NM 23.0.0 - NOP/B2B Reference Manuals - AIXM 5.1 Temporality Model Profile for NM B2B

Edition No.	:	23.0.0.5.26
Edition Issue Date	:	29/04/2019
Author	:	NM NOP/B2B Team
Reference	:	B2B/23.0.0/AIXMTemporalityModelProfile-ForNMB2B
Copy No.	:	← stamp here

Document Control

Copyright Notice

Document Identification

Full Title:	NM 23.0.0 - NOP/B2B Reference Manuals - AIXM 5.1 Temporality Model Profile for NM B2B
Total Number of Pages:	?

Table of Contents

Chapter 1. Need for an AIXM 5.1 Temporality Model Profile in NM

(1) The AIXM 5.1 Temporality Model is not fully applicable when mapping the CACD data to AIXM 5.1. The main difficulty is due to the fact that the CACD database always stores the full states of each object (which are in a way similar to AIXM 5.1 BASELINEs), not the changes between states. So the difficulty is that the PERMDELTAs are not available but need to be derived from the full states. At times it would be impossible to know whether a state is a corrected version of an existing state or if it is a new one. Hence we cannot make a meaningful use of the properties *correctionNumber* and *sequenceNumber*.

(2) Even without using the *sequenceNumber* and *correctionNumber* properties it is still possible to create PERMDELTAs to achieve the same final result, which is the new life of the object after applying the PERMDELTAs.

(3) The *sequenceNumber* and *correctionNumber* properties are especially needed when dealing with TEMPDELTA timeslices. For their nature, the TEMPDELTA timeslices always overlay on each other so that the latest TEMPDELTA "hides" all previous TEMPDELTAs during its validity period. If a TEMPDELTA is supposed to correct a previously given TEMPDELTA (e.g. to shorten its validity period) it needs to explicitly say that it is a correction and not another overlay.

(4) Currently there seem to be no need for using TEMPDELTA timeslices when exporting the CACD data. Should there be a need in the future the same result can be achieved without *sequenceNumber* and *correctionNumber*, like for the PERMDELTAs.

Chapter 2. Differences with respect to AIXM 5.1 Temporality

(1) The AIXM 5.1 Temporality Model Profile differs from the AIXM 5.1 Temporality Model in the following:

a) It does not use TEMPDELTA timeslices

b) It does not make use of *sequenceNumber* and *correctionNumber*

c) It allows communicating a decommissioning also using a BASELINE timeslice. This is useful because it allows the data consumers to choose to always ignore PERMDELTA timeslices and only process BASELINE timeslices, which are simpler and less error prone to apply.

Chapter 3. AIXM 5.1 Temporality Model Profile fo NM usage

(1) This chapter shows how the profiled temporality model is used by NM to export changes to the Airspace data. This is done by providing several examples.

3.1. Timeslices

(1) The AIXM temporality forces a data consumer to be able to process both BASELINE and PERMDELTA timeslices. BASELINE timeslices are easy to process because they always carry the full state of a Feature. PERMDELTA timeslices are however much more error prone to use. A small error in applying a PERMDELA timeslice may have large consequences because the error is propagated, causing the data consumers database to diverge from the original data source.

(2) The advantages of the PERMDELTA timeslices should be that they carry less information and they should pinpoint the changes, however if the change happens inside a complex property, the whole property must be re-issued, making the advantage of PERMDELTA very thin.

(3) In addition to the inherent complexity of the PERMDELTA timeslices, the NM systems impossibility to use the sequenceNumber and correctionNumber attributes make the PERMDELTA timeslices even more complex to use.

(4) For these reasons, the AIXM 5.1 Temporality Model Profile for NM is built in such a way that it allows the data consumer to choose whether to process only BASELINE timeslices or PERMDELTA timeslices too. In other words a data consumer may choose to safely and simply process only BASELINE timeslices.

