Using the .bak file for data-delivery for the instrumentholders of NMD3.0

Date: 20 February 2020
Version: 1.1
Author: R.H. Dijkstra, RBit Informatica BV, commissioned by SBK

Table of Contents

Preface

This document comes with the .bak file for the delivery of the computable data of the NMD3.0 to the instrumentholders, who can then use this data to make their MKI and MPG computations for buildings in the Dutch market.

This document specifically deals with the practical issues involved with the use of the data-delivery file.

the practical details of the accompanying new Endpoints in the API for retrieving updates en newly published product-cards will be dealt with in later documents (which will become available when these Endpoints are available)

The technology and content of the .bak file

Use of SQLSERVER

We have chosen to use SQLSERVER .bak file for the delivery of the data in NMD3.0, for a number of practical reasons.

1. Ease of use for the instrumentholder. All you really need for the transferral of the data using a .bak file is a PC with windows (probably 8.1 or later), and SQLSERVER EXPRESS edition that you can download for free (!) (if you do not have a developers license) from Microsoft and install in a few minutes. In the Netherlands, it is legal to use SQL EXPRESS that for this kind of purpose
2. Though SQLSERVER EXPRESS is actually limited in some ways (maximum number of cores used, maximum memory space used), that should cause no problem with this NMD database (we tested this). Of course with a developer's license, these limitations do not apply.
3. All you need to do then is to install Visual Studio or some other programming environment on the PC that can talk to SQLSERVER, and you can quickly and simply write a bit of code to transfer the data to any other DB, data-format or caching-structure that you like.
4. Ease of documentation: in SQL EXPRESS you have all the data on every field (data-type, nullability, etc) immediately at your disposal, without any extra effort.
5. Therefore we think that this by far the most flexible, cost-efficient and reliable method for the transferral of the data...

Only relevant Published data

We have stripped the .bak of all information that is not pertinent to use by the instrumenholders

- All technical helper tables that are not applicable for the instrumentholder and all management tables about accounts data-owners and the like have been removed
- Some fields and tables have been rearranged to make them more applicable for the specific use case of the instrumentholder
- The .bak file only contains officially Published data.

This .bak file contains all available NMD data per the date of this document

This .bak contains all officially published data available in NMD3.0. That is to say:

1. All converted product-cards from NMD2.3 (Utility buildings)
2. All Product-cards (“ItemObjects”) from DuboCalc (Ground, Roads, and Waterways)
3. A few dozen new “Real” NMD3.0 product cards (this number will only grow from now on
4. The endpoints for updating en logging are on the work list for realization. Untill they are available you use the data in the datadelivery file “as is” (but you will be required to use the update and logging endpoints when they are made available!)

The relationship between the .bak and the coming Endpoints for updating and logging

The Update Endpoint

The new Endpoint for updating your data caches will be coming soon. Once in service, this Endpoint will be updated every day to contain the publications and updates for product records of the specific date.

This Endpoint will also work with the “ZoekDatum” (SearchDate) parameter so that when for instance you have missed a few updates (which could make the data inconsistent), you can easily solve this by consecutively retrieving the data for the days you (or your software) missed

Instrumentholders are expected to implement a process that automatically retrieves the updates en publications every day and does so between 02 am and 05 am in the morning (before office hours, and after our maintenance window…)

(an alternative that we are thinking about is that an automatically updated copy of the .bak database will be made available for download every day and be removed after a day, when the next one is available, which you can then download, and use to update your caches. The decision on this is dependent on the efficiency and cost of both alternatives)

Logging Endpoint.

SBK (the owner of the NMD) had been commissioned by the Dutch government to

- Monitor the computational processes of the instrumentholders and the MKI’s and MPG’s computed,
- to guard the quality of those processes and,
- to provide statistical information that the government can use to make policy.

For each of those goals, SBK needs a minimum of information about the use that the instrumentholders make of the data in the NMD(3.0).

