

Navigation Data Standard

Update Specification

NDS Version 2.1.1.9 and 2.1.1.10.1 - 2.1.1.10.6
(Preliminary Release)



Navigation Data Standard
PSF Physical Storage Format

Intellectual Property Rights

This document of a specification is released by the Navigation Data Standard (NDS) e.V., in the following called NDS e.V. It is released as a development partnership and intended for the purpose of information only. The NDS e.V. will not be liable for any use of this specification. Following the completion of the development of the Navigation Data Standard PSF specifications, commercial exploitation licenses will be made available to end users by way of written License Agreement only.

Navigation Data Standard PSF and the associated specification documents are subject to change and are continually updated as the development of Navigation Data Standard PSF progresses. The responsibility for maintaining and interpreting the Navigation Data Standard PSF specification documents lies with the bodies of the NDS e.V. Navigation Data Standard PSF development is driven by a well-defined process for releasing documented versions of the standard.

No part of this document shall be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from NDS e.V.

Specification documents for the Navigation Data Standard PSF may contain exemplary items (exemplary reference models, “use cases”, and/or references to exemplary technical solutions, devices, processes or software). Any such exemplary items are contained in the specification documents for illustration purposes only, and they themselves are not part of Navigation Data Standard PSF. Neither their presence in such specification documents, nor any later documentation of standard conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to Navigation Data Standard PSF.

Navigation Data Standard PSF is under constant development. For this reason, the description in this document may deviate from the physical implementation. If this is the case, the physical implementation in DataScript shall prevail.

Copyright © 2012 – Navigation Data Standard (NDS) e.V. All Rights Reserved.

Document version 1.6 (based on NDS versions 2.1.1.9 and 2.1.1.10.1 to 2.1.1.10.6)

Table of Contents

1	About this Document	7
1.1	Document History	7
1.2	Purpose and Subjects	9
1.3	Target Audience	9
1.4	NDS Documentation Overview	10
1.5	Use of Modal Verbs	10
1.6	Typographical Conventions	11
2	Introduction	13
2.1	Central Elements for Updates	13
2.2	Update Methods	14
2.3	Update Scenarios	14
2.3.1	Incremental Update of an Entire Product Database	15
2.3.2	Incremental Update of a Single Update Region (Regional Update)	16
2.3.3	Map Expansion	17
2.3.4	Adding Content to an Existing Product Database	18
2.3.5	Patch Update	18
2.3.6	POI Update	19
2.3.7	Attribute Update	19
2.4	Update Methods Used for Update Scenarios	20
2.5	Transmission Scenarios	21
3	Products and Update Regions	23
3.1	Product Database	23
3.2	Update Region	23
3.2.1	Partially Filled Update Regions	27
3.2.2	Ocean and Ferry Update Regions	30
3.2.3	World Overview Map	32
4	Database Elements for Updating	33
4.1	Versioned Items	33
4.2	Versioning Parameters	35
4.2.1	Technical Version Data	35
4.2.2	Business Version Data	37
5	Versioning Concept	39
5.1	Design Considerations for Versioning	39
5.2	Subversion and Version Control for NDS	40
5.3	NDS Version Tree	41
5.4	Version Control for Patches	45

5.5	NDS Versioning Concept – Example.	46
6	Gateways Between Update Regions	51
6.1	Route Link Gateways.	53
6.1.1	Changes at Route Link Gateways	55
6.2	Intersection Gateways	57
6.2.1	Changes at Intersection Gateways	59
6.3	Gateway Concept vs. Raw Data Specifications	60
6.4	Installation of Update Regions	60
A	Glossary	61

List of Figures

2-1	Changed tiles in level 13	16
2-2	Map expansion	17
3-1	Overview of products and update regions	24
3-2	Duplication of (partially filled) tiles at update region borders.	26
3-3	Duplication of (partially filled) tiles at update region borders for update region 1.	26
3-4	Duplication of (partially filled) tiles at update region borders for update region 2.	27
3-5	Part of region is filled	28
3-6	Entire update region is filled with reduced data	29
3-7	Ocean and ferry area	30
3-8	Ocean areas and gateways	31
3-9	World overview map	32
5-1	Considerations for selecting versioned items	40
5-2	NDS version tree	42
5-3	Dependencies for assigning version numbers to versioned items	44
5-4	Patch update versioning	45
5-5	Version tree for the NDS database (version V1)	46
5-6	Version tree for the incremental update (version V2)	47
5-7	Version tree for the incremental update (version V3)	48
5-8	Version tree for the patch update (version V4)	49
5-9	Version tree for the incremental update (version V5)	50
6-1	Gateways between update regions	52
6-2	Route link as gateway between two update regions.	53
6-3	Changes at a route link gateway	55
6-4	Intersection as gateway between two update regions	57
6-5	Changes at an intersection gateway	59

List of Tables

1-1	Rules for use of modal verbs	10
2-1	Impact on referencing types	19
2-2	Update methods and update scenarios	20
2-3	Overview of possible transmission scenarios	21
4-1	Technical version data	35

1 About this Document

This chapter gives an overview of the document's history, purpose, subjects and target audience, followed by abstracts of the subsequent chapters.

1.1 Document History

Version number	Published	Changes
1.0	2008-04-24	Initial version for NDS version 1.8.2
1.0.1	2009-07-10	Version for NDS 1.9. Only minor layout changes compared to version 1.0
1.1	2009-10-20	<p>General changes: Language revision, structural enhancements, reworked images, added references to other Specification documents and added DataScript locations (where applicable)</p> <p>Reworked introductions of all chapters and moved essential conceptual information to Chapter 2 Introduction on page 15</p> <p>Revision of Chapter 1 About this Document on page 9, added 1.6 Typographical Conventions on page 13</p> <p>Revision of Chapter 2 Introduction on page 15:</p> <ul style="list-style-type: none"> – Added section 2.1 Central Elements for Updates on page 15 to this chapter – Moved 2.3 Update Scenarios on page 16 to this chapter <p>Chapter 4 Database Elements for Updating on page 35:</p> <ul style="list-style-type: none"> – Changed list of versioned items – Collections are now described as non-atomic versioned items – Removed implicit version coding from document – Table 4-1 <i>Technical version data</i> on page 35: Replaced general terms for version data with DataScript names <p>Global gateway IDs introduced in Chapter 6 Gateways Between Update Regions on page 53</p> <p>New glossary terms added in Chapter A Glossary on page 63</p> <p>[Bug515] Version info for metadata: Sentence added at the end of chapter 4.1</p> <p>Section on Collections rewritten</p>
1.1	Dec 2009	Replaced term <i>collection</i> with <i>non-atomic versioned item</i>
1.2	2010-01-20	<p>[Bug636] Copyright section and subsequent changes due to transition to NDS e.V.</p> <p>Add info about use of modal verbs</p> <p>Figures 5-4 to 5-8 updated.</p>

Version number	Published	Changes
1.3 (for NDS 2.1.1 and NDS 2.1.1.1)	2010-06-03	<p>[Bug 575] Introduced new terms and clarified terms: update methods and update scenarios</p> <p>Added index at the end of the document</p> <p>[Bug 608 and others]: Added information about new versioning concept for relational tables and columns to be used, for example, for attribute and metadata updates.</p> <p>[Bug 851] Changed Section 6.4</p> <p>Added section on update methods, see 2.2 <i>Update Methods</i> on page 14</p> <p>Added section on attribute update, see 2.3.7 <i>Attribute Update</i> on page 19</p> <p>Updated topic on versioned items, see 4.1 <i>Versioned Items</i> on page 33</p> <p>Updated topic on NDS version tree, see 5.3 <i>NDS Version Tree</i> on page 41</p>
1.4 (for NDS 2.1.1.7)	2011-06-03	<p>Added subsection <i>Route Link Gateway vs. Intersection Gateway</i> in Chapter 6 <i>Gateways between Updat Regions</i></p>
1.5 (for NDS 2.1.1.8)	2011-08-25	<p>[Bug 2136] Changed filling level description in subsection <i>Partially Filled Update Regions</i></p> <p>[Bug 2138] Applied new format to SQL query and added space before WHERE statement in Section 6.4 <i>Installation of Update Regions</i></p> <p>[Bug 2137] Changed sentence in Section 6.3 <i>Gateway Concept vs. Raw Data Specifications</i> to describe issue more clearly</p> <p>[Bug 2022] Changed figure 2-1 <i>Changed tiles on level 13</i> and the explaining text to make it more clear</p>
1.6 (for NDS 2.1.1.9 and 2.1.1.10.x)		<p>[Bug 2136] Changed description of rules for definition of update regions to make it more clear</p> <p>[Bugs 2151 and 2152] Added tile content indices and POI virtual tiles to list of atomic / non-atomic items</p>

1.2 Purpose and Subjects

This document contains the update concepts specified for Navigation Data Standard (NDS), a standardized physical storage format for navigation systems. The format has been developed by NDS e.V., a registered society of car manufacturers, navigation system suppliers, and map suppliers.

The standardized binary format for navigation data as specified in the *NDS – Format Specification* also offers new possibilities for updating navigation databases. This document, the *NDS – Update Specification*, describes a harmonized and flexible concept for content updates for NDS databases, including incremental and partial updates. This concept also allows using update processes for enhancing the content of navigation databases, for example, by adding countries or building blocks without changing the basic database structure. The NDS update concept ensures that the navigation database is still consistent after an update.

This document deals with the following subjects:

- Chapter 2 *Introduction* on page 13. This chapter introduces the update topic, explains basic principles of updating NDS databases, and gives an overview of update scenarios, update methods, and transmission scenarios.
- Chapter 3 *Products and Update Regions* on page 23. This chapter introduces the concept of product databases and update regions as central high-level elements for updating.
- Chapter 4 *Database Elements for Updating* on page 33. This chapter introduces the database elements that may be updated.
- Chapter 5 *Versioning Concept* on page 39. This chapter introduces the versioning concept for NDS. It explains the implemented version control mechanisms, the concept of NDS version trees, and the version control for patches.
- Chapter 6 *Gateways Between Update Regions* on page 51. This chapter describes how update regions can be compiled and updated independently while maintaining connectivity.
- Chapter A *Glossary* on page 61. This chapter defines the main terms of the NDS update concept.

1.3 Target Audience

This document is provided for IT professionals who want to get familiar with the update processes of Navigation Data Standard. This can be, for example, software development managers, product managers specifying navigation systems, or engineers developing such systems.

The following knowledge is assumed:

- Familiarity with digital maps and navigation systems
- Solid know-how regarding the structure and functionality of databases

1.4 NDS Documentation Overview

The following documentation on Navigation Data Standard (NDS) is available:

- *NDS – Format Specification*: This document contains detailed descriptions of the data types of NDS databases, meaning features and their attributes, as well as metadata.
- *NDS – Compiler Interoperability Specification*: This document contains interoperability guidelines for compiling navigation databases based on NDS.
- *NDS – Update Specification*: This document describes the general concepts and processes for updating NDS databases.
- *NDS – Certification Specification*: This document describes the process and requirements for certification of navigation databases complying with the Navigation Data Standard (NDS) physical storage format.
- *NDS – Physical Model Description*: This documentation comprises the NDS DataScript implementation. It also contains the comments which describe the structure and properties of the NDS data types.
- *Documentation for NDS Validation Suite*: This document describes how to use the NDS Validation Suite and describes its map API.

1.5 Use of Modal Verbs

For compliance with the NDS standard, database suppliers need to be able to distinguish between mandatory requirements, recommendations, permissions, as well as possibilities and capabilities. This is supported by the rules for use of modal verbs that are followed in the NDS specification documents to express the four kinds of provisions.

Table 1-1 Rules for use of modal verbs

Provision	Verbal form	Allowed alternative expressions
Requirement Requirements shall be followed strictly in order to conform to the standard. Deviations are not allowed.	shall	is to, is required to, it is required that, has to, only ... is permitted, it is necessary, must
	shall not	is not allowed [permitted, acceptable], is required to be not, is required that ... be not, is not to be, must not
Recommendation Recommendations indicate that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others.	should	it is recommended that, ought to

Provision	Verbal form	Allowed alternative expressions
	should not	it is not recommended that, ought not to
Permission Permissions indicate a course of action permissible within the limits of NDS deliverables.	may	is permitted, is allowed
	need not	it is not required that, no ... is required
Possibility and capability These verbal forms are used for stating possibilities or capabilities, being technical, material, physical, or causal.	can	be able to, there is a possibility of, it is possible to
	cannot	be unable to, there is no possibility of, it is not possible to

1.6 Typographical Conventions

This documentation uses the following typographical conventions:

- **Code elements:** This format is used for code elements, such as technical names of classes and attributes, as well as attribute values.
- **File and path names:** This format is used for the names of files and folders, as well as for directory paths.
- **Cross-references:** This format is used for cross-references to topics in the current document, to topics in other specifications that are part of Navigation Data Standard, or to external documents.
- **Terms:** This format is used to introduce glossary terms, new terms and emphasize particular terms.
- **<Variables>:** For placeholders that need to be replaced by actual values, angle brackets are used. Example: The variable *<installation directory>* needs to be replaced by, for example, *C:\Program Files*.

The following types of note are used:

Information Notes of type “Information” offer background knowledge, show alternative ways of performing a task or provide information to make your work more efficient.

Note	Notes of type “Note” provide important information that helps you avoid errors and problems.
-------------	--

Caution	Notes of type “Caution” contain essential information that you must follow to avoid malfunctions, incorrect data and other major problems.
----------------	--

Example	Notes of type “Example” contain examples for the concept described in the current topic.
----------------	--

2 Introduction

The guiding principle that needs to be adhered to when developing update concepts for databases is *consistency*: After the update, the entire navigation database on the system must still be consistent and must not return incorrect results.

For updating NDS databases, the main requirement, therefore, is ensuring the consistency of the basic navigation, mainly the consistency of the data of the mandatory building blocks Basic Map Display, Routing (including traffic), and Names. The NDS database supplier shall ensure that all references are correct:

- References within a building block, between building blocks, and between different levels
- References of data stored in tiles and data not stored in tiles, for example, names in the Name building block
- References to optional building blocks, such as POI and Digital Terrain Model

This chapter introduces the main elements of the update concept and describes update methods and scenarios, as well as transmission scenarios for update data that motivated the design of the update concept.

2.1 Central Elements for Updates

NDS defines special elements to enable database updates and to ensure database consistency.

Product Database and Update Region

On the product level, the elements *product database* and *update region* are defined:

- **Product database:** Each product database belongs to one database supplier, has its own version control, and may, therefore, be updated independently from other product databases. Product databases may contain one or more building blocks. They cover a geographic area which may be further divided into several update regions.