(5) To this purpose the AIXM 5.1 Temporality Model Profile for NM defines a way to use a BASELINE timeslice also to communicate a decommissioning (for which only a PERMDELTA can be used in the AIXM Temporality Model).

(6) However, the AIXM 5.1 Temporality Model Profile for NM still provides full support for PERMDELTA timeslices.

(7) It is useful to define the following concepts:

a) **Commissioning timeslice**

b) **Decommissioning timeslice**

c) **Permanent change timeslice**

These concepts can be applied to both BASELINE and PERMDELTA timeslices.

3.1.1. BASELINE timeslices

3.1.1.1. Commissioning BASELINE timeslice

(1) It is a BASELINE timeslice that communicates a commissioning of a Feature. It contains:

a) *featureLifetime.beginPosition*

b) *featureLifetime.endPosition* (it can be unknown)

c) *validTime.beginPosition = featureLifetime.beginPosition*

d) *validTime.endPosition* (it can be unknown)

e) all the non null properties

3.1.1.2. Decommissioning BASELINE timeslice

(1) It is a BASELINE timeslice that communicates a decommissioning. It only contains a value for:

a) *featureLifetime.endPosition*

b) *validTime.beginPosition = validTime.endPosition = featureLifetime.endPosition*

3.1.1.3. Permanent Change BASELINE timeslice

(1) It is a BASELINE timeslice that communicates a permanent change. It contains:

a) *validTime.beginPosition*

b) *validTime.endPosition* (it can be unknown)

c) all the non null properties

3.1.2. PERMDELTA timeslices

(1) And the same concepts are defined for the PERMDELTA timeslices:

3.1.2.1. Commissioning PERMDELTA timeslice

(1) It is a PERMDELTA timeslice that communicates a commissioning. It contains:

a) *featureLifetime.beginPosition*

b) *validTime.instant = featureLifetime.beginPosition*

c) all the non null properties

3.1.2.2. Decommissioning PERMDELTA timeslice

(1) It is a PERMDELTA timeslice that communicates a decommissioning. It contains:

a) *featureLifetime.endPosition*

b) *validTime.instant = featureLifetime.endPosition*

3.1.2.3. Permanent Change PERMDELTA timeslice

(1) It is a PERMDELTA timeslice that communicates a permanent change. It contains:

a) *validTime.instant*

b) all the changed properties
A PERMDELTA timesice can also set the end of life to "unknown" to indicate an "undecommissioning", i.e. to extend the feature's life time to eternity for a feature that was previously decommissioned (see examples below).

3.2. PERMDELTA examples

(1) This paragraph illustrates the use of PERMDELTA timeslices by means of simple examples.

(2) Each example is based on an `AirportHeliport` Feature and simulates changes to the following properties:

a) *name:* a string

b) *cities:* a list of strings

c) *degrees:* a pair of decimal numbers

(3) The notation *all_other_fields* is used to indicate all the other properties other than the three mentioned above.

(4) The notation (...) in a timeslice means that the timeslice's end of time is undetermined.

(5) When a property undergoes a change of value between two consecutive timeslices, this is highlighted in red.

(6) Each example shows two consecutive versions of the same Feature, labelled *V1* and *V2* respectively. *V2* is a new version of the feature obtained by applying some changes to *V1.* Each example shows the lifetime of the Feature at version *V1* and that at version *V2.*

(7) Each example focuses on the production of the PERMDELTA timeslices and therefore shows the PERMDELTA timeslices that are exported by NM and that need to be applied to V1 in order to obtain V2.

3.2.1. Commissioning

(1) The commissioning of a feature is when the entity starts its life.

3.2.1.1. Example: Commissioning of a feature with a single timeslice

(1) The example shows the commissioning of a feature that contains a single timeslice (*V1* is null in the sense that the feature does not exist)

(2)

```
V1:          NULL

             T0
V2:          <------------------------------------------------------------ (...)
                cities={A}
                degrees=(45, 10)

PermDeltas:

       PD1:  |-> validTime.timeInstant=T0
             |-> featureLifetime.beginPosition=T0
             |-> cities={A}
             |-> degrees=(45, 10)
             |-> all_other_fields
```

(3) In this case a single PERMDELTA is published effective at T0, that sets the start of life of the Feature to T0 and contains all non null properties.

3.2.1.2. Example: Commissioning of a feature with multiple timeslices

(1) This example is similar to the above but the commissioned Feature already contains more than one timeslice.

(2) In this case a PERMDELTA is published for the commissioning and another one for the change that produces the second timeslice.

(3)

```
V1:          NULL

             T0                                      T1
V2:          <------------TS1----------><-------------------TS2--------------- (...)
               cities={A}                cities={A}
               degrees=(45, 10)          degrees=(45, 10.2)

PermDeltas:

       PD1:  |-> validTime.timeInstant=T0
             |-> featureLifetime.beginPosition=T0
             |-> cities={A}
             |-> degrees=(45, 10)
             |-> all_other_fields

       PD2:                             |-> validTime.timeInstant=T1
                                        |-> degrees=(45,10.2)
```

(4) The second PERMDELTA (PD2) only contains the changed attribute with respect to PD1.

3.2.1.3. Example: Earlier commissioning

(1) In this example the Feature was published as being commissioned at time T1 but its commissioning was than moved to T0.

```
                                    T1
V1:                                 <---------------------------------------- (...)
                                       cities={A,B}
                                       degrees=(45,10)
            T0
V2:         <------------------------------------------------------------------ (...)
               cities={A,B}
               degrees=(45,10)

PermDeltas:

       PD1:  |-> validTime.timeInstant=T0
             |-> featureLifetime.beginPosition=T0
             |-> cities={A,B}
             |-> degrees=(45,10)
             |-> all_other_fields
```

3.2.1.4. Example: Delayed commissioning

(1) In this example the Feature was published as being commissioned at time T0 but its commissioning was than postponed to T1.

(2) This is done by re-publishing a new PERMDELTA timeslice that postpones the Features start of life.

```
           T0
V1:        <------------------------------------------------------------------ (...)
              cities={A}
              degrees=(45,10)

                       T1
V2:                    <------------------------------------------------------ (...)
                          cities={A}
                          degrees=(45,10)

PermDeltas:

       PD1:            |-> validTime.timeInstant=T1
                       |-> featureLifetime.beginPosition=T1
                       |-> cities={A}
                       |-> degrees=(45,10)
                       |-> all_other_fields
```

3.2.1.5. Example: Undo Commissioning

(1) The example shows a feature that was supposed to be commissioned at time T0, but it was then decided to withdraw it.

(2) This is done by issuing a decommissioning PERMDELTA effective at T0.

```
            T0
V1:         <---------------------------------------------------------- (...)
```

```
               cities={A}
               degrees=(45,10)

            T0
V2:         | (stopped)

PermDeltas:

       PD1: |-> validTime.timeInstant=T0
            |-> featureLifetime.endPosition=T0
```

3.2.2. Decommissioning

(1) The decommissioning of a feature is when the entity ends its life.

3.2.2.1. Example: Decommissioning of a feature

(1) The example shows the decommissioning of a feature. The feature had an undetermined end of life, so a PERMDELTA timeslice is issued to set its end of life.

```
V1:       <----------------><------------------------------------------------------ (...)
            cities={A}          cities={A}
            degrees=(45,10)     degrees=(45,10.2)

                                                    T0
V2:       <----------------><--------------------->| (stopped)
            cities={A}          cities={A}
            degrees=(45,10)     degrees=(45,10.2)

PermDeltas:

       PD1:                                         |-> validTime.timeInstant=T0
                                                    |-> featureLifetime.endPosition=T0
```

3.2.2.2. Example: Earlier decommissioning

(1) The example shows a feature that was supposed to be decommissioned at time T1, but its decommissioning was moved to time T0, effectively removing an entire timeslice.

```
                                                                                  T1
V1:       <----------------><-------------------------------><-------------->| (stopped)
            cities={A}          cities={A}                      cities={A,B}
            degrees=(45,10)     degrees=(45,10.2)               degrees=(45,10)

                                                      T0
V2:       <----------------><---------------------->| (stopped)
            cities={A}          cities={A}
            degrees=(45,10)     degrees=(45,10.2)

PermDeltas:

       PD1:                                           |-> validTime.timeInstant=T0
```

```
                                                                |-> featureLifetime.endPosition=T0
```

3.2.2.3. Example: Delayed decommissioning

(1) The example shows a feature that was supposed to be decommissioned at time T0, but its decommissioning was postponed to time T1 by adding a new timeslice.

```
                                                T0
V1:        <---------------><------------------>| (stopped)
             cities={A}        cities={A}
             degrees=(45,10)   degrees=(45,10.2)

                                                                  T1
V2:        <---------------><------------------><---------------->| (stopped)
             cities={A}        cities={A}          cities={A,B}
             degrees=(45,10)   degrees=(45,10.2)   degrees=(45,10.2)

PermDeltas:

       PD1:                                      |-> validTime.timeInstant=T0
                                                 |-> featureLifetime.endPosition=unknown
                                                 |-> cities={A,B}

       PD2:                                                  |-> validTime.timeInst...=T1
```

(2) NOTE: The first PERMDELTA timeslice (PD1) that communicates the permanent change to the *cities* property must also set the end of life to "*unknown*", to undo the decommissioning previously communicated in *V1.*

(3) The second PERMDELTA timeslice (PD2) will communicate the new decommissioning time.

3.2.2.4. Example: Undo decommissioning

(1) The example shows a feature that was supposed to be decommissioned at time T0, but its decommissioning was cancelled, i.e. the end of life is unknown.

```
                                                   T0
V1:        <------------------><----------------------->| (stopped)
             cities={A}          cities={A}
             degrees=(45,10)     degrees=(45,10.2)

V2:        <------------------><-------------------------------------------------- (...)
             cities={A}          cities={A}
             degrees=(45,10)     degrees=(45,10.2)

PermDeltas:

       PD1:                                          |-> validTime.timeInstant=T0
                                                  |-> f...Lifetime.endPosition=unknown
```

3.2.3. Permanent changes

(1) This is the nominal case for a PERMDELTA.

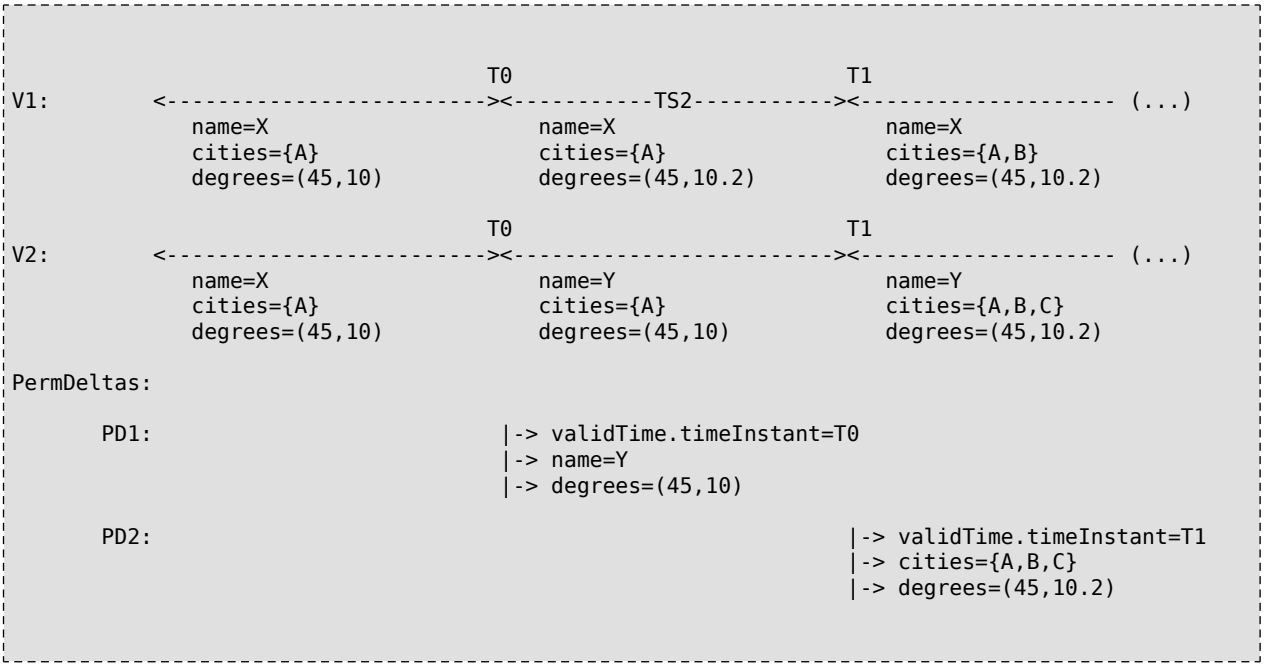
3.2.3.1. Example 1

(1) The example shows the evolution of a feature from version *V1* to version *V2.* The feature maintains its three timeslices, but the properties undergo the following changes:

a) The *name* property changes at T0

b) The change of *degrees* is postponed until T1

c) The cities property still changes at T1 but it has now a new value.



(2) Note that the first PERMDELTA timeslice (PD1) must re-communicate the value of the property degrees because it has to overrule what was published in timeslice TS2

3.2.3.2. Example 2

(1) The example shows a case in which some changes to a feature are anticipated. In particular the change of property cities is anticipated from T1 to T0.

```
              cities={A}                          cities={A,B}
              degrees=(45,10)                     degrees=(45,10)

                                    T0
V2:         <-------------------------><---------------------------------- (...)
              cities={A}                  cities={A,B}
              degrees=(45,10)             degrees=(45,10)

PermDeltas:

       PD1:                            |-> validTime.timeInstant=T0
                                       |-> cities={A,B}
```

3.2.3.3. Example 3

(1) The example shows several changes effective at different times.

```
V1:          <------------------------------------------------------------------------ (...)
               cities={A}
               degrees=(45,10)

             T0                      T1                                  T2
V2:          <---------------------><---------------------><------------------------ (...)
               cities={A,B}            cities={A,B,C}          cities={A,B,C}
               degrees=(45,10)         degrees=(45,10)         degrees=(45,10.2)

PermDeltas:

       PD1:  |-> validTime.timeInstant=T0
             |-> cities={A,B}

       PD2:                          |-> validTime.timeInstant=T1
                                     |-> cities={A,B,C}

       PD3:                                                 |-> validTime.timeInstant=T2
                                                            |-> degrees=(45,10.2)
```

3.2.3.4. Example 4

(1) The example shows the following:

a) The change of the attribute *cities* is anticipated from T1 to T0.

b) The change of the property *degrees* is postponed from T1 to T2.

```
             T0                       T1                          T2
V1:          <----------TS1----------><-------TS2-------><----------------TS3----------- (...)
               cities={A}               cities={A,B}       cities={A,B}
               degrees=(45,10)          degrees=(45,10)    degrees=(45,10.2)

             T0                                                  T2
```

```
V2:           <----------------------------------------><------------------------------ (...)
                 cities={A,B}                              cities={A,B}
                 degrees=(45,10)                           degrees=(45,10.2)

PermDeltas:

       PD1:   |-> validTime.timeInstant=T0
              |-> cities={A,B}
```

(2) NOTE: It is not necessary to issue a second PERMDELTA to re-communicate the new value of *degrees* because what was published with V1.TS3 is still valid (A PERMDELTA has effect only on the properties it contains).