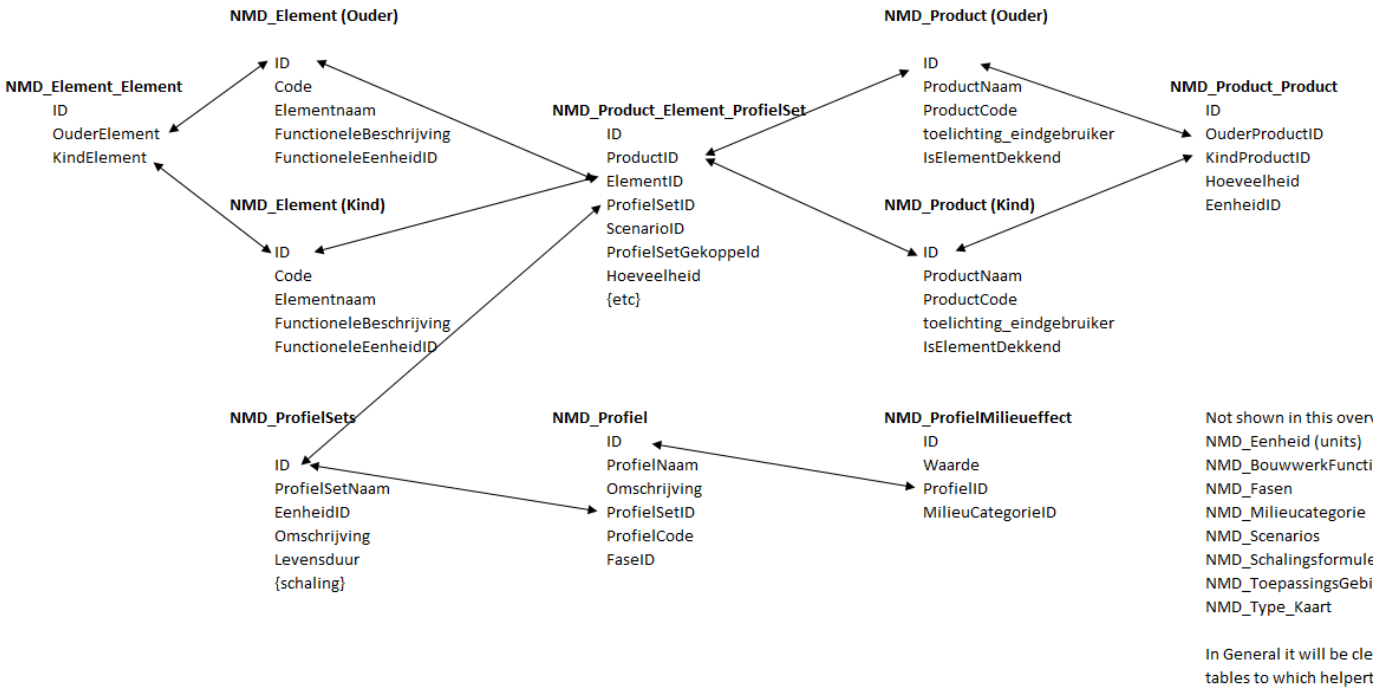
To enable this a data-logging Endpoint will be implemented, to which an anonymized and limited dataset about each building that is processed must be written. On the exact specification of the data that we expect in this logging Endpoint and on the mechanism of delivery (how and when is the data to be provided, etc.), later communications will follow.

The structure of the data in the .bak file

As stated, the .bak file that is used for the delivery of the data of the NMD3.0 is a version of the NMD3.0 database that has been cleaned of all information that the instrumentholders do not need.

The relational structure of the tables

The relational structure of the content tables in this DB is shown in the Excel sheet that comes with this document, and below.



Additional remarks on linking structure:

1. For clarity of exposition, the relations depicted here are between base tables. In actual fact, each of those base-tables is coupled with a **versioning table** that contains the actual data.
2. The core table or master table of the whole structure is NMD_Product_Element_ProfielSets that links Elements, Products, and ProfileSets together.
3. In the upper part of the graph, you see two “diamonds”. The two different routes in each of those diamonds depict the difference between relationships for the “Totaalproducten” (element-covering products, or “total products”) and the relationships for the “Deelproducten” (partially element-covering products, or partial products for short)
4. For the Parent products, the relationship between the ProductID and the foreign key in the master table points directly from the productID in the master table.
5. The same thing goes for the corresponding Parent element that the Parent product is linked to.
6. In the case of a **Totaalproduct** (total product in English), the graph then directly continues from the parent product and the parent element to the profielsets (profile sets), the profielen (profiles) and ultimately the environmental impact values in the table NMD_ProfielMilieueffect
7. For partial products, however, an extra “loop” through the bottom part of each diamond is necessary: parent product and parent elements are linked to their children (product parts and element parts respectively) through the hierarchical linking tables:
   - MD_Product_Product
   - NMD_Element_Element.
8. For the partial products, the profiles and environmental impact values are now linked to the child products (product-parts) and child elements through the table

NMD_Product_Element_ProfielSets. (This Linkage to the child elements is important for the analysis of hiatuses in the coverage of the elements). The same goes for the “free rider” product parts that have no calculated output values, but that do cover element parts.

9. Thus for partial products, the linkage to the actual environmental data runs from the child products to the profile sets to the profiles to the impact data.

Meelifters or “free riders”

We already mentioned the meelifters or “free riders”. These are product parts that are added by the data owner to show that the specific child element or element part is actually covered by the product, although for the specific product part there are no separate environmental impact data provided to calculate with.

In the database, this free-rider status is represented by the fact that although there actually is a Profile set record, the quantity that goes with its foreign key is always -1 (and also there are no linked records in the tables for Profiles and impact data.

The implementation of scenarios

In the NMD3.0 A provision is made for the use of scenarios, especially pertaining to the handling of End of Life and Waste.

However, until now these scenarios have not been implemented. Thus, all products are by default linked to scenario nr.1 which stands for “Default scenario”. And because this is always the case this link is not explicitly stored in the database (that would be very redundant). Thus for 99,9% of products when you search in the linking table “NMD_Producten_Scenarios_Versies” for the product ID of a specific product you will find no record, which is **equivalent** to a link to the default scenario, with ID = 1.

Base tables and versioning tables

In the above, we tried to keep things as tractable as possible by not distinguishing between base tables and their versioning counterparts.

However, for each of the base tables mentioned above, there is a duplicate that contains exactly the same columns, but with (always the same) four extra columns, and a postfix added to its name, following the format:

{TableName} + _Versies

Extra fields/columns in the versioning tables

As said the versioning tables always have the same 4 extra columns:

- {TableName}ID
- DatumActief
- DatumInActief
- VersieNr

Where the column **{TableName}ID** is the foreign key that links the versioning table to the base table. Thus f.i. the base table “NMD_Product” has a versioning table “NMD_Product_Versies”, where the column ProductID in NMD_Product_Versies points to the column (and primary key) “ID” in the table NMD_Product. The same goes, ceteris paribus, for **all the tables shown in the graph above.**

DatumActief is the column that contains the date for which the active status for the specific record starts

DatumInActief is the name of the column that contains the date at which the specific record has been deactivated (and thus no longer available for computations). Normally, if DatumInActief is set there will also be a new record (= version) with a DatumActief that is equal to the DatumInActief.

VersieNR (version number) is a handy field that enables to you directly establish the index of the published version in the history of the specific items (=> should always be equal to the version NR of the product because products are always published integrally), without running logic on DatumActief and DatumInActief or IDs.

On the Uniqueness of records and versions

Definition: we say that a record is "active" for a certain day when for that day it has been published/activated and it has not been de-activated again.

In terms of the information in the database, this means that the record is active iff:

1. The record has DatumActief **<=** the search date (ZoekDatum) AND
2. The record has a DatumInActief that is NULL, OR the record has a DatumInActief **>** search date

Together the columns DatumActief and DatumInActief guarantee the uniqueness of the data retrieved. Without the DatumInActief this would not be the case, because an administrator can do several publications on the same day, which, in our setup where all publications are done per 00:00:00 hours (and at the earliest the next day), would lead to the possibility of several records with exactly the same date and time of publication, and therefore not be unique in terms of publication date and time. Which would create a "multiple records problem" when we try to search for an item where the search date is equal to this date. However, the addition of the DatumInActief and the second condition above prevents this problem from occurring because the publication process is such that with each publication of a record the previously "active" record is made inactive starting at exactly the same moment as the newly published record becomes active, and the second condition excludes records where DatumActief and DatumInActief are the same. Thus if several publications of an item have been done for the same day, only the last publication done for that specific day will have a DatumInActief that is either NULL or > Search Date (as shown in the example below).

ID	ProductID	ProductNaam	DatumActief	DatumInActief
11632	15907	Deelproduct: Verhardingen in kg, Asfalt...	2019-08-21 00:00:00.000	*NULL*
11633	15907	Deelproduct: Verhardingen in kg, Asfalt...	2019-08-19 00:00:00.000	2019-08-21 00:00:00.000
11630	15907	Deelproduct: Verhardingen in kg, Asfalt...	2019-08-19 00:00:00.000	2019-08-19 00:00:00.000
11631	15907	Deelproduct: Verhardingen in kg, Asfalt...	2019-08-19 00:00:00.000	2019-08-19 00:00:00.000
11616	15907	Deelproduct: Verhardingen in kg, Asfalt...	2019-08-17 00:00:00.000	2019-08-19 00:00:00.000

Example of a very basic query that implements this structure:

```
USE NMD_dataDelivery_okt2019
DECLARE @DAT DateTime; SET @DAT = '2019-08-19 00:00:00.000';
SELECT * FROM NMD_Product_Versies WHERE DatumActief <= @DAT AND (ISNULL(DatumInActief
,0) <= 0 OR ISNULL(DatumInActief ,0) > @DAT )
ORDER BY ID DESC
```

What we see here is a query that retrieves all product records as they were active on the date 18th of august 2019. When you run this query in your query designer in the SQL SERVER Management Studio, you will see that it neatly only selects the last of the 15907 records published on 2019-08-19

The crucial part of select statements in the NMD3.0 (!!)

What the above illustrates is that in your select statement you should always AND FOR EVERY CONTENT TABLE (!) include this snippet of SQL (or some equivalent SQL that produces the same result…):

```
(…) {WHERE/AND} DatumActief <= @DAT AND (ISNULL(DatumInActief ,0) <= 0 OR
ISNULL(DatumInActief ,0) > @DAT )
```

Example of a complete query:

As an illustration of the above, we hereby give an example of a nested query that descends the structure shown in the graph above for a **Partial Product** and retrieves all impact values for one specific productID:

```
USE NMD_dataDelivery_okt2019
DECLARE @DAT DateTime; SET @DAT = '20191016 00:00:000';
DECLARE @PRODUCTID INT; SET @PRODUCTID = 15907;
SELECT * FROM NMD_ProfielMilieueffect_Versies PMEV WHERE ProfielID IN (
SELECT ProfielID FROM NMD_Profiel_Versies PV WHERE ProfielSetID IN
(
SELECT ProfielSetID FROM NMD_ProfielSets_Versies WHERE ProfielSetID IN
(
SELECT ProfielSetID FROM NMD_Product_Element_ProfielSet_Versies WHERE ProductID IN (
SELECT KindProductID FROM NMD_Product_Product_Versies WHERE OuderProductID =
@PRODUCTID AND  DatumActief <= @DAT AND (ISNULL(DatumInActief ,0) <= 0 OR
ISNULL(DatumInActief ,0) > @DAT )
) AND DatumActief <= @DAT AND (ISNULL(DatumInActief ,0) <= 0 OR ISNULL(DatumInActief
,0) > @DAT )
)  AND DatumActief <= @DAT AND (ISNULL(DatumInActief ,0) <= 0 OR ISNULL(DatumInActief
,0) > @DAT )
)
AND DatumActief <= @DAT AND (ISNULL(DatumInActief ,0) <= 0 OR ISNULL(DatumInActief ,0)
> @DAT )
) AND DatumActief <= @DAT AND (ISNULL(DatumInActief ,0) <= 0 OR ISNULL(DatumInActief
,0) > @DAT )
```

NOTE: in the above, we technically could have omitted the join on the table NMD_ProfielSets and directly joined the ProfielSetID in NMD_Profiel_Versies to the ProfielSetID in NMD_Product_Element_ProfielSet_Versies, which would actually be more efficient, but we included this join in the Example because we wanted to illustrate the descending through the complete structure of tables plus the recursive addition of the conditions on the DatumActief and DatumInActief for all tables in the JOINs.

Further remarks on the data structure

indexes

All indexes except for the standard clustered index on the primary keys fields (IDs) have been removed. You can add your own indexes if you want to.

Recognizing converted NMD2.3 records

All converted records from NMD2.3 can be recognized by the fact that they have a record in the table NMD_Product_Attributen, which acts as a universal holdall for additional information on products.

The column Product_ID in this table is the foreign key that links the record to the specific NMD3.0 product.

The column “attribuut” gives the attribute that is described

The columns “Waarde1”, “Waarde2” and “Waarde3” contain the specific values for the specific attribute of this product. (see below for an illustration)

Converted NMD2.3 products can be recognized by an Attribute “NMD23_ProductId”, where column “Waarde1” will hold the original ID of this product in NMD2.3.

NDM2.3 scaling information.

Because of fundamental reasons having to do with respective data structures in the NMD2.3 and the NMD3.0, which have been explained in a number of elaborate mails[1], and which will not go into here, it was not possible to include the scaling information for the converted records from NMD2.3 directly into table NMD_ProfielSets whereas NMD3.0 stores the scaling information[2].

Because this scaling information in the NMD2.3 applies to the product as a whole (with the exception of the “schaalfactor” or “scaling factor” to which we will return later) we stored this information in the table NMD_Product_Attributen.

In the NMD2.3 there were three kinds of scaling:

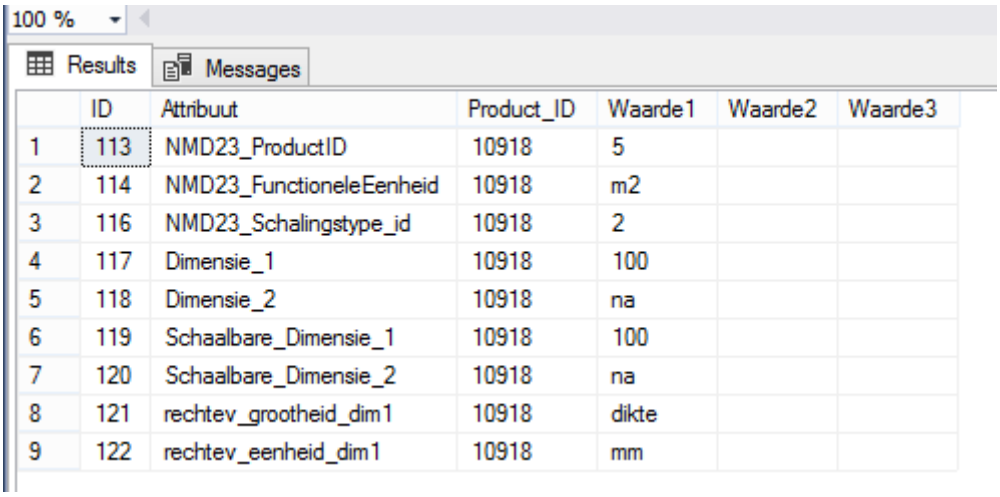
1. No scaling
2. “Rechtevenredig” (which means something like “straightly proportional..”)
3. “Massa volgens tabel” (mass according to a table)

Below we two examples to show how these data are stored in the table NMD_product_Attributen:

Proportional scaling Example:

[1] This explanation is available from the author of document for those are interested.

[2] We do not go into the explanation of the NMD2.3 scaling mechanisms here, because we assume that the instrumentholders will have this knowledge. If this is not the case, the rules for calculating (including scaling) in the NMD2.3 are available at SBK, or the author of this document.



100 %

Results | Messages

	ID	Attribuut	Product_ID	Waarde1	Waarde2	Waarde3
1	113	NMD23_ProductID	10918	5		
2	114	NMD23_FunctioneleEenheid	10918	m2		
3	116	NMD23_Schalingstype_id	10918	2		
4	117	Dimensie_1	10918	100		
5	118	Dimensie_2	10918	na		
6	119	Schaalbare_Dimensie_1	10918	100		
7	120	Schaalbare_Dimensie_2	10918	na		
8	121	rechtev_grootheid_dim1	10918	dikte		
9	122	rechtev_eenheid_dim1	10918	mm		

Here we see how the NMD2.3 scaling info is stored for a product with ID “10918” in NMD3.0, which had product ID “5” in the NMD2.3, and which has proportional scaling

Scaling information at the level of ProfileSets:

For the products with proportional scaling in NMD2.3, there also was a “schaalfactor” or “scaling factor” which in NMD2.3 is stored at the level of product-parts. Now the product-parts in NMD2.3 were converted to ProfielSets in NMD3.0. But in NMD3.0 we do not have a table with attributes at the level of ProfielSets, such as we have at the level of products.

To solve this we added a column “NMD23_SchaalFactor” to the table NMD_ProfielSets, which holds the information about the scaling factor for the NMD2.3 information that used to fill the specific record, if applicable…

A “mass according to table” Example:

To start with it is important to note that the mechanism for scaling a product through “mass according to table” in NMD2.3 actually amounts to choosing between different implementations of the same product with different dimensions.

In NMD3.0 a separate structure for variants of the same product is not available, but we solved this by storing the different implementations of the product as separate products and then using the Attributes table to hold the information that these products are variations of the same product.

If for instance we look at the product with ID = 11000 in NMD3.0 we use this query to find the attributes:

```
DECLARE @DAT DateTime; SET @DAT = '2019-10-17 00:00:00.000';
SELECT        TOP (200) ID, Attribuut, Product_ID, Waarde1, Waarde2, Waarde3
FROM            NMD_Product_Attributen_Versies
WHERE       Product_ID = 11000
AND DatumActief <= @DAT AND (ISNULL(DatumInActief ,0) <= 0 OR ISNULL(DatumInActief ,0)
> @DAT )
```

Which produces the following output:

100 %

Results | Messages

	ID	Attribuut	Product_ID	Waarde1	Waarde2	Waarde3
1	718	NMD23_ProductID	11000	11398		
2	719	NMD23_FunctioneleEenheid	11000	m1		
3	721	NMD23_Schalingstype_id	11000	3		
4	722	NMD23_ProductVariantID	11000	8		
5	723	massa_volgens_tabel_grootheid	11000	diameter		
6	724	massa_volgens_tabel_eenheid	11000	mm		
7	725	VariantDimensie_1	11000	400		
8	726	VariantDimensie_2	11000	na		
9	727	ProductVariant	11000	diameter 400	400	

We see here that NMD23_Schalingstype_id = 3, thus this is a case of “mass according to table”, and we also see the other data that we need to do the calculations, however, “mass according to table” implies that you have to select a variant of the product. Therefore we want to know which NMD3.0 products are actually NMD2.3 variants of this product.

We can quickly find this information because we know that these NMD3.0 products must have had the same product ID in NMD2.3. (11398 in this case) So we can query the NMD_Product_Attributen table for all NMD3.0 products with the value (waarde1) 11398 and attribute ‘NMD23_Product’ where waarde1 = 11398.

```
DECLARE @DAT DateTime; SET @DAT = '2019-10-17 00:00:00.000';
SELECT        TOP (200) ID, Attribuut, Product_ID, Waarde1, Waarde2, Waarde3
FROM            NMD_Product_Attributen_Versies
WHERE       Attribuut = 'NMD23_ProductID' AND Waarde1 = '11398'
AND DatumActief <= @DAT AND (ISNULL(DatumInActief ,0) <= 0 OR ISNULL(DatumInActief ,0)
> @DAT )
```

This produces the following output:

Results | Messages

	ID	Attribuut	Product_ID	Waarde1	Waarde2	Waarde3
1	1435	NMD23_ProductID	11000	11398		
2	1455	NMD23_ProductID	11001	11398		
3	1475	NMD23_ProductID	11002	11398		
4	1495	NMD23_ProductID	11003	11398		

In which we see that these four products in NMD3.0 are actually four variations of the same product in NMD2.3

By then running the query for the “mass according to table” that we showed earlier for each of the NMD3.0 Product_ID as shown, all necessary information for each of those products/variations can then be retrieved.