For more information about the product database concept, refer to Section 3.1 *Product Database* on page 23.

- **Update region:** An update region represents a geographic area in a database that may be subject to an update, for example, Germany or the Benelux countries. Any database that complies with Navigation Data Standard may be logically divided into update regions. Two or more adjacent update regions of a product database overlap at their borders, their interior is disjoint.

For more information about the update region concept, refer to Section 3.2 *Update Region* on page 23.

Versioned Items

Within product databases and update regions, different items may be updated, such as complete building blocks or individual tiles. For this reason, so-called *versioned items* are defined. A versioned item shall always be updated as a whole in one update transaction. This ensures consistency and structural integrity of an updated database. A versioned item may be a logical NDS element, such as country and region, or a physical NDS element, such as building block and BLOB. For more information on versioned items, refer to Section 4.1 *Versioned Items* on page 33.

2.2 Update Methods

NDS differentiates between *update methods* and *update scenarios*. Update methods describe how updates are executed: by exchanging the complete database, by exchanging relational tables, or by replacing a subset of records. Update scenarios, on the other hand, describe what needs to be updated or added to a database, for example, a country that needs to be added to a product database. The update scenarios may be realized by one of the available update methods.

NDS distinguishes the following update methods:

- Full update: In a full update, all data records of an NDS product database are exchanged by replacing the database's SQLite files with new files containing the updated content.
- Partial update: In a partial update, a subset of a product database's records is exchanged. The exchanged records form a logical unit, meaning an update region or a building block. The corresponding SQLite files are replaced with new files containing the updated content.
- Incremental update: In an incremental update, a subset of a product database's records is exchanged by executing SQL transactions on relational tables.
- Patch update: A patch update is a type of incremental update that is limited to a very small number of data records.

2.3 Update Scenarios

NDS defines a number of update scenarios. NDS-compliant systems shall support at least these scenarios:

- Incremental update of an entire product database
- Incremental update of a single update region
- Map expansion
- Adding content to an existing product database
- Patch update
- POI update
- Attribute update

Other update scenarios are feasible and may be supported additionally. This chapter explains the general scenarios and describes the interaction required between user and system, as well as the functionalities of the update mechanisms.

Note In NDS, elements are referenced by a positional index in the corresponding list. During an update, the order of the list entries can be changed and, as a consequence, the position IDs would change. All references to features with changed position IDs would have to be updated accordingly. Therefore, an apparently small change can result in major database changes, a so-called *avalanche effect*. As a consequence, inconsistencies in the database can occur.

To avoid the avalanche effect, each feature has a permanent identity in addition to the position ID. The permanent identity remains stable after an update whereas the position ID may change. A range table maps the permanent identities to the position index of the corresponding table. By using range tables, the impact of changed position IDs due to an update is kept to a minimum.

For more information on range tables, refer to *NDS – Compiler Interoperability Specification*, 3.6 *Range Tables* on page 63.

It is in the NDS database supplier's responsibility to consider possible avalanche effects and ensure that the database is free of avalanche effects and consistent after an update.

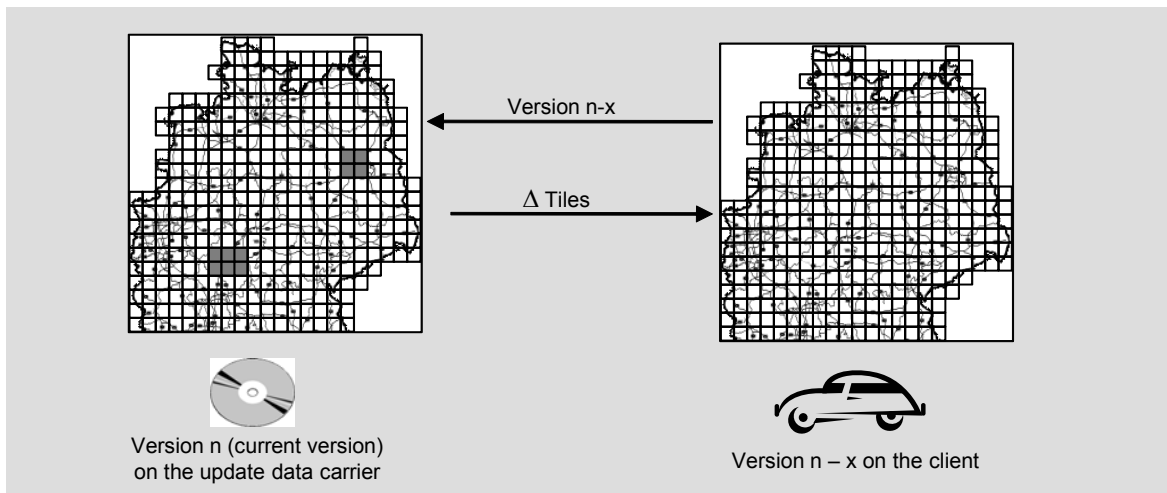
2.3.1 Incremental Update of an Entire Product Database

In this update scenario, the entire product database of a system is checked for outdated content. During the update process, all tiles, BLOBs, and other data structures of all building blocks and all levels that have changed between the version on the client and the current version are replaced with the current versions.

The result is a consistent, up-to-date version of the product database on the system.

Figure 2-1 shows the outdated version of a product database on the client on the right hand side (version n-x on the client). The left hand side shows an update medium with the current version of the product database (version n). The update processes implemented in navigation systems are started once update data is received. During the update process, the system on the left compares the version trees and replaces the modified versioned items in the client, for which a newer version is available.

Figure 2-1 Changed tiles in level 13



2.3.2 Incremental Update of a Single Update Region (Regional Update)

In this update scenario, only one update region within a product database is updated. This update scenario is also referred to as *regional update*.

Example A customer has the product database for central Europe and wants to update Germany.

During the update process, all tiles, BLOBs, and other data structures of all building blocks and all levels of one update region that have changed between the version on the client and the current version are replaced with the current tiles, BLOBs, and other data structures.

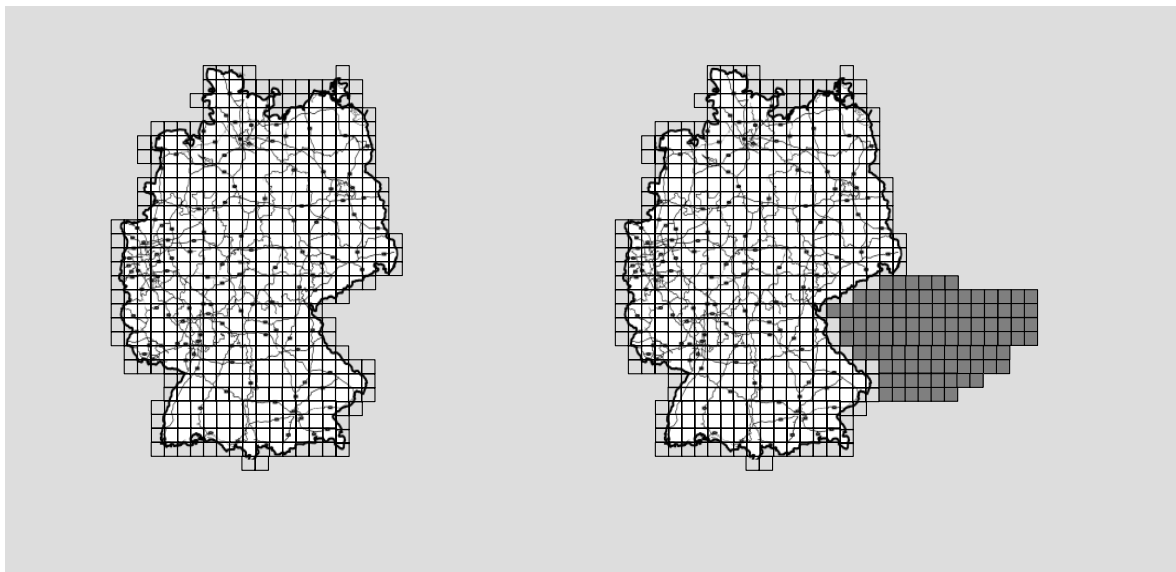
The result is a completely consistent, up-to-date version of the update region on the client.

For more information about update regions, refer to Section 3.2 *Update Region* on page 23.

2.3.3 Map Expansion

In this update scenario, new update regions are added to an existing product database. The map expansion scenario is not an update in literal sense. It is, however, a very important scenario for increasing the database coverage in terms of adding a new geographic region, and is, therefore, explained here. For this scenario, the entire content of at least the mandatory building blocks (Basic Map Display, Routing, and Name) of a new update region is added to the existing product database (see Figure 2-2).

Figure 2-2 Map expansion



It is not possible to add geographic regions that are smaller than an entire update region. The added update region does **not** need to be adjacent to the existing update region in the product database.

Example A German customer owns a summer cottage in Spain and uses a mobile client with a product database for Germany. For driving in Spain during the holidays, the customer wants to add Spain to the product database.

To ensure connectivity at borders of adjacent update regions, proper gateways need to be present.

Information For more information on the gateway concept, refer to Chapter 6 *Gateways Between Update Regions* on page 51.

2.3.4 Adding Content to an Existing Product Database

Navigation Data Standard defines mandatory and optional building blocks. For basic navigation, the following building blocks are mandatory:

- Routing building block
- Name building block

If the Basic Map Display building block is present in the database, it must be updated together with the Routing and Name building blocks. In this update scenario, optional building blocks, such as the Digital Terrain Model building block, are added to the product database. This update scenario excludes adding data that is closely linked to data in a mandatory building block, for example speed limits to the Routing building block.

For more information on the building blocks, refer to the *NDS – Format Specification*.

The following building blocks may be added:

- Speech
- POI
- Digital Terrain Model
- Orthoimages
- 3D
- Full-text Search

The content may be added per update region. Some building blocks require that data is only added for the entire product database.

Provided that there are no references between the mandatory building blocks and the optional content, the product database version of the client and the version of the content update may be different. To avoid mismatches (for example, when displaying the road network on top of orthoimages), it is the NDS database supplier's responsibility to ensure that the additional content fits to the existing product database.

2.3.5 Patch Update

In this update scenario, individual errors or problems of one or more features in a product database can be fixed immediately without waiting for the next regular incremental or full update. A patch update can be required, for example, if a very important transport connection is unusable for a longer time, or if a serious error in the product database needs to be fixed.

Patch updates are mainly applied to the building blocks for basic navigation, for example, Basic Map Display, Routing, Name, and integrated POI. They shall be restricted to significant malfunctions, for example, a major detour in routing. Patch updates shall not be used for minor problems, for example, spelling errors in names. Minor problems may be solved with the next regular incremental or full update. The database supplier decides if a full or an incremental update is provided.

2.3.6 POI Update

In this update scenario, only POI data is updated. Due to a higher change rate, POIs need to be updated more frequently than basic navigation data. Also, customers would usually expect a higher flexibility with regard to POIs.

To allow a more flexible update, the POI building block needs to be physically separated from the basic navigation building blocks. It is not mandatory to update the POI building block in tiles, as it is very unlikely to get updates for private POIs and third-party POIs in the same tiles as basic navigation data.

POIs may be referenced by a road network by means of

- Direct references
- Geographic coordinates
- AGORA location referencing

Table 2-1 shows the impact of update complexity, memory requirements, and matching quality for the three referencing types. They have to be taken into consideration for the update specification.

Table 2-1 Impact on referencing types

	Memory requirement	Complexity	Matching quality
Coordinates	medium	low	medium
Direct references	low	high	high (only feasible with knowledge of the road network)
AGORA	high	medium	high

Only AGORA and coordinates allow for independent update of POI and routing data. If direct references are used to connect POIs to the road network, the POI and Routing building blocks must to be updated together.

For more information, refer to *NDS – Format Specification*, 13 *POI Building Block* on page 253.

2.3.7 Attribute Update

NDS supports updates only for attribute layers that are factored out from tile BLOBs to a dedicated column of a relational table. For updating these attribute layers, the versioning mechanism for relational tables or columns in relational tables may be used, see *NDS – Format Specification*, 4.5.5 *Building Block Component Version Table* on page 68.

Currently, the following attribute layers of the Routing building block may be updated:

- ADAS attributes
- Truck attributes
- Name attributes

Each attribute BLOB has an own version number, which makes it possible to insert, update, and replace these attribute blobs without changing the corresponding routing features.

Caution Routing attributes stored in these layers are closely linked with the corresponding routing features. When compiling attribute updates, special care shall be taken in order to preserve database consistency during update.

2.4 Update Methods Used for Update Scenarios

Table 2-2 illustrates which update methods may be used to realize different update scenarios.

Table 2-2 Update methods and update scenarios

Update scenarios	Update methods		
	Full update	Partial update	Incremental/ patch update
Incremental update of an entire product database	x	x	x
Incremental update of a single update region		x	x
Map expansion		x	
Adding content to an existing database		x	
Patch update			x
POI update		x	x
Attribute update			x

2.5 Transmission Scenarios

There are several possible scenarios for transmitting update data to navigation systems. The general procedure for transmitting data, however, is similar for all transmission scenarios:

1. Determining the required update data

To determine the data that needs to be updated, some basic information about the navigation system and the customer's license is needed, such as the navigation database version, as well as the type and the version of the application. This information is made available via so-called *versioning parameters* (see Section 4.2 *Versioning Parameters* on page 35).

2. Collection of the update data files



Based on the information gathered in step 1, the update data files are collected and prepared for download.



3. Importing update data files into the navigation system

Update files can be imported, for example, by means of an USB storage device or via a wireless connection.

Table 2-3 summarizes the possible transmission scenarios.

Table 2-3 Overview of possible transmission scenarios

Transmission scenario	Advantage	Disadvantage
Update at the dealer's garage Update data is imported at the dealer's garage during routine visits, such as car service. 	Can be offered as a service at an attractive price at the dealer's garage.	Not suitable for navigation systems with data that needs to be updated at shorter intervals
Indirect update via Internet Customer downloads update data from an Internet server and imports the update in the navigation system by means of an USB storage device or flash card. 	Customer can download updates at any time. Additional features with costs can be purchased in addition to the update.	A hotline service needs to be installed to offer support for all problems the customer might encounter during the update process.

Transmission scenario	Advantage	Disadvantage
<p>Direct update via Internet</p> <p>Customer downloads update data from an Internet server via a WLAN connection between the navigation system and the update server.</p> 	<p>Same advantages as the indirect update via internet.</p> <p>An additional advantage is that no separate data carrier is required.</p>	<p>A hotline service needs to be installed to offer support for all problems the customer might encounter during the update process.</p>
<p>Automatic update via UMTS or GPRS</p> <p>A wireless connection between navigation system and update server enables automatic updates according to the customer's license.</p> 	<p>Most comfortable update scenario especially for navigation data with a high update frequency.</p>	<p>Limited bandwidth of wireless connections</p> <p>Costs for wireless data transmission might be high.</p>

3 Products and Update Regions

An NDS database may consist of several *product databases*, and each product database may be divided further into *update regions*. This concept makes it possible to integrate databases from different database suppliers into one NDS database and supports a flexible and consistent versioning concept for NDS databases.

This chapter introduces the concept of product databases and update regions for NDS databases.

3.1 Product Database

Each product database belongs to one database supplier, has its own version control and may therefore be updated independently from other product databases. Product databases may contain one or more building blocks and cover a geographic area. The geographic area may be further divided into several update regions. An NDS database may contain several product databases in parallel.

Each NDS database supplier uses its own version control, which is not necessarily synchronized with the version control of another NDS database supplier. For more information on the version concept, refer to Chapter 5 *Versioning Concept* on page 39.

Example An NDS database comprises the following two product databases:

- Europe basic navigation (produced by Harman Int.)
 - Europe POIs (produced by Michelin)
-

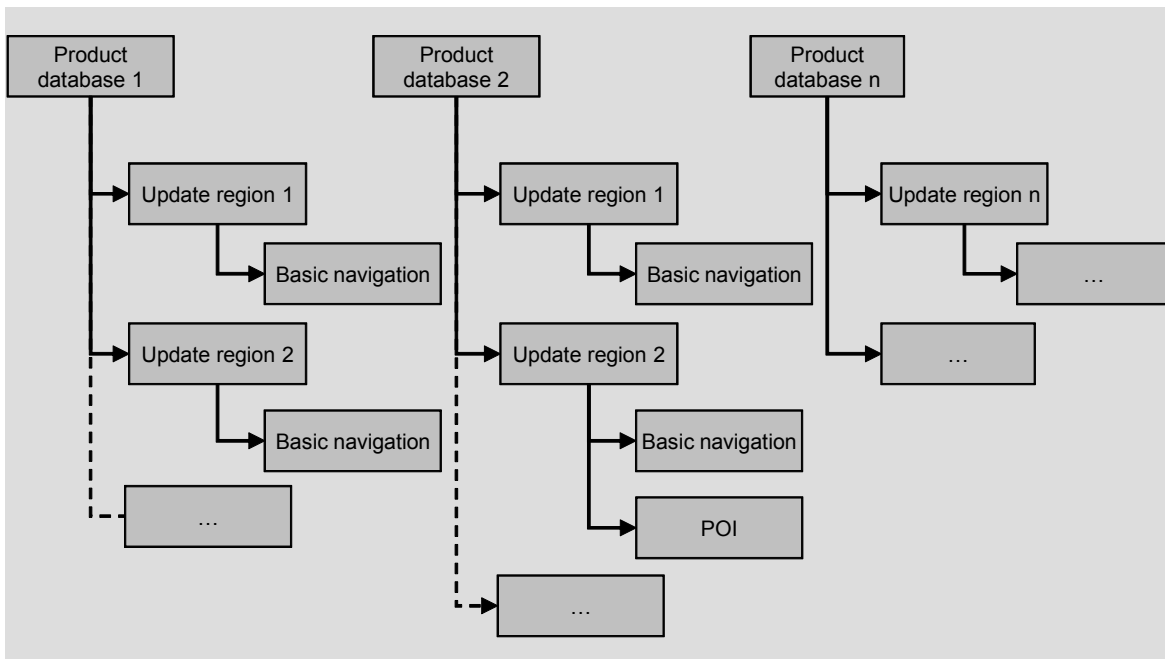
For detailed information about the definition of product databases in the physical database structure, refer to *NDS – Format Specification, 4 Database Structure* on page 59.

3.2 Update Region

The geographic area of a product database is partitioned into one or more update regions. An update region is a geographic area of a product database that may be subject to a regional update (see Section 2.3.2 *Incremental Update of a Single Update Region (Regional Update)* on page 16). Two or more adjacent update regions of a product database overlap at their borders, their interior is disjoint. Update regions may contain holes and disconnected areas, for example, enclaves and exclaves.

Figure 3-1 illustrates the concept of product databases and update regions.

Figure 3-1 Overview of products and update regions



Geographical Extent of Update Regions

Navigation Data Standard does not define the concrete geographical extent of update regions. It is left to the requirements of the map database products defined by map suppliers, system suppliers, or car manufacturers.

Example Update regions based on countries/country groups for a Europe product database:

- Update region 1: Germany
- Update region 2: Austria
- Update region 3: Switzerland
- Update region 4: Benelux
- Update region 5: Scandinavia

Update regions based on state groups for a US product database:

- Update region 1: West Coast
- Update region 2: East Coast
- Update region 3: Middle West

Another NDS database supplier may choose a different update region schema. Even the same supplier may change its schema over time.

Technical Implementation of Update Regions

An update region is technically defined by a set of data (tiles, data in tiles, and data not contained in tiles) from all building blocks which are contained in the update region.

Update regions in a product database shall always contain data. Empty update regions are not physically stored (in terms of creating empty directories, database files, or tables). If update regions are added during the update, the database files and tables needed for the new update region are created on the fly (for an example of the map expansion update scenario, see Section 2.3.3 *Map Expansion* on page 17).

For detailed information about the definition of update regions in the physical database structure, refer to *NDS – Format Specification, 4 Database Structure* on page 59.

To uniquely identify the version of update region data and, therefore, to decide whether the data needs to be updated, a `versionId` is assigned to the update region. To define partly filled update regions, the `isPartiallyFilled` attribute is assigned to an update region (DataScript location: `UrBuildingBlockVersionTable` table in `nds.overall.productdbversion`). For more information, refer to Section 3.2.1 *Partially Filled Update Regions* on page 27.

Example	Product database A has version ID 42; the attribute <code>isPartiallyFilled</code> is set to 0. The version ID identifies the map data of Q4/2008 and the value of the <code>isPartiallyFilled</code> attribute indicates that the database contains complete data. A value <code>n</code> for the <code>isPartiallyFilled</code> attribute indicates that the update region contains only the main roads or only a part of the update region.
----------------	--

Updating an Update Region

An update region may be updated by means of the following update methods:

- Incremental update, if the update affects only a small amount of data: Replace the changed tiles and update the data that depends on the replaced tiles.
- Full update, if the update affects a large amount of data: Replace the entire update region, for example, by copying all relevant SQLite files, and, if applicable, by updating the data that depends on it.

Tiles at Update Region Borders

Two or more adjacent update regions only overlap at borders. Due to the regular tiling schema, it is likely that update region borders do not coincide with tile borders. A logical tile at overlapping borders is therefore physically stored in each intersecting update region. The respective tile is only filled with the content belonging to the corresponding update region.

Figure 3-2 shows the logical tile 0201 that is cut by an update region border. Figure 3-3 shows the partially filled tile 0201 for update region 1. Figure 3-4 shows the partially filled tile 0201 for update region 2.

Figure 3-2 Duplication of (partially filled) tiles at update region borders

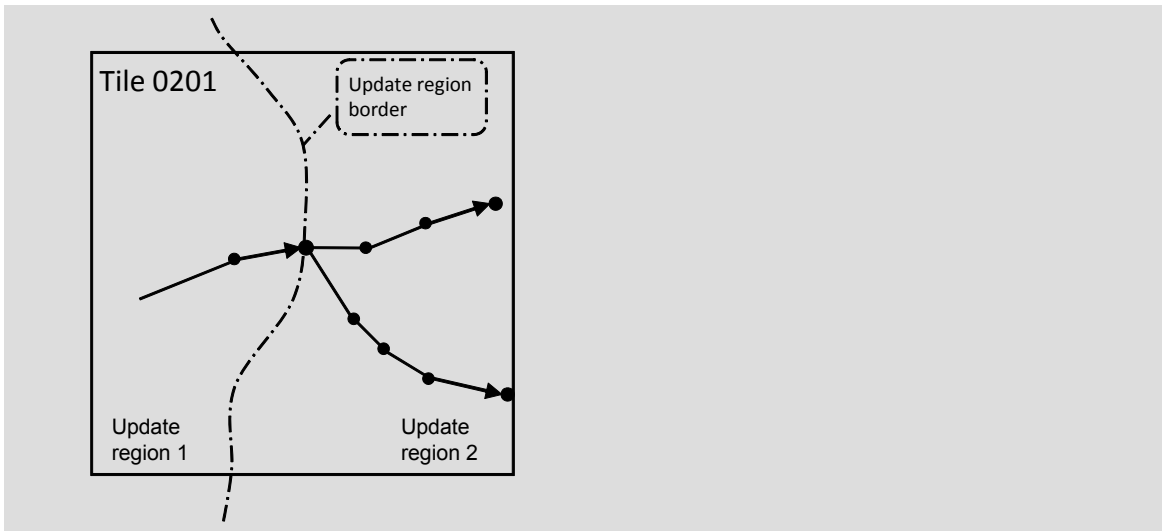


Figure 3-3 Duplication of (partially filled) tiles at update region borders for update region 1

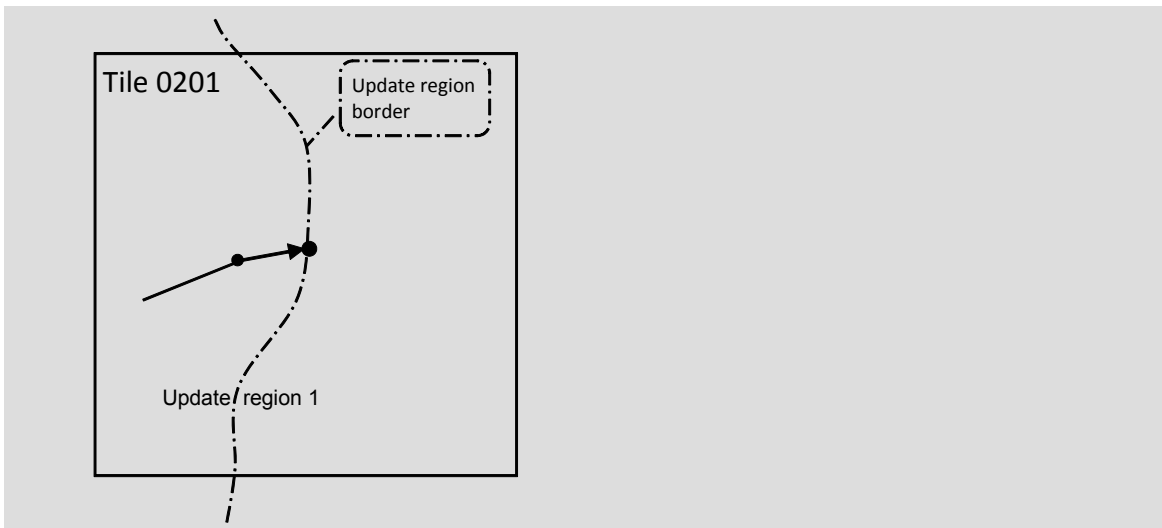
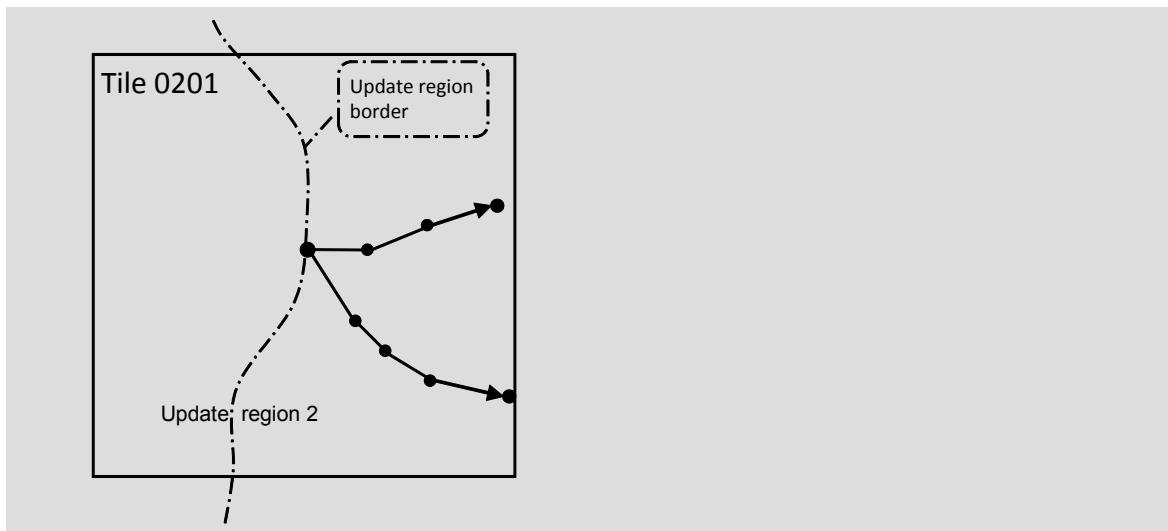


Figure 3-4 Duplication of (partially filled) tiles at update region borders for update region 2



Special Types of Update Regions

NDS defines some special types of update regions:

- Update regions that are only partly filled, see 3.2.1 *Partially Filled Update Regions* on page 27
- Update regions for oceans and ferry lines, see 3.2.2 *Ocean and Ferry Update Regions* on page 30
- Update region for a world overview map, see 3.2.3 *World Overview Map* on page 32

3.2.1 Partially Filled Update Regions

Partially filled update regions do not contain all data on all levels, but are only filled to a defined grade. Partially filled update regions may, for example, contain only main roads or only data for specific areas. NDS database suppliers may individually define filling levels in the `isPartiallyFilled` field of the `UrBuildingBlockVersionTable`.

The NDS database supplier shall ensure that the following conditions are met:

- Partially filled update regions shall be consistent; dangling pointers are not allowed.
- The following rules must be applied for the definition of update regions:
 - Filling level = 0 means that the update region is completely filled
 - Filling level $x+1$ is less filled than filling level x
 - Filling level x can overwrite any filling level that is less filled than x

Ensuring Consistency when Updating Partly Filled Update Regions

To ensure consistency when updating partially filled update regions, the following guidelines shall be followed:

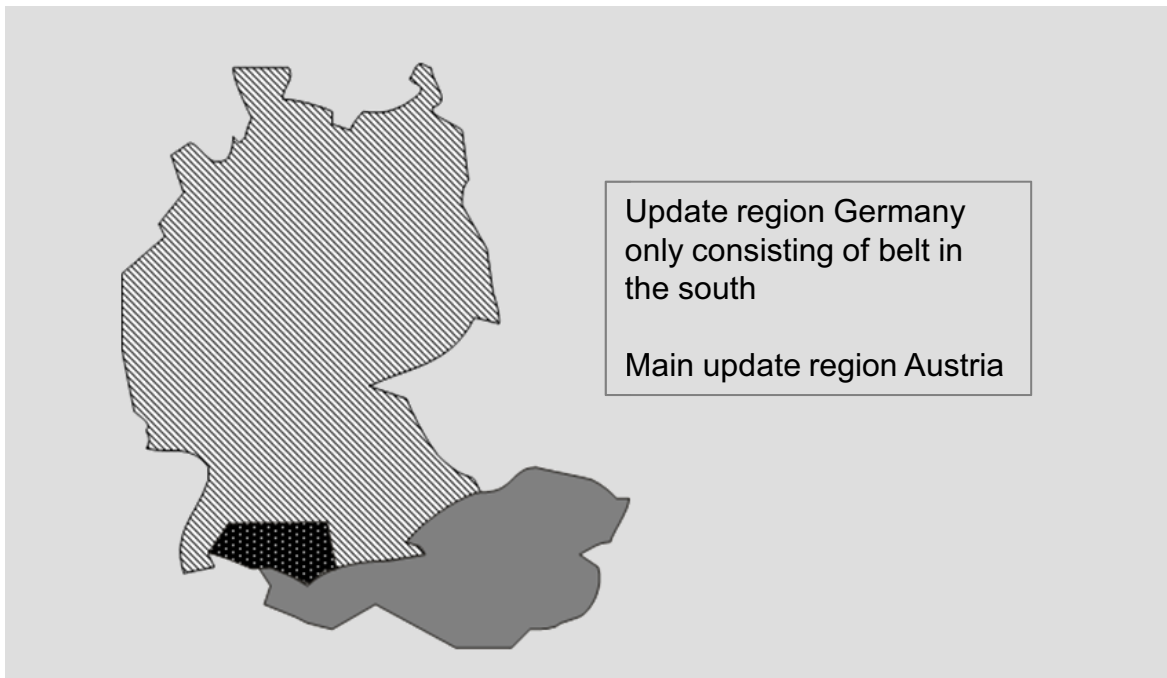
- Update regions with a numerically lower filling level are updated before update regions with a numerically higher filling level.
- When updating a partially filled update region, consistency with all adjacent update regions shall be ensured. All combinations shall be possible.
- Existing partially filled update regions may only be updated with:
 - Partially filled update regions with a numerically identical filling level
 - Partially filled update regions with a numerically lower filling level
 - Partially filled update regions containing the data of the current partly filled update region and other data

Example 1: Part of an update region is filled

The filled part would usually be an area adjacent to another update region. For example, Austria and the “Kleinwalsertal”: Austria is the main update region. The Kleinwalsertal is located in the north-west of Austria. It can only be reached by driving through a part of Southern Germany.

To ensure that route calculation to the Kleinwalsertal is possible, the NDS database supplier would provide the complete update region Austria (filling level = 0) and the partly filled update region Germany. Only the area of Germany that is needed for route guidance to the Kleinwalsertal is filled with data; all other areas of the update region Germany will not be filled with data.

Figure 3-5 Part of region is filled

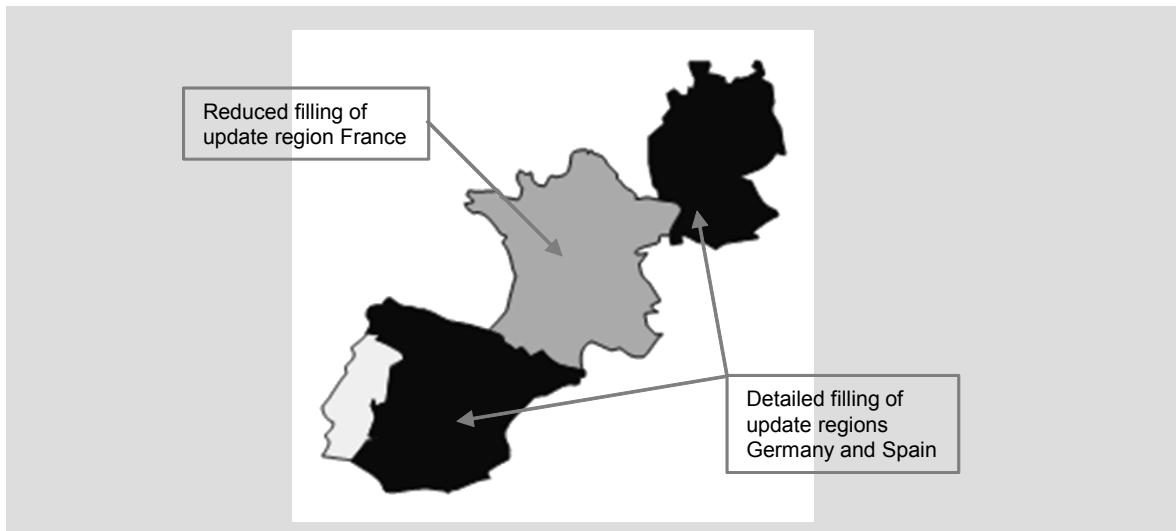


Example 2: Entire update region is filled with reduced data

In this example, the entire update region is filled, but with reduced data. A use case would be a customer who lives in Germany and has a summer cottage in Spain. To navigate from Germany to Spain, the customer also needs information on the main roads for France.

To cover this use case, the NDS database supplier would provide the completely filled update regions Germany and Spain, and a partially filled update region France which only contains the major road network.

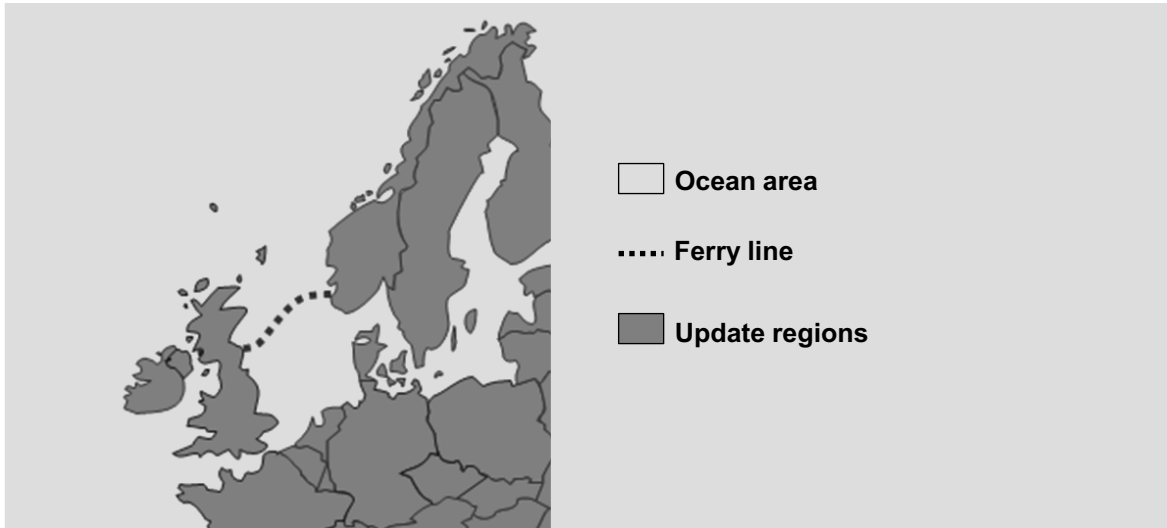
Figure 3-6 Entire update region is filled with reduced data



3.2.2 Ocean and Ferry Update Regions

For proper map rendering and long distance routing, the international ocean areas and ferry lines must be included in product databases. The NDS update concept describes the general design of update regions, but does not specify in detail how they shall be filled. This also applies to international areas. However, the NDS update concept recommends storing international ocean areas and ferry lines in one or more dedicated update regions.

Figure 3-7 Ocean and ferry area



The following definitions and restrictions apply for ocean and ferry areas:

- Ocean and ferry update regions underly the same restrictions as ordinary update regions
- Gaps between ocean and ferry update regions and other update regions are not allowed
- Ocean and ferry update regions must include the mandatory building blocks:
 - Basic Map Display for ocean areas and the rendering of the ferry line
 - Names for ocean area and ferry lines
 - Routing for ferry lines and for tunnels and bridges between update regions

The following recommendations are given:

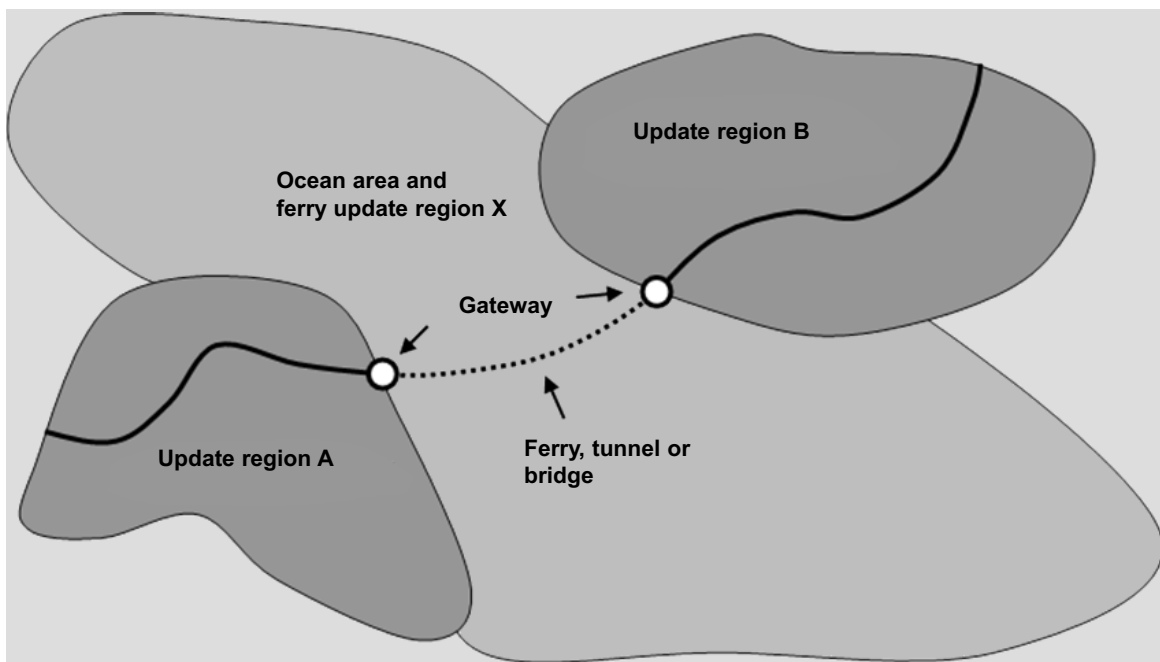
- Create separate ocean and ferry update regions for each product database
- International waters should be modelled by means of ocean and ferry update regions. The coastal waters of countries including the three-mile zones as provided by map suppliers, should, on the other hand, be modelled by means of a narrow fringe around the update region that the country belongs to.

Gateway Concept for Ferries, Tunnels and Bridges

The gateway concept specifies how update regions are compiled and updated independently while maintaining connectivity between the update regions (see Chapter 6 *Gateways Between Update Regions* on page 51). The gateway concept also applies to connecting ferries, tunnels, and bridges by integrating them into the topological network.

In an ocean and ferry update region, ferry lines, tunnels, and bridges are treated the same way as roads in a “normal” update region. Therefore two gateways are necessary for each ferry, tunnel, and bridge: One gateway from the update region A to the ocean and ferry update region X, and another gateway from the ocean and ferry update region X to the update region B.

Figure 3-8 Ocean areas and gateways



Example Examples for ferry, tunnel, and bridge gateways:

- Ferry line from Kiel, Germany, to Oslo, Norway
- Eurotunnel from France to Great Britain
- Öresund bridge from Denmark to Sweden

3.2.3 World Overview Map

The World overview map is a special update region, and is delivered as an own product database. It is used for showing the whole world or large sections of the world map.

A product database contains a flag indicating whether or not it represents a world overview map. The metadata of the corresponding building blocks contain the information about the levels filled in the product database.

Figure 3-9 World overview map



The following definitions and restrictions apply for the World overview map:

- It is an optional product database which may be updated independently from other product databases and contains only one update region.
- It contains no references to detailed update regions of other product databases in the same NDS database; coordinates must be used for referencing to other product databases.
- It is intended for map rendering including names and selected POIs.
- It provides no routing functionality.
- It is only available above the highest routing level.

4 Database Elements for Updating

Within product databases and update regions, different items may be updated, such as complete building blocks or individual tiles. This section explains which elements of an NDS database are versioned and which parameters are used to store version information for an NDS database.

4.1 Versioned Items

A central element of the NDS update concept are so-called *versioned items*. A versioned item may be:

- A logical NDS element such as country, region, and POI
- A physical NDS element such as building block, table, tile, BLOB, and record

A versioned item must always be updated as a whole in one update transaction. This ensures consistency and structural integrity of an updated database. As versioned items can depend on other versioned items, updating one versioned item can require the update of a dependent versioned item.

Each versioned item has a version identifier, which is stored in a separate data structure. Storing the versioning information separately helps avoid compilation errors, as version information is usually generated in a post process of the NDS compilation and re-editing structures is error-prone. Furthermore, separate structures allow fast compilation of a complete version state of the database and also faster execution of an update package on an NDS database.

As the scenarios defined in the NDS update concept do not focus on updating individual versioned items, but on updating groups of versioned items, NDS differs between *atomic* and *non-atomic* versioned items.

Atomic versioned items represent individual physical objects in the database that are subject to changes and thus are versioned. The following elements are examples for atomic versioned items in NDS:

- **Tiles:** BmdTile, BmdTilePattern, RoutingTile, RoutingGeoTile, OrthoImageTile, DtmSurfaceTile, HeightMapData, AttributeMapList (in TMC tile table)
- **BLOBs:** NamedObjectPhoneticTranscriptionListBlob, NamedObject, NvcSubTreeBlob
- **Records:** POI object, junction view object
- **Metadata:** tile content indices (see also *Update of Metadata* on page 35)
- Icons

Non-atomic versioned items form groups of atomic versioned items and carry their own explicit version identifier, which is derived from the version identifiers of the underlying atomic versioned items (see Section 5.2 *Subversion and Version Control for NDS* on page 40). They enable a more efficient execution of updates on a database by comparing the version identifiers of groups of objects rather than the version identifiers of individual objects. The following elements are examples for non-atomic versioned items:

- Product database
- Update region
- Building block
- (Relational) table
- Columns of a relational table
- Icon set
- Icon collection
- POI virtual tile
- Junction view tile

Update of Shared Tables

NDS allows storing individual tables of a building block in different SQLite files. This facilitates versioning and updating of individual tables, as well as sharing individual tables between different building blocks. An example for a table shared between two building blocks is the `ColorTable`, which may be shared between BMD and 3D building block. Other examples are the `GlobalGatewayTable`, `IconTable`, and the `FontTable`, which may be shared between building blocks in different update regions.

The SQLite file in which a table is stored is specified in the `BBlockCompVersionTable`, see *NDS – Compiler Interoperability Specification, 2.1 Distribution of Tables over SQLite Files* on page 39.

Note

Sharing an individual table between several building blocks creates additional update dependencies. Data in such tables is used by several building blocks. When a building block using data stored in a shared table is updated, special care must be taken to ensure database consistency, as the changes may affect other building blocks that also use this table, but would normally not be affected by the update. Based on the NDS version tree concept, the version numbers of all building blocks sharing this table need to be adapted when the shared table is subject to an update.

For detailed information the version tree concept, refer to 5.3 *NDS Version Tree* on page 41.

Update of Metadata

Metadata that is stored in tables or columns of tables within building blocks may be updated, for example, the metadata in `PoiCategoryTable` and `PoiMetadataTable`. Metadata updates use the versioning mechanism for relational tables or columns in relational tables, see *NDS – Format Specification*, 4.5.5 *Building Block Component Version Table* on page 68.

Metadata stored inside a BLOB always carries the same version ID as the BLOB itself.

The following restrictions apply to metadata versioning:

- Metadata for an update region always has the same version as the update region itself.
- Metadata for a product database always has the same version as the product database itself.

4.2 Versioning Parameters

NDS uses versioning parameters to store version information for versioned items. The following types of parameter are used:

- Technical version data
Contains version information about data, formats, and releases that is important to identify the correct data for an update process. Includes, for example, data version, NDS database version, and compiler information.
- Business version data
Contains business-related production information, such as product identification.

4.2.1 Technical Version Data

The technical version data contains the information that is necessary to identify particular data unambiguously. This ensures that the update process knows whether an update package can be executed on a database or not.

Table 4-1 shows the information contained in the technical version data.

Table 4-1 Technical version data

Version data	Description
NdsDbSupplierId	IDs for suppliers of digital road maps, such as NAVTEQ, Tele Atlas, and Zenrin, but also for providers of other content, such as POIs, or orthoimages The identifier is to be defined and maintained by the NDS consortium, see <i>NDS – Format Specification</i> , B <i>NDS Database Supplier IDs</i> on page 369 DataScript location: <code>nds.overall.version > NdsDatabaseSupplierTable</code>

Version data	Description
versionId	<p>Uniquely identifies the status of a portion of data</p> <p>The version ID can be used, for example, to find out which data is the most recent one.</p> <p>DataScript location: <code>nds.overall.productdbversion > VersionTable</code></p> <p>The version table defines the versions that are available in a system. Each table containing BLOBs has a <code>versionId</code> column defining the version of the data in the BLOB.</p>
Coverage information in UpdateRegionTable	<p>In most update scenarios, the content provider makes the update data available at least for a country or comparable area, for example, an incremental update for Germany or Europe. Versioning information is used to find out whether an update is available for the area in question or not. In Europe this will likely be a country or a group of countries. In other parts of the world, appropriate areas can be different, for example, a group of states in US reflecting a particular time zone.</p> <p>DataScript location: <code>nds.overall.productdbversion > UpdateRegionTable</code></p>
buildingBlockId	<p>Identifies the building block types available in an update region</p> <p>Not all systems using the NDS format support all building blocks. It is possible, for example, that low-end systems do not contain the Digital Terrain Model building block. Thus, update processes require information about the building blocks supported by the current system.</p> <p>DataScript location: <code>nds.overall.productdbversion > BuildingBlockTable</code> and <code>UrBuildingBlockVersionTable</code></p>
creationDateTime	<p>Date and time of the database's compilation</p> <p>This information can be used to present version information in the navigation device (HMI).</p> <p>DataScript location: <code>nds.overall.productdbversion > VersionTable</code>, column <code>creationDateTime</code></p>
dataModelVersion	<p>Version of the NDS physical model used for this particular NDS database</p> <p>As NDS shall be backward compatible, applications compliant to a certain NDS version shall be able to handle maps compiled in an older NDS version. The NDS version will help to determine whether an update is possible or not.</p> <p>DataScript location: <code>nds.overall.version > DataModelVersionTable</code> (defines the NDS versions that are available in a system). Each table containing BLOBs has a column defining the NDS version of the data in the BLOB.</p>

Version data	Description
compilerVersion and compilerConfiguration	<p>Identification of the compiler used for generating the database</p> <p>Each compiler creates map data with minor and major differences compared to data created by another compiler. Minor differences may be a slightly different shape point reduction, generalization, or other optional attributes. Major differences may arise by assigning identifiers for gateways or by using different methods for referencing POIs. Even the same compiler supplier may have multiple generations of compilers with slight differences. Moreover, the same compiler may be operated with different compiler settings producing different results. Thus, it is required to identify both different compiler generations, and the same compiler with different settings.</p> <p>DataScript location: <code>nds.overall.productdbversion > VersionTable</code></p>

4.2.2 Business Version Data

Business version data is required for marketing and advertising purposes, as well as for presenting a clear and easy business version to the customer. This information can, for example, be used in sales materials or be presented to users when they want to find out the status of the databases installed in their navigation system. In case of an error, users will refer to the business version.

Thus, a business version needs to be human-readable. The business version has a direct relationship with technical version information for a particular database.

The business version data is stored in the `ProductDbTable` (fields: `productName`, `copyright`).

5 Versioning Concept

Navigation Data Standard offers flexibility for navigation products with regard to the composition of maps, the content, and also updates. To enable update scenarios as described in Section 2.3 *Update Scenarios* on page 14, a granular version concept is provided for NDS databases, which, for example, includes version identifiers for each part that may be updated.

5.1 Design Considerations for Versioning

The version concept for NDS databases serves two different tasks:

- Unambiguously identifying data, content, maps and providers
- Supporting flexible updates with an expected diversification of data content

For the selection of versioned items, the following aspects need to be considered:

- **Storage space:** The number of versioned items influences the storage space for version information. A version identifier needs at least one byte – therefore, not every attribute should get a version.
- **Absolute number of dependencies:** For versioned items that are updated together, dependencies between the versioned items may be neglected. The more atomic versioned items are grouped in non-atomic versioned items, the fewer dependencies have to be considered.
- **Relative number of dependencies:** Even though the absolute number of dependencies decreases if more versioned items are grouped, the relative number of dependencies per group increases. Relative dependencies can cause avalanche effects: A single update can cause the execution of a large amount of other updates because of the dependencies.
- **Freedom:** Large non-atomic versioned items reduce the freedom to select atomic versioned items for an independent update. Atomic versioned items belonging to the same non-atomic versioned item cannot be updated independently.

Figure 5-1 shows the pros and cons for selecting elements for versioning. The left column specifies the versioned items, the header row the aspects mentioned above.

Figure 5-1 Considerations for selecting versioned items





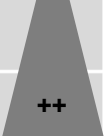







Versioned Item	Storage space	Dependencies (absolute)	Dependencies (relative)	Freedom
Single element (atomic versioned item)	Large 	Many 	Few 	Much 
Element group (non-atomic versioned item)				
Whole database	Small 	Few 	Many 	Little 

Figure 5-1 shows that selecting single elements as versioned items means that much storage space is needed, that there are many absolute, but only few relative dependencies, and that the freedom to select elements for update is high. Element groups, however, need less storage space, provide a lower number of absolute and a higher number of relative dependencies, and mean less freedom to select elements for update.

5.2 Subversion and Version Control for NDS

The problem of higher-level versioned items is very similar to handling directory versions in a software configuration management (SCM) system like Subversion. Thus, the version control concept defined for NDS adopts some of the concepts of Subversion.

Subversion is an open-source version control system. Subversion manages files and directories in a central repository and allows multiple users to work on local copies of (parts of) the repository in parallel. Possible conflicts are resolved by forcing users to merge different versions. Subversion stores the history of its repository and therefore enables the rebuild of any version of the directory/file structures. Subversion is often used as a version control system for source code in a multi-programmer environment.

The version concept for NDS is simpler as data always flows downstream, meaning that the application receives updated data from the NDS compilation facility, but never commits changes back.

Atomic versioned items correspond to files in the SCM repository. The non-atomic versioned items like building blocks and update regions correspond to directories in the SCM repository, which may or may not contain subdirectories.

Defining version numbers for a SCM directory is not trivial. If only some of the files in a SCM directory are updated, the directory version number cannot be simply increased to the latest file revision, because not all the changes for that revision have been implemented.

Instead, the concept of change sets and revisions in Subversion is also applied in NDS. Revision numbers are assigned globally for each transaction that changes the repository. The transactions are numbered sequentially. A transaction corresponds to a change set, that is, a set of differences applied to a set of atomic versioned items. The revision number of a file is the number of the latest change set or transaction containing a change to the given file. Thus, looking at a single file, revision numbers do increase monotonically, but not sequentially. When updating a file which is currently at revision 7, the new revision can be 12 and not 8, because transactions 8, 9, 10 and 11 were related to other files.

NDS adapts this concept as follows: The repository corresponds to the NDS database supplier's master database. It contains a set of atomic versioned items, for example, tiles. With the delivery of an NDS-compliant database, the version numbers of all versioned items have the same initial value. Such a general set and/or reset of version numbers is only done for a full update of an entire product database. After a new source data delivery (new input data from the map supplier) and compilation, some versioned items change. A compilation corresponds to a transaction and thus the revision numbers of the versioned items represent their versions. The initial version of a versioned item is 1.

For a given NDS database, the combination of all version numbers of versioned items and their dependencies is described by a tree. This tree is explained in Section 5.3 *NDS Version Tree* on page 41.

Atomic versioned items carry the version number of their last revision. The version number increases if the corresponding data is physically replaced. Non-atomic versioned items carry the version number of the latest complete revision of all underlying versioned items. If only a subset of the underlying atomic versioned items has been updated, the version of the given non-atomic versioned item is not changed. To be able to track that the current version does not reflect the status of the underlying atomic versioned items, the flag `isDirty` is provided:

- `isDirty = TRUE`: Revisions have not been applied to all underlying atomic versioned items and the version number of the non-atomic versioned item is not increased.
- `isDirty = FALSE`: The full set of the revisions has been applied to all underlying atomic versioned items and the version number of the non-atomic versioned item may therefore be increased.

5.3 NDS Version Tree

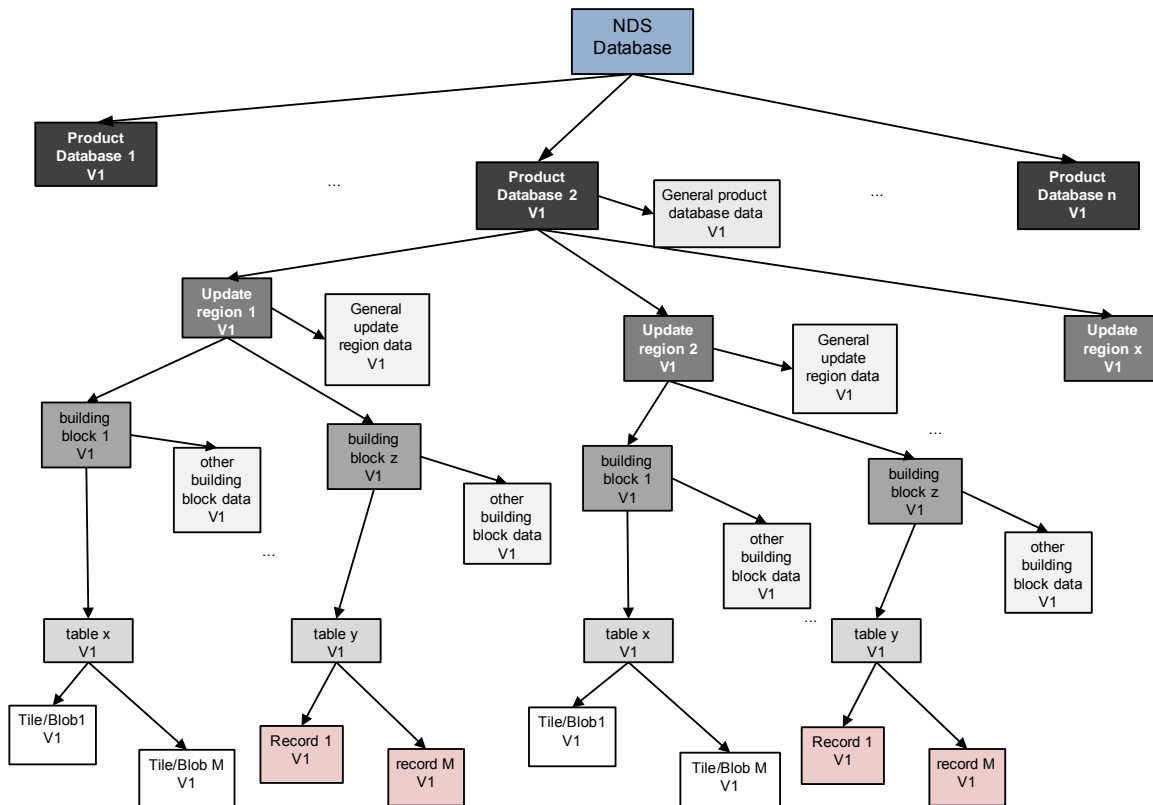
The versioned items build the version tree for a product database. The version tree is a hierarchical tree describing an NDS database. The root node is a product database which is further subdivided into update regions. An update region includes one or more building blocks. Each building block is further subdivided into relational tables containing records representing BLOBs, tiles, and other types of data.

An update region may have common data that is not associated with a single building block, but instead may apply to the whole update region. Likewise, there is common data for a building block and common data for a whole product database. All such common data is considered as metadata of the respective node in the version tree and underlies the same version control as regular data. See also *Update of Metadata* on page 35.

The leaf nodes in this version tree represent the tiles, BLOBs, or records within the different building blocks, stored in tables.

Figure 5-2 illustrates the NDS version tree.

Figure 5-2 NDS version tree



Update Scenarios

The NDS update concept defines several scenarios for updating NDS databases, for example, full or incremental updates of product databases and update regions, POI update, or map expansion. For more information on these scenarios, refer to Section 2.3 *Update Scenarios* on page 14.

Updates of update regions and map expansion are supported by the update region concept as described in Section 3.2 *Update Region* on page 23. Updates per building block (for example, POI) are supported by the building block concept of NDS databases. An incremental update is done by updating atomic versioned items, and is, therefore, the most complicated update scenario.

Versioning of Tree Nodes

From a technical point of view, the version of the atomic versioned items would be sufficient in order to define the version of the tree. During an update, it is, however, always necessary to traverse the whole tree if the version of the tree representing the original database is older than the version of the updated tree. Thus, it is an improvement if versioned items on a higher level receive an aggregated version of all underlying versioned items.

To ensure this, the NDS update concept defines the following rules for versioning of tree nodes:

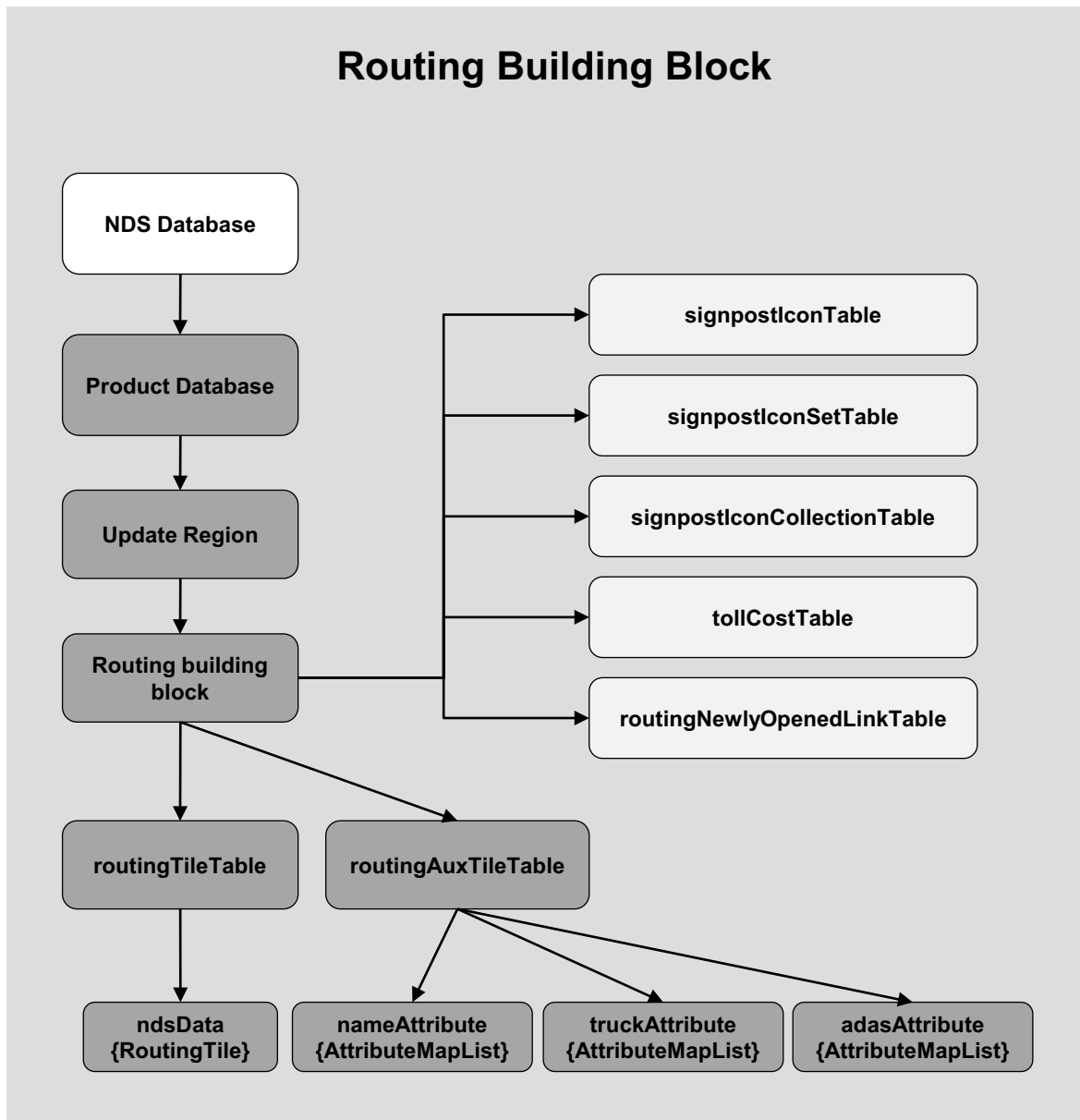
- Each node in the tree has a version number.
- In general, the version number of a node equals the highest version number of its children.
An exception from this rule may occur when an incomplete update of a node's children was performed. In this case, the node keeps the version number of the last complete update; only the version of the updated children is changed. The `isDirty` flag of the node is set to `TRUE` to indicate the incomplete update.
- The NDS database has no version number because it is possible that the versions of the underlying product databases are not aligned if they come from different database suppliers.

The following figure illustrates the versioning of tree nodes for the versioned items of the Routing building block. It shows which parent nodes need to be get a higher version number when the version of a child node changes. If, for example, an entry in the `RoutingNewlyOpenedLinkTable` is changed, this table and all its parents must get a higher version number:

- Routing building block
- Update region
- Product database

Updating the `RoutingNewlyOpenedLinkTable`, however, does not require an update of the `RoutingTileTable`, as both tables are child nodes of the Routing building blocks and are thus siblings in the version tree.

Figure 5-3 Dependencies for assigning version numbers to versioned items



The rules for managing the versioning of hierarchical trees in the NDS update concept are derived from software configuration management of a directory tree of files. For more information, refer to Section 5.2 *Subversion and Version Control for NDS* on page 40.

5.4 Version Control for Patches

The NDS update concept defines single version numbers for patches. Patches are treated in the same way as full updates: A patch version number is higher than the version number of the last full update. The atomic versioned items changed by the patch update are assigned a new version number. For more information about the patch update scenario, refer to Section 2.3.5 *Patch Update* on page 18.

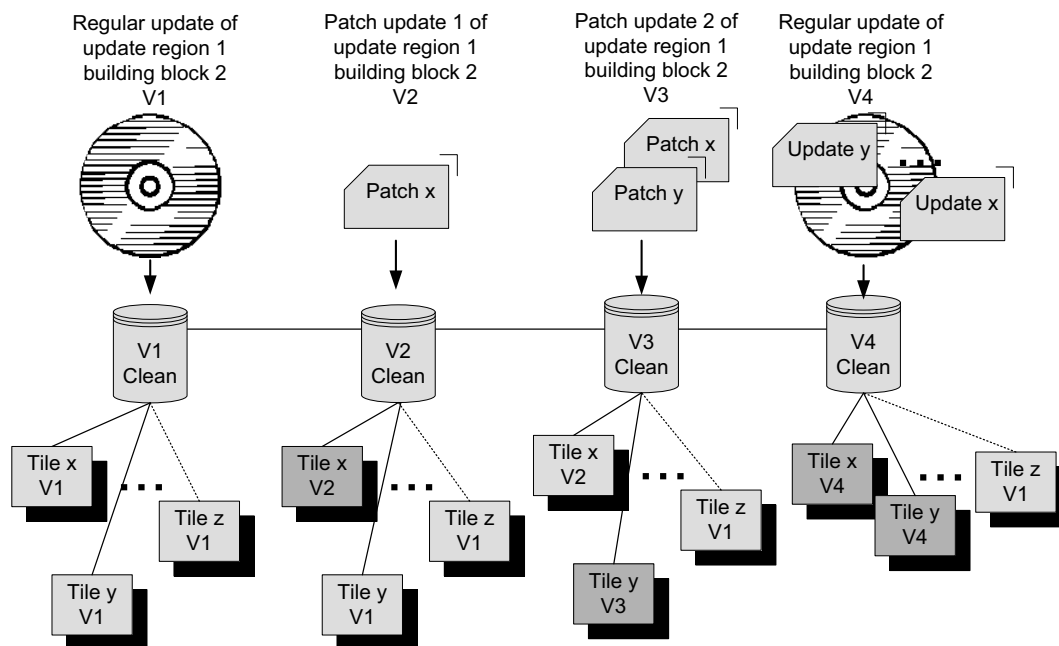
Patches shall be applied sequentially. A current patch, therefore, always has a higher version number than previous patches. Within a sequence of patch deliveries, a patch delivery *n* also includes all previous patches *n-1*, patch *n-2* ... patch 1.

The NDS database supplier decides which update transactions are published in a patch update and which will be published with a regular update. However, the consistency must not be affected by this decision.

The next regular update shall include all patch operations since the previous regular update. The version of the next regular update shall be higher than the version of the last patch update. A patch can be seen as a “dead” branch of the version tree and will be cut with the next patch or next regular update.

Figure 5-4 shows an example for the versioning concept for patch updates.

Figure 5-4 Patch update versioning

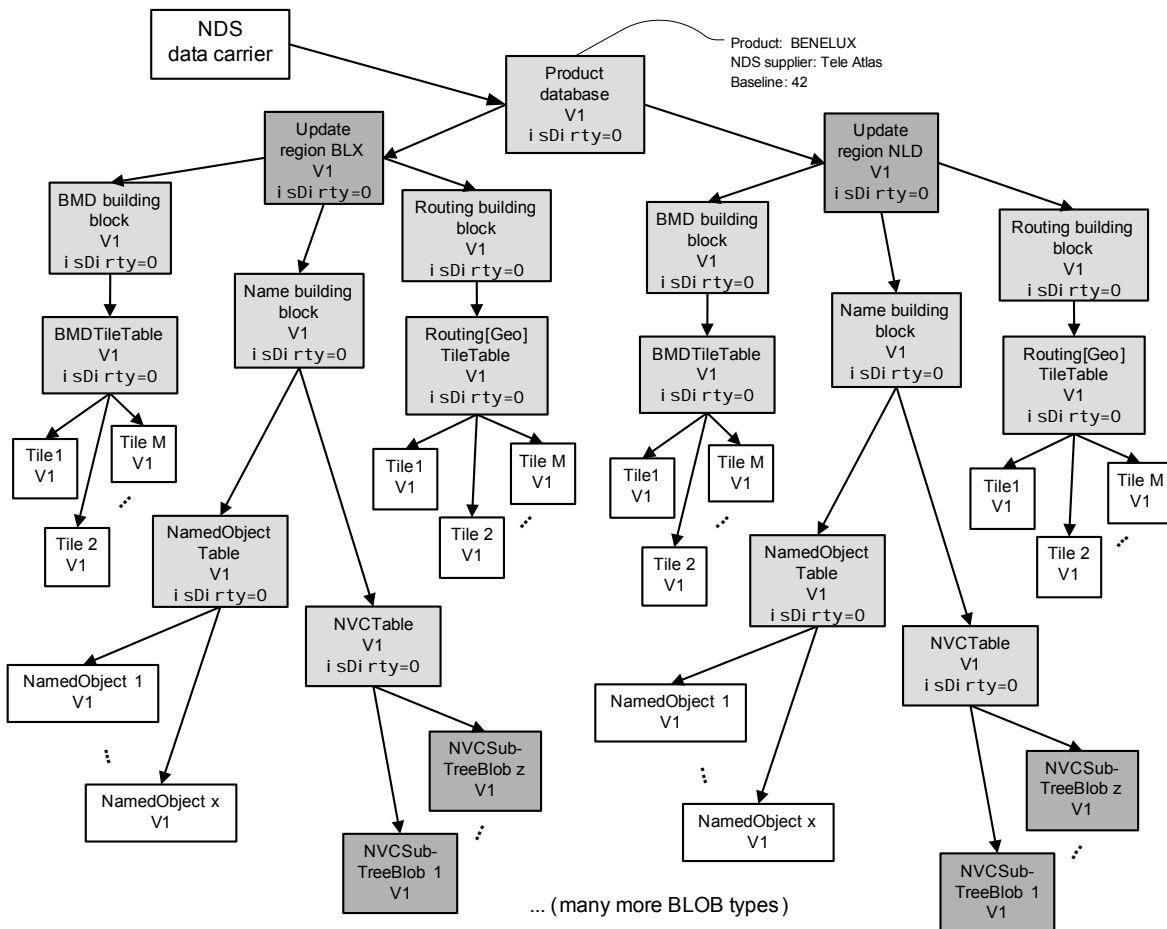


5.5 NDS Versioning Concept – Example

A navigation system has a full installation of an NDS product database specified as follows:

- Content: Belgium, Netherlands, Luxemburg
- Compiled by Tele Atlas
- Two update regions: BLX and NLD
- Building blocks: Basic Map Display, Routing, Name
- Version: V1

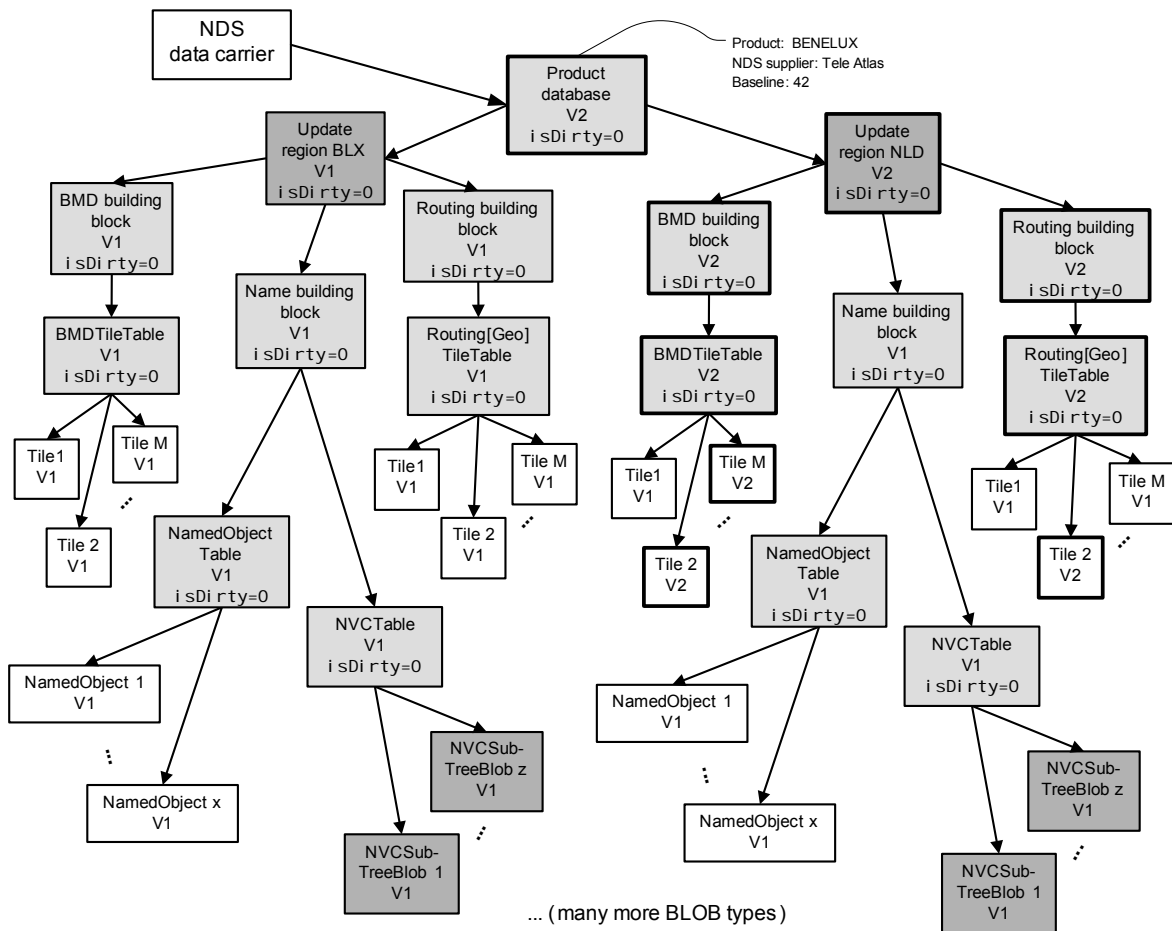
Figure 5-5 Version tree for the NDS database (version V1)



Version V2

The system receives a regional incremental update (V2) that includes only geometric and attribute changes in update region NLD, and thus replaces tiles in the Basic Map Display building block (BmdTile) and in the Routing building block (RoutingTile and corresponding RoutingGeoTile). The Name building block is not changed. All changes in the version tree are marked with black borders in the figure below.

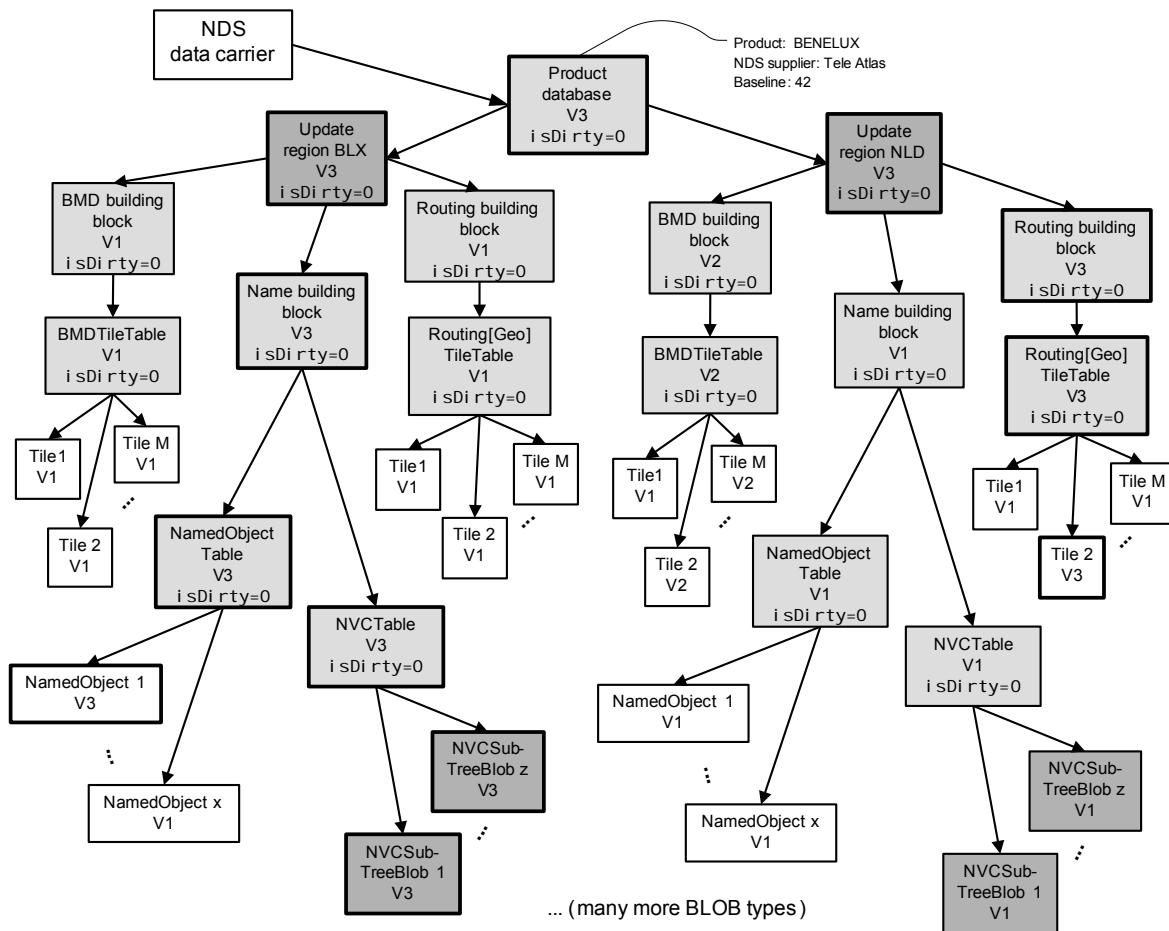
Figure 5-6 Version tree for the incremental update (version V2)



Version V3

The system receives the incremental update V3, including road name changes in update region BLX. The update replaces several *NvcSubTreeBlob* in the Name building block. The update also changes some attributes in the Routing building block of NLD. These changes do not affect the Basic Map Display building block.

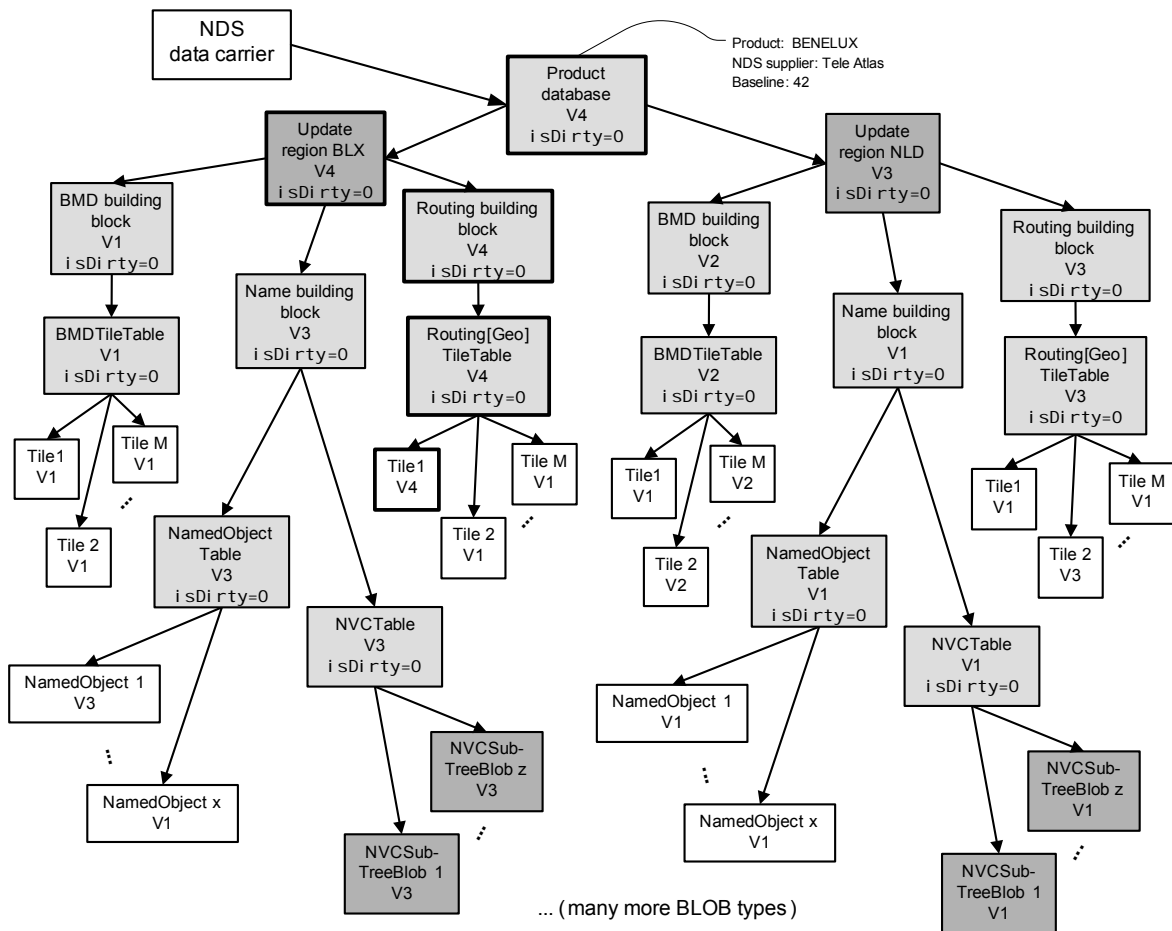
Figure 5-7 Version tree for the incremental update (version V3)



Version V4

Shortly after update V3, a major road is being closed for reconstruction in BLX. Thus, the NDS database supplier decides to publish a patch update V4. The change affects just a few attributes in one `RoutingTile` and in the corresponding `RoutingGeoTile`. Other real-world changes that occurred in the meantime and would affect adjacent tiles are not delivered. They are considered as minor and are thus published with the next regular release.

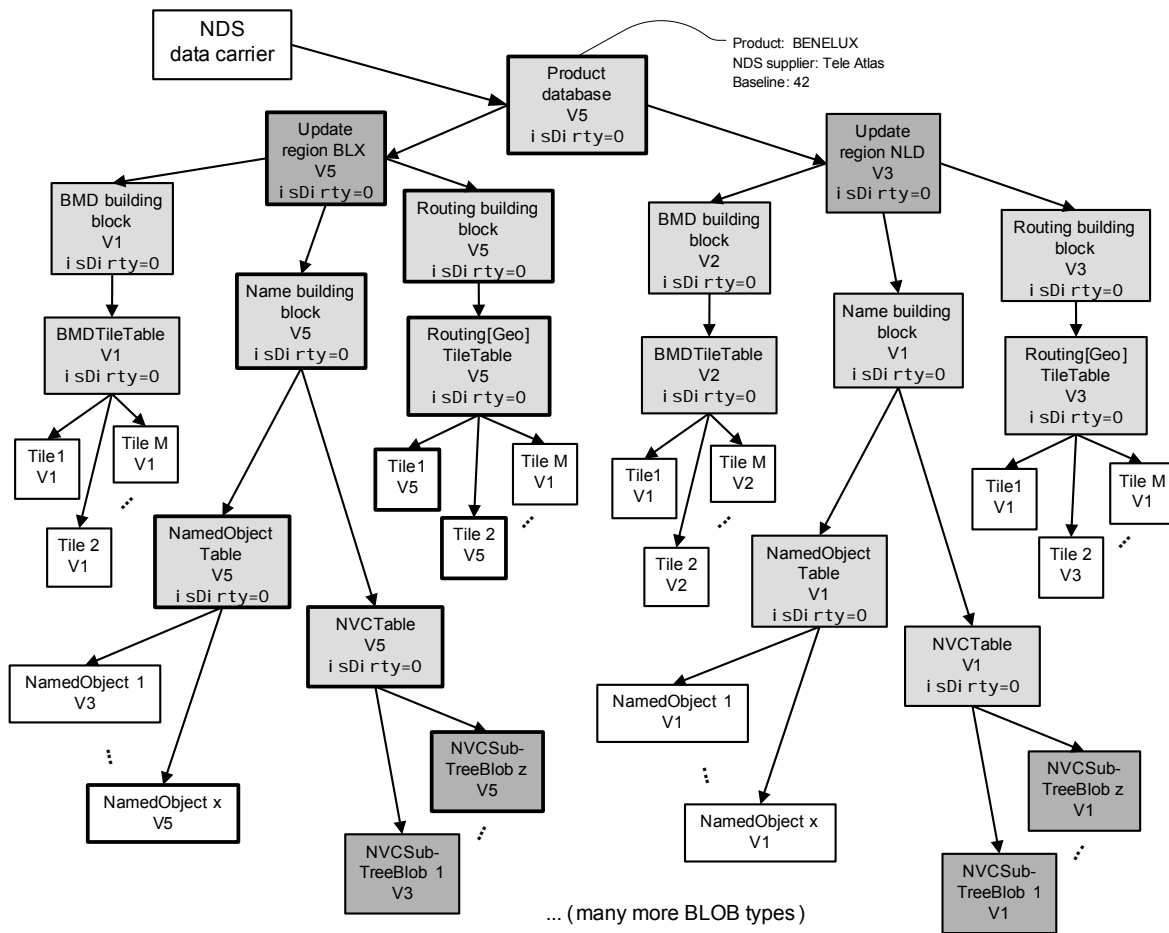
Figure 5-8 Version tree for the patch update (version V4)



Version V5

The system receives the next (regular) incremental update V5. It includes some changes, a redelivery of the patched `RoutingTile` and the corresponding `RoutingGeoTile` as well as the adjacent tiles with all changes that happened in the meantime. A regular update thus always includes the patches published after the previous regular update.

Figure 5-9 Version tree for the incremental update (version V5)



6 Gateways Between Update Regions

When dividing a product database into multiple update regions, it shall be ensured that each update region can be updated independently from the other update regions in the product database without putting the overall consistency at a risk. This is especially important for routing through several update regions.

The gateway concept describes how update regions can be compiled and updated independently while maintaining connectivity and how applications can find the gateways connecting the update regions, for example, when traversing topology for route calculation.

If changes in the real world extend across update region borders, the resulting database shall keep its topological consistency even if only one update region is updated. That means that it shall still be possible after the update to calculate a route from one update region to another via the location that has changed in the real world. An application shall, therefore, be able to find the intersection at the start or end of a route link also beyond update region borders. Likewise, it shall be possible to find all route links that are connected to an intersection on the update region borders. Cost estimations for route calculation, such as length or travel time, can become inaccurate for the non-updated part of a road affected by the change. The geometry of such a road network can show gaps at the update region border if only one region has been updated.

The resulting database shall keep its topological consistency after updating only one update region, also if:

- Changes in the real world extend across update region borders
- Changes in the real world in one update region result in topological changes to the adjacent update region

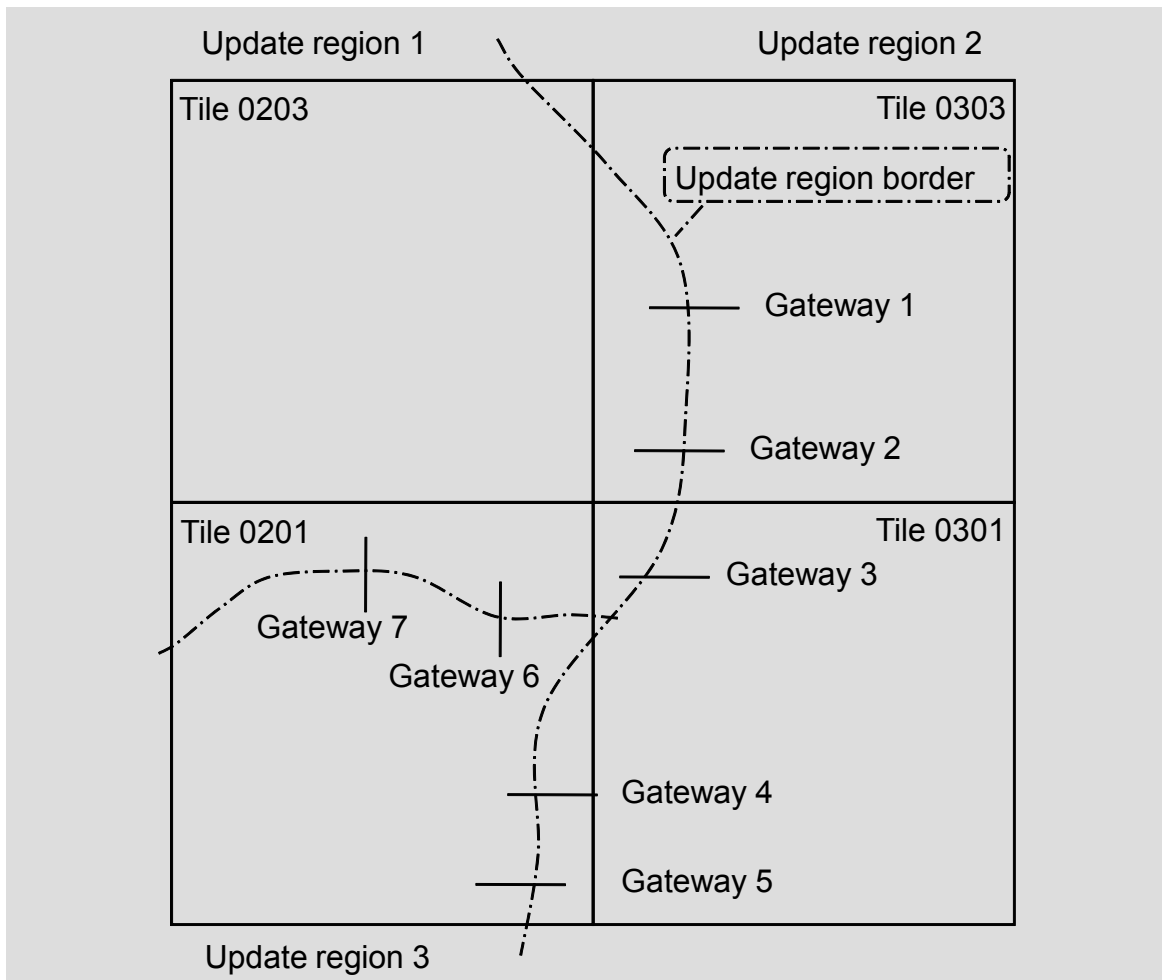
Assigning Gateway Numbers

To uniquely identify gateways, so-called *gateway numbers* are assigned. Figure 6-1 shows three update regions. Update region 1 is connected with update region 2 by three gateways numbered from 1 to 3. Update region 2 is connected with update region 3 by two gateways numbered as 4 and 5. The gateway numbers are unique per NDS product database. Adding a new update region to an NDS product database must not violate this rule.

The gateway numbers are stored in a global gateway table which is stored in the product database (DataScript location: `nds.overall.gateway`). The supplier of an NDS database is responsible for maintaining the gateway IDs.

A gateway keeps the same gateway number on all levels. That means, for example, that for an intersection gateway with gateway number 42 on level 13, the gateway number 42 must also be assigned to this intersection gateway on level 10 and level 8.

Figure 6-1 Gateways between update regions



Route Link Gateway vs Intersection Gateway

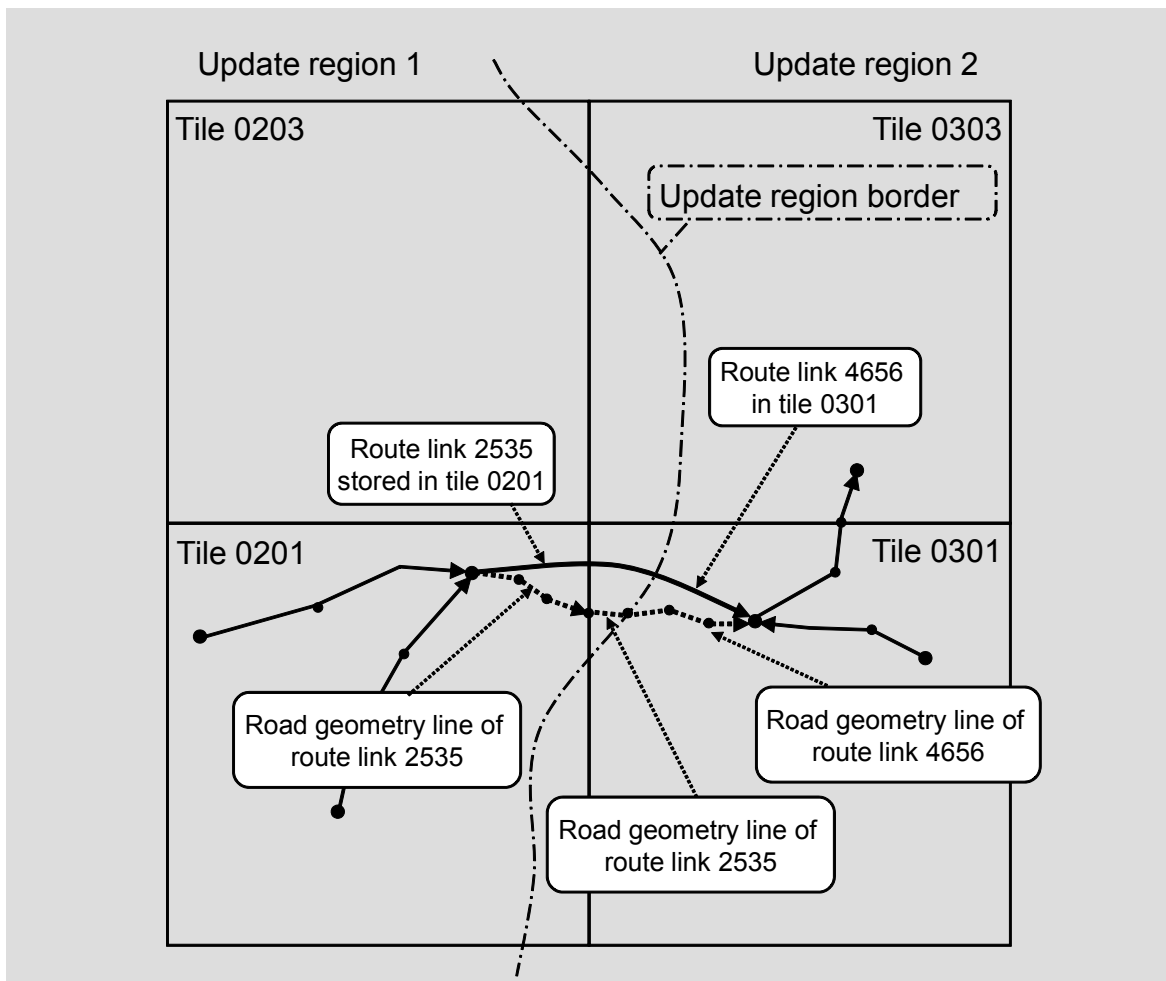
Two types of gateways exist: Route link gateways and intersection gateways. Which one is used in a particular database is decided at compilation time. An NDS application needs to support both types. Typically, the intersection gateway is more simple and supports most use cases. Route link gateways support specific corner cases.

The following sections introduce the concept of route link gateways and intersection gateways.

6.1 Route Link Gateways

Gateways between update regions may be created by means of route links. Figure 6-2 shows four tiles with an update region border crossing the tiles. As both update regions cover a part of each tile, the tiles are stored independently in both regions (see *Tiles at Update Region Borders* on page 25).

Figure 6-2 Route link as gateway between two update regions



A road is crossing the region border. The road element between the intersection in tile 0201 of update region 1 and the intersection in tile 0301 of update region 2 is represented by a route link. During the independent compilation of both update regions, this route link can get different IDs. For determining the attributes of the route link, the whole link shall be considered during compilation. It can, however, be assumed that the part in the adjacent update region is available for compilation.

Example In the example in Figure 6-2, the route link has the ID 2535 in update region 1 and refers to two road geometry lines in tile 0201 and tile 0301. The geometry ends at the update region border. In update region 2 the same route link has the ID 4656 and refers to a road geometry line in update region 2. This results in the following:

- The route link 2535 in tile 0201 in update region 1 has the following flexible gateway attribute:
gatewayNumber = 15
 - The route link 4656 in tile 0301 in update region 2 has the following flexible gateway attribute:
gatewayNumber = 15
 - The GlobalGatewayTable has the following entry:
Gateway 15; updateRegionId 1; tile 0201; level 13
 - The GlobalGatewayTable has the following entry:
Gateway 15; updateRegionId 2; tile 0301; level 13
-

For cross-update region connectivity, a GlobalGatewayTable must be stored per product database along with other metadata.

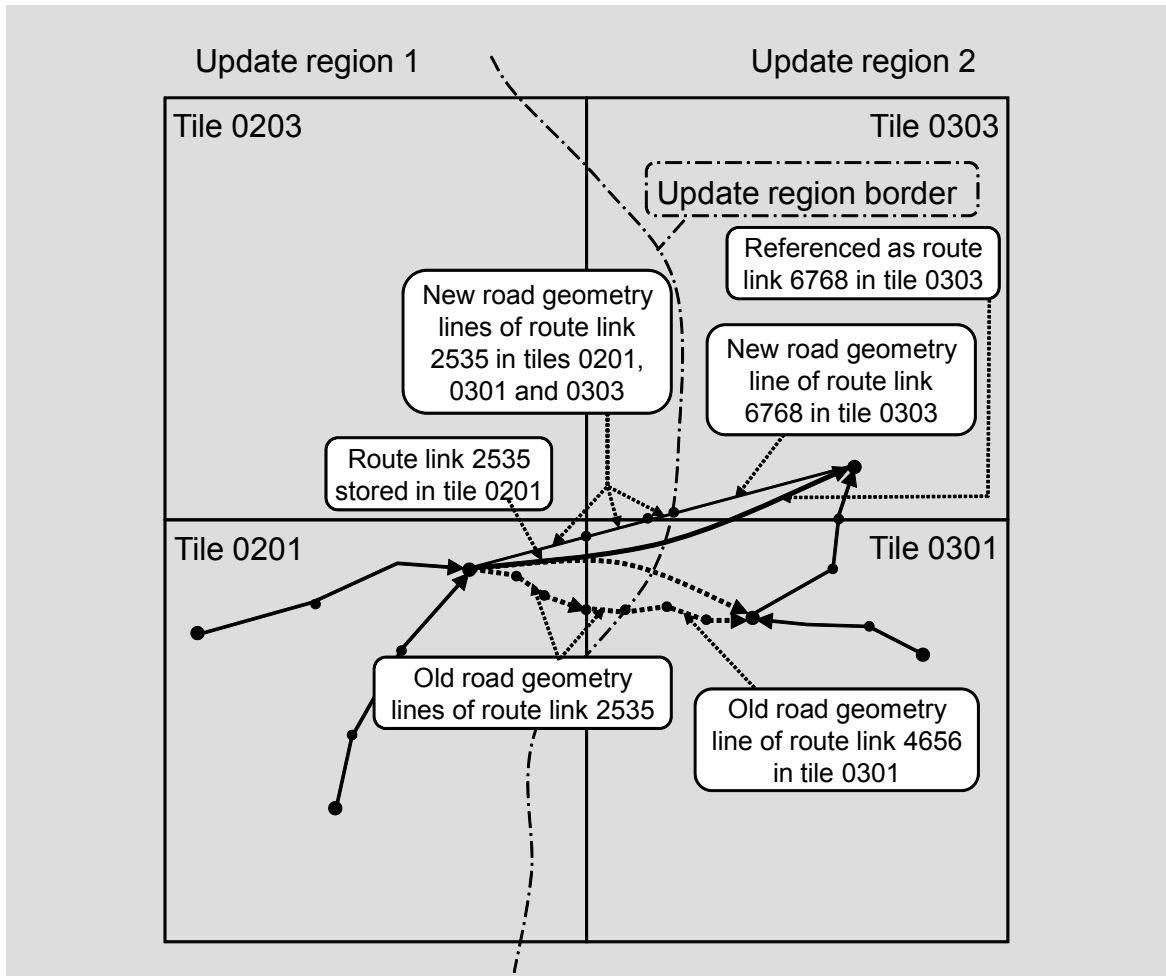
Also, the following rules for gateways must be applied to cover special cases, such as:

- A route link gateway shall intersect an update region border only in one point.
- Road elements that cross an update region border more than once shall be cut.

6.1.1 Changes at Route Link Gateways

In Figure 6-3, the road crossing the update region border from Figure 6-2 has been moved to an intersection at its end from tile 0301 to tile 0303 as a result of an update in update region 2.

Figure 6-3 Changes at a route link gateway



The consequences are:

- Route link 2535 in update region 1 now refers to three new road geometry lines in tile 0201, tile 0301, and tile 0303
- Route link 4656 in tile 0301 of update region 2 has been replaced by route link 6768 in tile 0303, which also refers to a new road geometry line.
- