3).

The sub-APMs are defined as two-dimensional arrays (2D full factorial). They depend on the main-AMP vectors and the vector containing the relative control surface deflections.

A major advantage of this proposal is that there is no need for placeholders like `NaN` if values are not given (reduced APMs). In the current example the required disk space reduces by $\approx 50\,\%$. This goes along with reduced line lengths ($\approx 50\,\%$). Since only the length of the vectors is reduced, the number of lines remains constant (see Fig. B.1 to B.3).

Table 3.2: Pros and cons of the schema proposal 2.

pro	con
No `NaN` in main APM	**No Polar characteristics** (user could define values in arbitrary order)
Significant reduction in **file size** and required **disk space** for reduced APM	**Repeating values** for boundary conditions → small **data overhead** and poor **readability** with text editor
Very simple **data handling** with index vectors	Definition of **main- and sub-APM differs**
Easy to **add new data**	Does **not reflect the common approach of how to interpolate** the data
Consistent with **enginePerformanceMap**	

3.3 Prop. 3: Vectors of equal length for main and sub APM

This approach is comparable to Section 3.2. But instead of having sub-APMs with delta values, this proposal only consideres absolute values which are stored in one-dimensional vectors of equal lengths. This further simplifies the accessibility of the data because the matrix operations in Section 3.2 reduce to a single `id`-vector when relative deflections are considered (see Fig. 3.3).

```
<aeroPerformanceCards>
  <controlSurfaces>
    <controlSurface>
      <controlSurfaceUID>InnerWingFlap</controlSurfaceUID>
      <relDeflection mapType="vector">-1;-0.5;0;0.5;1;-1;-0.5;0;0.5;1
    </controlSurface>
    <controlSurface>
      <controlSurfaceUID>InnerWingFlap_symm</controlSurfaceUID>
      <relDeflection mapType="vector">0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
    </controlSurface>
    <controlSurface>
      <controlSurfaceUID>OuterWingFlap</controlSurfaceUID>
      <relDeflection mapType="vector">0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
    </controlSurface>
    <controlSurface>
      <controlSurfaceUID>OuterWingFlap_symm</controlSurfaceUID>
      <relDeflection mapType="vector">0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
    </controlSurface>
    <controlSurface>
      <controlSurfaceUID>SplitFlap</controlSurfaceUID>
      <relDeflection mapType="vector">0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
    </controlSurface>
    <controlSurface>
      <controlSurfaceUID>SplitFlap_symm</controlSurfaceUID>
      <relDeflection mapType="vector">0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
    </controlSurface>
  </controlSurfaces>
  <machNumber mapType="vector">0.2;0.2;0.2;0.2;0.2;0.2;0.2;0.2;0.
  <reynoldsNumber mapType="vector">2.8e+07;2.8e+07;2.8e+07;2.8e+07;2.
  <angleOfYaw mapType="vector">-10;-10;-10;-10;-10;-10;-10;-10;-1
  <angleOfAttack mapType="vector">-2;-2;-2;-2;0;0;0;0;1;1;1;1;1;
  <cfx mapType="vector">0.0148501;0.0100987;0.00802427;0.00818708;0.0
  <cfy mapType="vector">-0.000885604;0.000441696;0.000223446;-0.0013
  <cfz mapType="vector">-0.205919;-0.164617;-0.126437;-0.0887757;-0.0
  <cmx mapType="vector">-0.0113911;0.0019652;0.0143506;0.0266343;0.03
  <cmy mapType="vector">0.0256078;0.0117534;-0.000845044;-0.0132134;-
  <cmz mapType="vector">-0.0038659;-0.000345569;0.000853611;9.0357e-0
```

Figure 3.3: Example for proposal 3. The values with index 2 in the vectors belong together.

One disadvantage of this approach is the fact that for each solution an input value (boundary condition) must be written in the corresponding vectos (`relDeflection`, `machNumber`, `reynoldsNumber`, `angleOfYaw` and `angleOfAttack`). In the current

example the maximum line length increases from 84 650 to 452 685 (+435 %, see Fig. A.3 and B.3) which causes a noticeable drop in performance of usual text editors (e.g. Notepad++) on a standard laptop. Although the number of lines reduces by 38 %, the required disk space increases by 359 % (see Fig. B.2 to B.1).

Considering the reduced APM, the maximum line length only increases by 167 % because the `NaN` values can be neglected. The number of lines is not affected compared to the full APM (again −38 % with respect to the current full factorial approach) and the required disk space increases by 129 % (see Fig. B.1 to B.3).

Table 3.3: Pros and cons of the schema proposal 3.

pro	con
No `NaN` in main APM	Large increase in **file size**
Very simple **data handling** with index vectors	Very long **line lengths**
Easy to **add new data**	**Repeating values** for boundary conditions → large **data overhead** and poor **readability** with text editor
	No Polar characteristics (user could define value in arbitrary order)

3.4 Prop. 4: 2D full factorial by joining two variables to sets

This proposal intents to reduce the dimensionality of the APM by combining the `machNumber` and `reynoldsNumber` to a first set of input values as well as the `angleOfYaw` and `angleOfAttack` to a second set of values. These two sets form a two-dimensional APM matrix. The main purpose is to be more flexible in the choice of the values for `machNumber` and `reynoldsNumber` and therewith reduce the need to provide data for unrealistic flight conditions.

The associated reduction of redundant boundary values compared to the preceding proposals increases the readability when using a text editor. Furthermore, the matrix approach slightly enforces the polar characteristics of the data since for each `machNumber-reynoldsNumber`-set the same sequence of `angleOfAttack` values applies. It also reflects the common approach to first interpolate in the `machNumber-reynoldsNumber` space and the in the `angleOfAttack-angleOfYaw`space. So it forces the user a little bit into the direction we intend to use the aeroPerformanceMap and thus supports the semantic meaning behind the definition of a common language.

If the full APM is considered this approach does not show it's advantages. The file size, number of lines and maximum line lengths almost remains unchanged compared to the current full-factorial approach. However, the reduced APM results into a reduction of the required disk space by 47 % and a reduction of the maximum line length by 48 % (see Fig. B.1 to B.3).

A drawback of the proposal is that the definition of defining the arrays in CPACS becomes a little more complicated since the dimensions depend on sets of variables. This requires a thorough documentation.

```
<aeroPerformanceCard>
  <machNumber mapType="vector">0.2 0.2;0.2;0.4;0.4;0.4;0.4;0.8;0.8;0.8;0.8;0.85;0.85;0.9</machNumber>
  <reynoldsNumber mapType="vector">2.8e+07 3.6e+07;5.6e+07;2.8e+07;3.6e+07;5.6e+07;1.12e+08;3.6e+07;5.
  <angleOfYaw mapType="vector">-10;-10 -10 -10;-10;-10;-10;-10;-10;-10;-10;-10;-5;-5;-5;-5;-5;-5;-
  <angleOfAttack mapType="vector">-2;0 1 2;3;4;6;8;10;12;14;15;16;-2;0;1;2;3;4;6;8;10;12;14;15;16;-2;(
  <cfx mapType="array">0.00802427;0.00632056 0.00627971 0.0067798;0.00782033;0.00940019;0.0141707;0.0
  <cfy mapType="array">0.000223446;0.000477298 0.000450143 0.00031789;7.91333e-05;-0.000267222;-0.001
  <cfz mapType="array">-0.126437;-0.0267957 0.023365 0.073693;0.124143;0.174669;0.275766;0.376611;0.4
  <cmx mapType="array">0.0143506;0.00523489 0.000655628 -0.00393092;-0.0085175;-0.013101;-0.0222333;-(
  <cmy mapType="array">-0.000845044;0.00255133 0.00416545 0.00571647;0.00721333;0.00865342;0.0113685;(
  <cmz mapType="array">0.000853611;0.000313143 0.000312899 0.000537445;0.000904808;0.00144697;0.00306(
  <dampingDerivatives>
    <positiveRates>
      <dcfxdpstar mapType="array">0.013913;0.00438083 -0.000387 -0.00515433;-0.00991967;-0.014682;-0.(
      <dcfxdqstar mapType="array">-0.0418403;-0.0109862 0.00443197 0.019838;0.0352267;0.050593;0.0812(
      <dcfxdrstar mapType="array">-0.000504;-0.000523133 -0.000532767 -0.000542333;-0.000552;-0.000562
      <dcfydpstar mapType="array">0.102864;0.0254028 -0.0133442 -0.0520877;-0.0908166;-0.12952;-0.206(
      <dcfydqstar mapType="array">0.00773063;0.00257503 -4.33333e-07 -0.0025703;-0.00513156;-0.007681(
      <dcfydrstar mapType="array">-0.0024385;-0.0036103 -0.00419467 -0.00477777;-0.00535942;-0.005939(
```

Figure 3.4: Example for proposal 4. The first `machNumber-reynoldsNumber`-set (red) and the second `angleOfYaw-angleOfAttack`-set (green) form a two-dimensional array with the corresponding values shown in blue.

Table 3.4: Pros and cons of the schema proposal 4.

pro	con
Reduction of `NaN` values due to increased flexibility in the choice of flight conditions	New convention for prescribing and reading arrays in CPACS
Reflects the **intended interpolation procedure**	still **full-factorial** in two-dimensions
Improved readability of the CPACS file compared to preceding proposals.	
Significant reduction in **file size** and required **disk space** for reduced APM	

3.5 Prop 5.1/5.2: Mach and altitude hierarchical, AoA and AoY as vector

Trying to combine the hierarchical approach (see Sec. 3.1) with a reasonable size and length of the resulting CPACS file some hybrid solutions have been discussed. The general idea is to treat `machNumber` and `reynoldsNumber` hierarchically and using vectors (referred to as proposal 5.1; see Sec. 3.2) or arrays (referred to as proposal 5.2; see Sec. 3.4) for `angleOfAttack` and `angleOfYaw`. Without going further into detail the following figures illustrate the approaches: Fig. 3.5 and Fig. 3.6 treat `angleOfAttack` and `angleOfYaw` as vectors. The difference between these two is that Fig. 3.5 threats the sub-aeroPerformanceMap as vectors while defining the sub-aeroPerformanceMap as arrays in Fig. 3.6 in order to define a much more compact schema (i.e., not a separate node for each control surface).

3.6 Prop 6: Mach and altitude hierarchical, AoA and AoY as array

Proposal 5 can also be modified in such way that the `angleOfAttack` and `angleOfYaw` parameter space is defined by an array (see Fig. 3.7). This comes along with the advantages and disadvantages of using the full-factorial approach.

Figure 3.5: MindMap of the APM schema 5.1

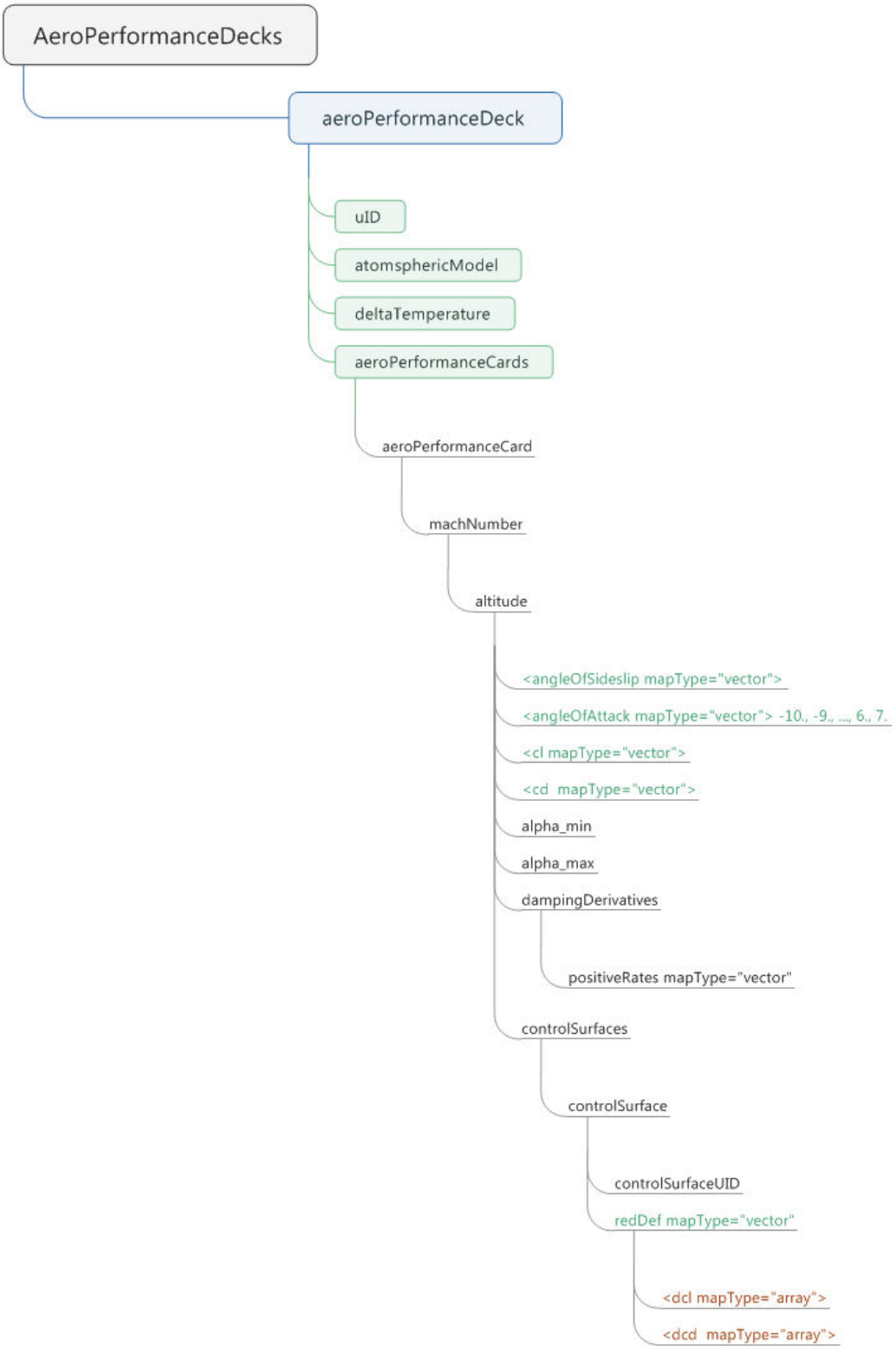


Figure 3.6: MindMap of the APM schema 5.2.

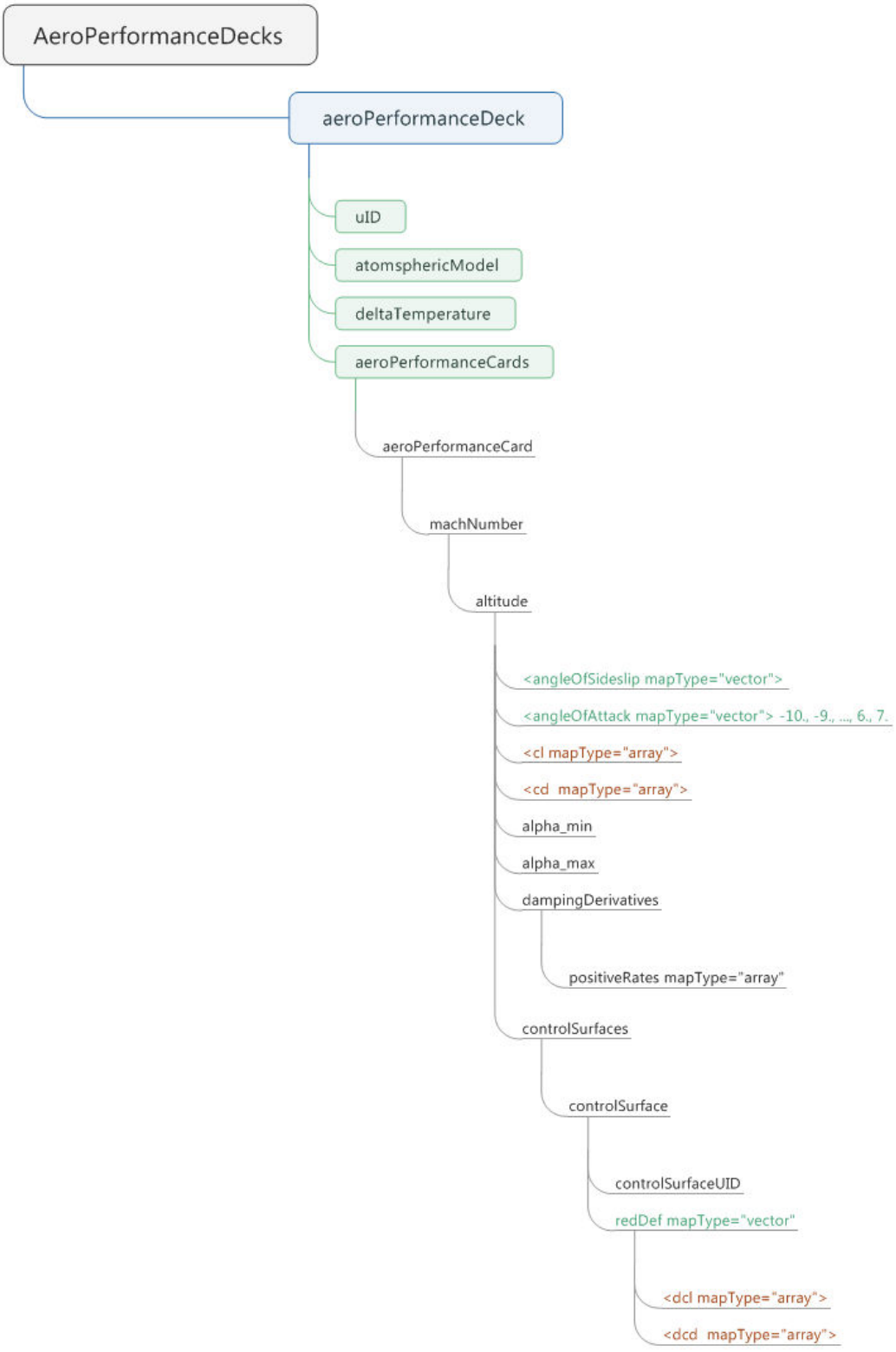


Figure 3.7: MindMap of the APM schema 6.

4 Loop 2: Decision and detailed elaboration of the schema

4.1 Comparison of the loop 1 results

Based on the results presented in Chapter 3 the proposal 1 can not be considered a potential alternative to the current 4D full factorial approach because the disproportional increase of lines. Even if the combined approaches (`machNumber` and `reynoldsNumber` hierarchically and `angleOfAttack` and `angleOfYaw` as vectors/arrays) result into xml-files which are dominated by the aeroPerformanceMap. Furthermore, proposal 3 can be eliminated due to the fact that the number of values written in the vector elements reaches a point where different file formats for big data (e.g., `hdf5`) should be applied.

Therefore, the choice reduces to proposal 2 (vectors of equal length for main APM) and proposal 4 (2D full factorial). Both approaches show comparable results in terms of the required disk space, the number of lines and the maximum line length (see Fig B.1 to B.3).

Table 4.1 compares the pros of the two schema proposals. Both approaches significantly reduce the need to write `NaN` values into the APM. The reduction in file size and required disk space in the case of reduced APMs is comparable. Opposed to the improved readability of proposal 4 is the simplified data handling (reading, processing and writing new data) of proposal 2. The latter could be accomplished by an appropriate API functionality, though. To be more consistent with the eningePerformanceMap in CPACS 3 is a valuable advantage of proposal 2.

Table 4.2 compares the cons of the two schema proposals. The two-dimensional representation of the APM in proposal 4 may lead to infinite numbers if the coefficients can not be computed for the complete range of `angleOfAttack` and `angleOfYaw` (e.g., due to stall). However, by using `NaN` strings this should be acceptable. The repeating values for the boundary condition vectors in proposal 2 decrease the readability of the ASCII file. However, for large matrices this holds as well for proposal 4 and by counting the values it is also possible to get an overview of the APM content at proposal 2.

Because of the arbitrary position of solution values in the APM, it is much easier for the user to destroy the polar characteristics. This makes it, in a first glance, harder to determine the derivatives for flight mechanics. However, proposal 4 does not force the polar characteristics as well, since the `angleOfAttack`-vector might be

Table 4.1: Pros of schema proposals 2 and 4.

proposal 2	proposal 4
No `NaN` in main APM	**Reduction of `NaN`** values due to **increased flexibility in the choice of flight conditions**
Significant reduction in **file size** and required **disk space** for reduced APM	Significant reduction in **file size** and required **disk space** for reduced APM
Very simple **data handling** with index vectors	**Improved readability** of the CPACS file compared to preceding proposals.
Easy to **add new data**	Reflects the **intended interpolation procedure**
Consistent with **enginePerformanceMap**	

given in an arbitrary order as well. Sorting and evaluating the content of the APM has therefore a high priority in the development of an appropriate CPACS library with API functionality.

Table 4.2: Cons of schema proposals 2 and 4.

proposal 2	proposal 4
No polar characteristics (user could define values in arbitrary order)	New convention for prescribing and reading matrices in CPACS
Repeating values for boundary conditions → small **data overhead** and poor **readability** with text editor	still full-factorial in two-dimensions
Definition of **main- and sub-APM differs**	
Does **not reflect the common approach of how to interpolate** the data	

4.2 Decision

With respect to the improved consistency of the performanceMaps in CPACS and the fact that most of the drawbacks of proposal 2 can be handled by an appropriate CPACS library, we decide to continue with proposal 2 (see Sec. 3.2). Being more future proof (better flexibility of proposal 2 when it comes to additional input vec-

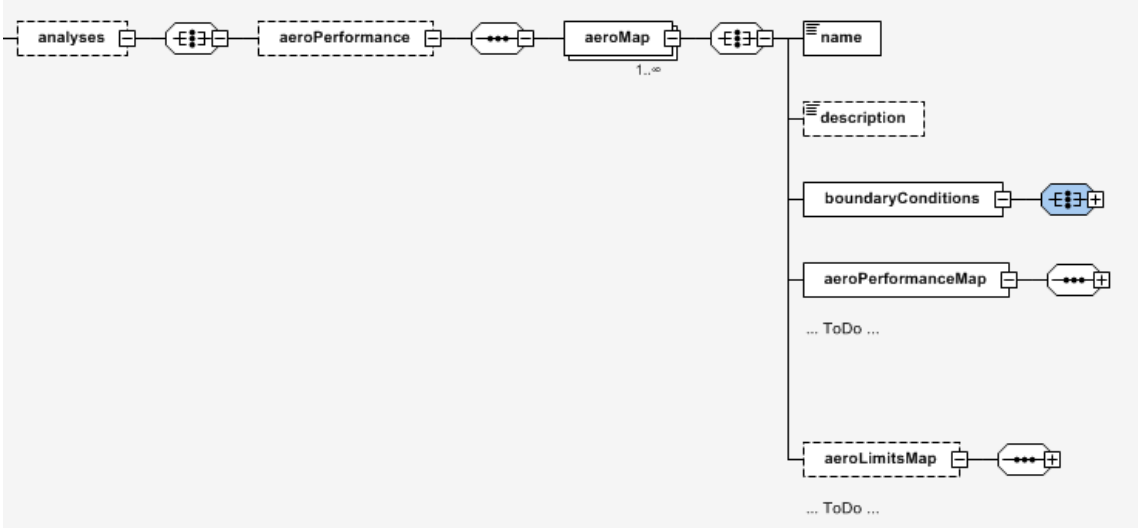


Figure 4.1: Backbone structure of the aeroPerformanceMap

tors) balances the drawback that we loose the semantic meaning of how to work with / interpolate in the aeroPerformanceMap.

4.3 Detailed schema definition

4.3.1 Introducing the backbone structure

The new aeroPerformanceMap will be embedded in the `aeroPerformance` element (see Fig. 4.1) which contains one ore more `aeroMaps`. This `aeroMap` is the container for a general description, `boundaryConditions`, the actual `aeroPerformanceMap` as well as an `aeroLimitsMap` (see Sec. 4.3.5).

The term `Map` will be used to refer to collections/sets of data. There will be only one `aeroPerformanceMap` and only one `aeroLimitsMap` to avoid redundancy and complexity be allowing the user to define multiple maps within the `aeroMap` node or using multiple `aeroMaps` instead.

4.3.2 Definition of boundary conditions

To better separate input and output values a `boundaryConditions` node combines information about the atmospheric model and the delta temperature to this model (see Fig. 4.2). Additionally the initial configuration of movable devices, such as control surfaces or landing gears, can be defined. Alternatively the user can refer to a control distributor.

This is a little step towards more high fidelity use-cases where no data about the clean configuration might be available, e.g. the high-lift configuration during the landing phase.

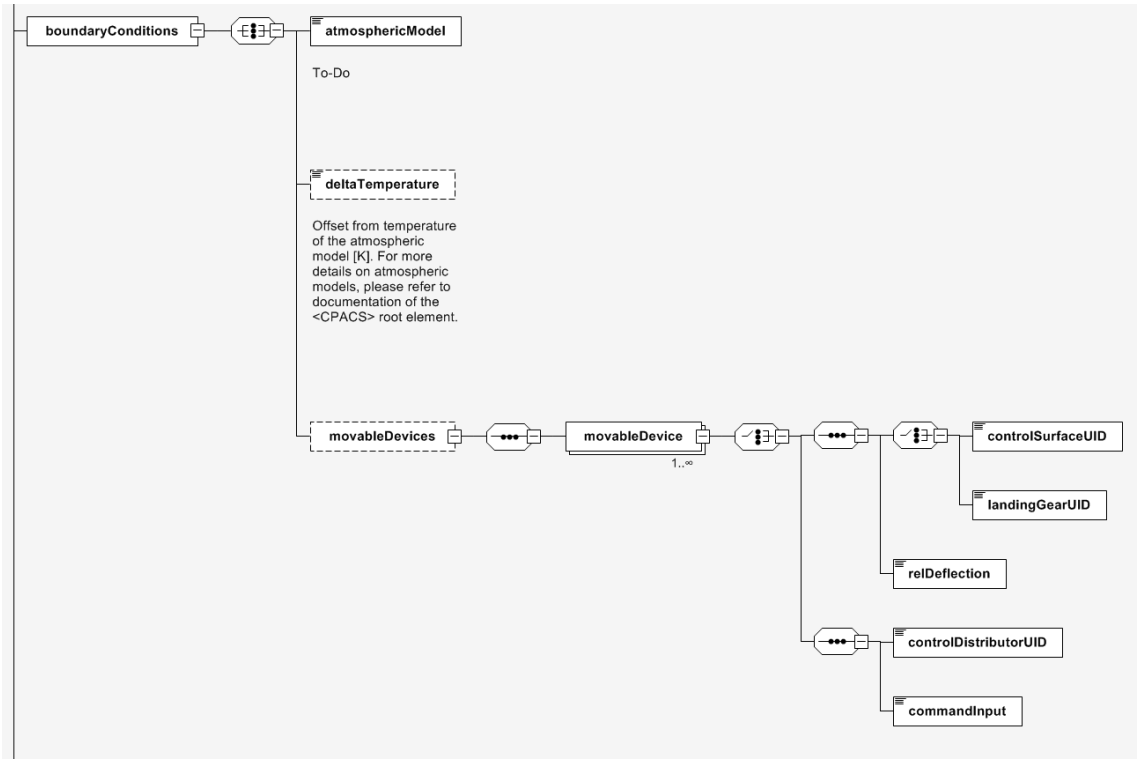


Figure 4.2: Boundary conditions of the aeroPerformanceMap

4.3.3 Definition of aeroPerformanceMaps

As decided in Sec. 4.2 the `aeroPerformanceMap` itself requires four input vectors: `altitude`, `machNumber`, `angleOfSideslip` and `angleOfAttack`. These vectors have the same length as the vectors containing the aerodynamic coefficients in the aerodynamic coordinate system, i.e. entries with the same index belong together. This holds as well for the `dampingDerivatives`.

4.3.4 Definition of sub-aeroPerformanceMaps

The sub-aeroPerformanceMap will be referred to as `incrementMaps` (see Fig. 4.4). A new feature is that now either a control surface (or landing gear) or a control distributor may serve as input parameter.

In order avoid redundancy in defining the initial configuration and to avoid redundant definitions compared to the control distributors it has been decided not to allow the combination of multiple control surfaces. If this is needed the user must define a corresponding control distributor.

It has been decided to define the `incrementMap` as two-dimensional array. The first dimension is the input of the main-aeroPerformanceMap and the second is the vector of relative deflections (or command inputs). This produces a little inconsistency in defining the maps (combination of vectors and arrays, i.e., not uniquely defined). However, the data is stored much more efficiently, i.e. we avoid defining a separate

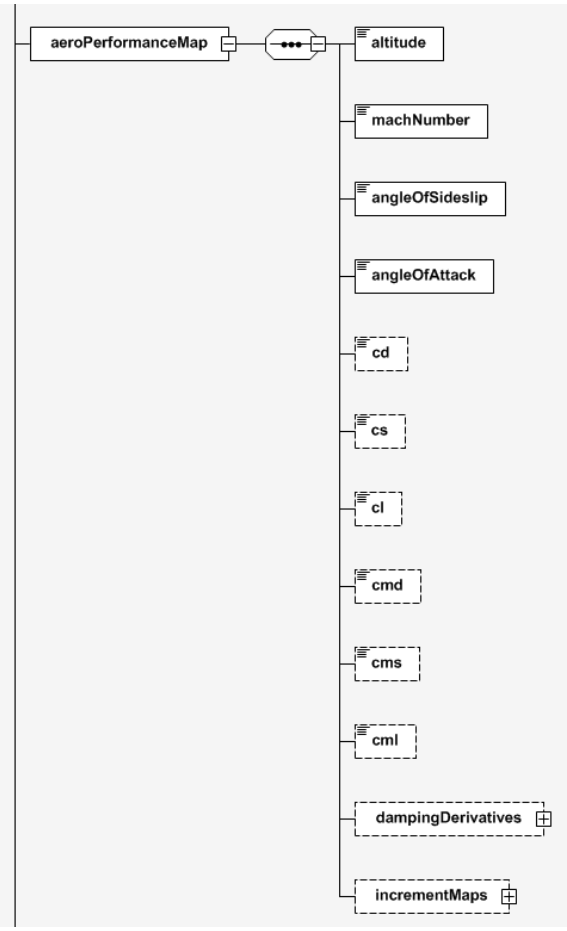


Figure 4.3: aeroPerformanceMap

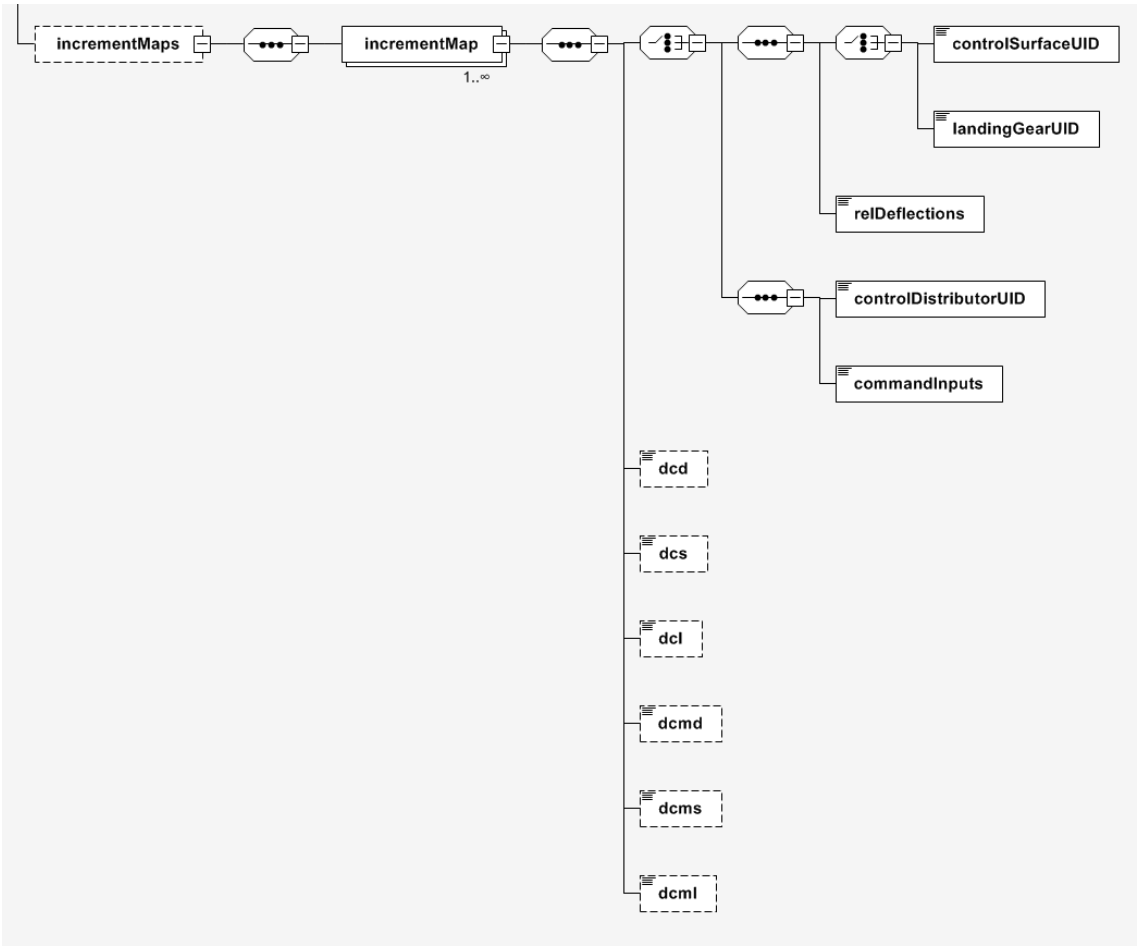


Figure 4.4: Increment Maps aeroPerformanceMap

node for each (combination of) control surface(s). This again underlines the need for a library function for convenient reading and writing the corresponding data.

4.3.5 Definition of limitations

The `aeroLimitsMap` hosts information about minimum and maximum angles in the aerodynamic coordinate system, e.g., the maximum angle of attack. As in the `aeroPerformanceMap` the `altitude` and `machNumber` serve as input vectors. The solution vectors containing the corresponding angles are having the same length as the input vectors. No corresponding aerodynamic coefficient is deliberately specified in order to avoid redundancy with the `aeroPerformanceMap`. If such coefficient needs to be determined from an existing CPACS dataset it must be extracted from the `aeroPerformanceMap` by direct querying or interpolation.

The `aeroLimitsMap` furthermore provides `incrementMaps` in the same way as used in the sub-aeroPerformanceMap (see 4.3.4). Since the relative deflections (or command inputs) are defined by vectors the increments are two-dimensional arrays.

It is still in discussion which kind of extrema values are required, e.g. the maximum flyable angle of attack may differ from the maximum angle of attack. However, this definition is flexible enough to add new variables whenever needed.

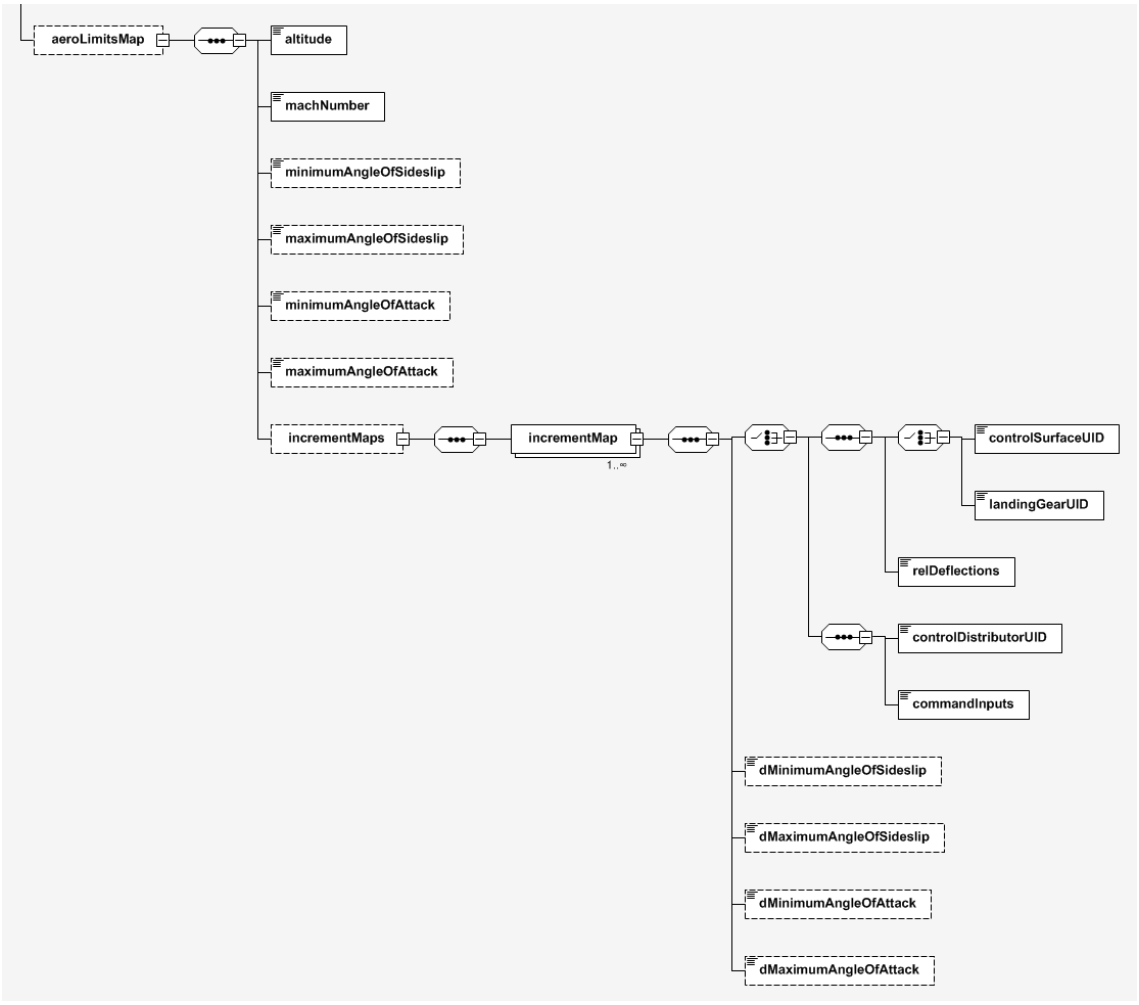


Figure 4.5: Handling limitations.

A Absolute deviations from 4D full-factorial

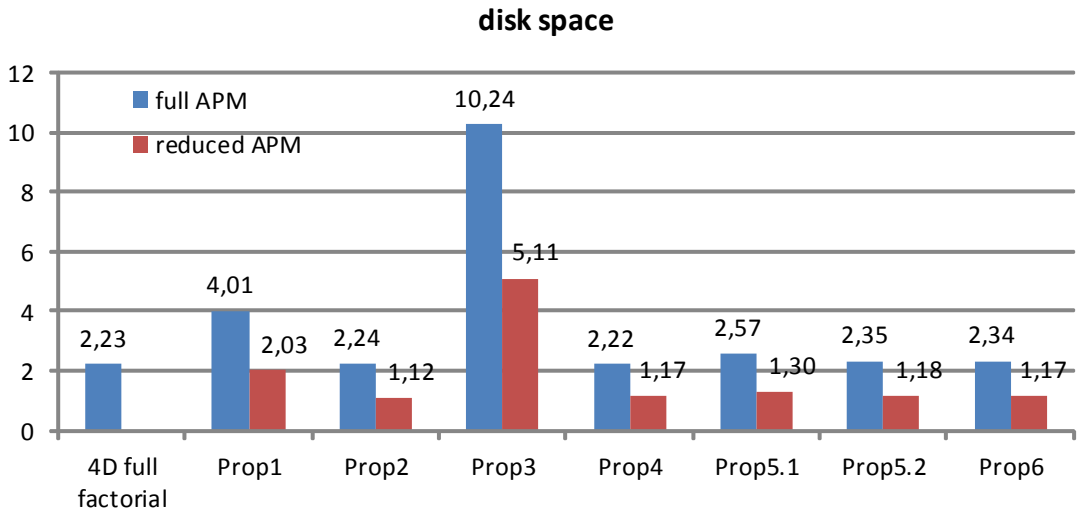


Figure A.1: Required disk space.

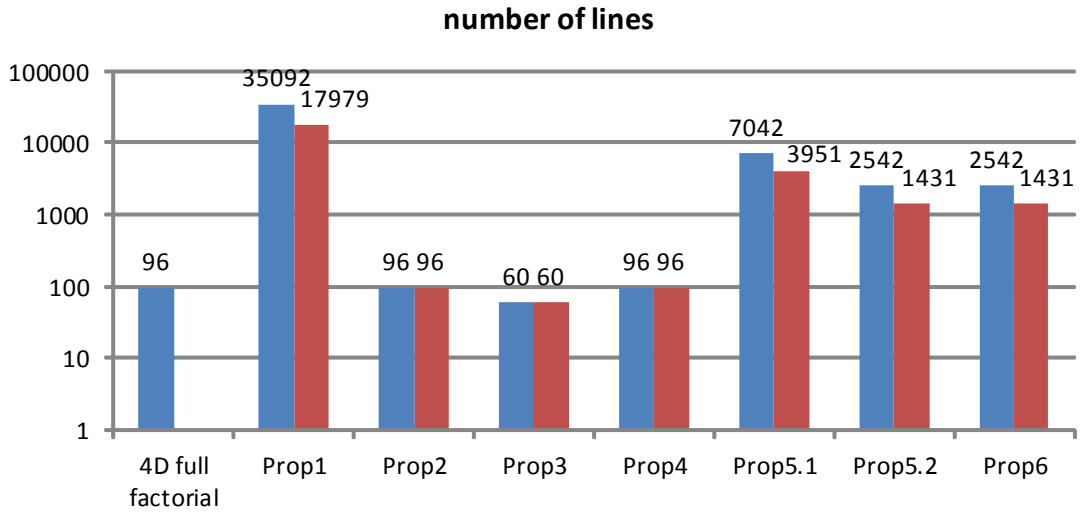


Figure A.2: Number of lines.

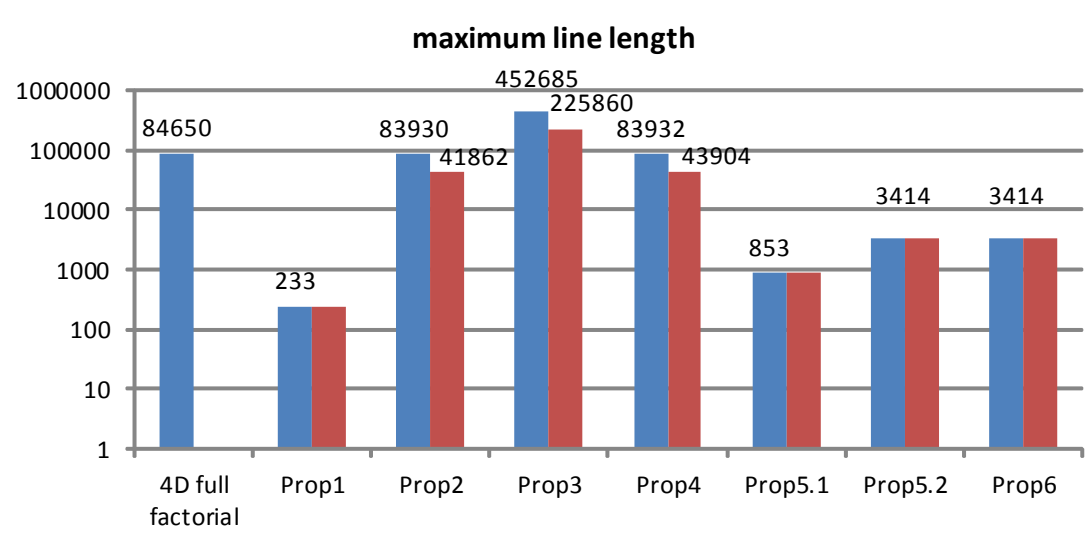


Figure A.3: Maximum line length.

B Relative deviations from 4D full-factorial

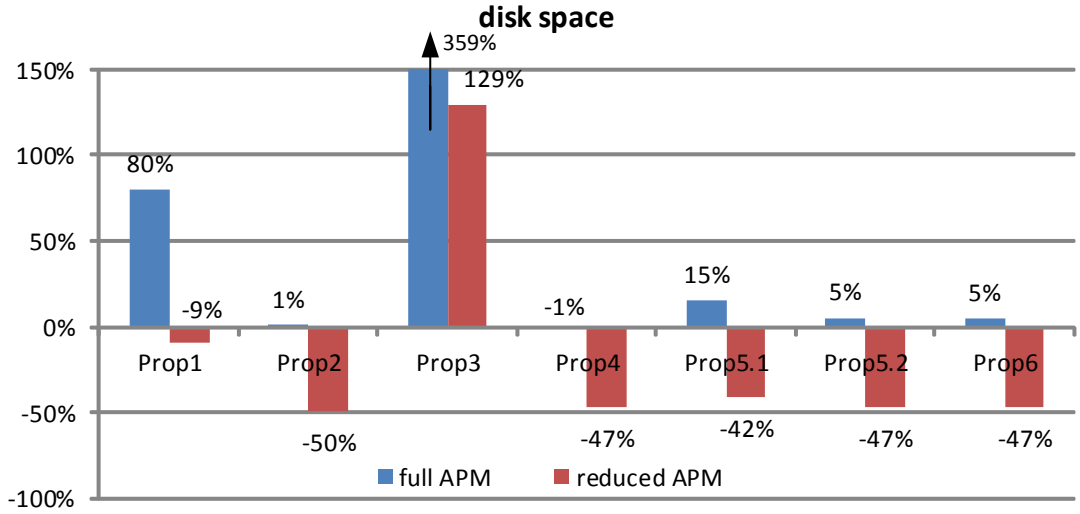


Figure B.1: Change in required disk space.

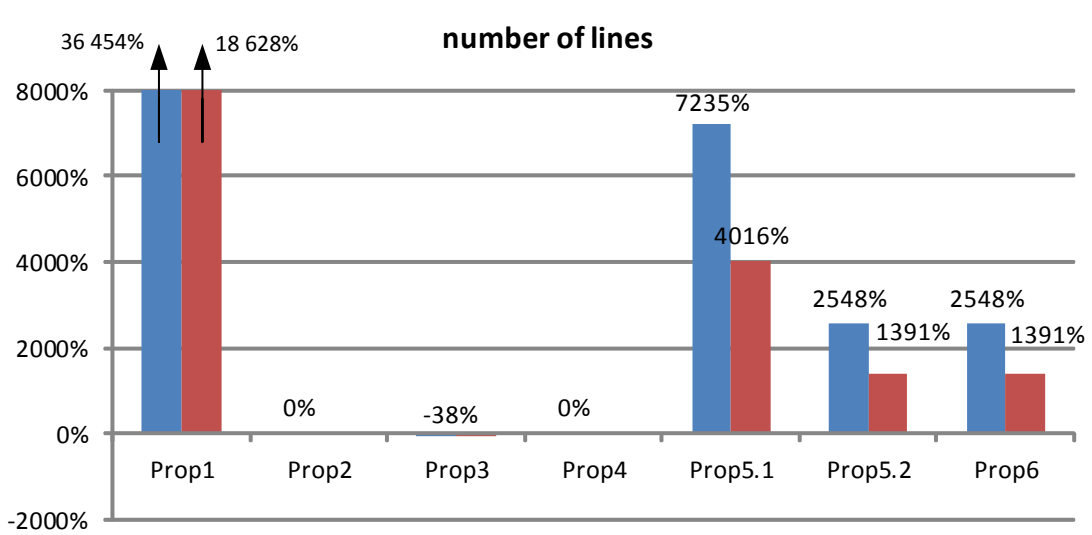


Figure B.2: Change in number of lines.

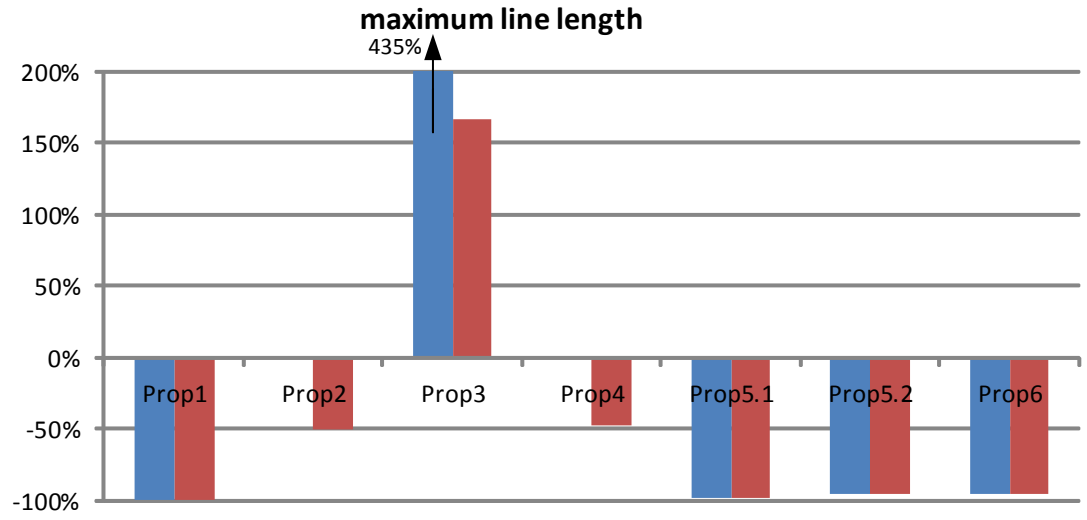


Figure B.3: Change in maximum line length.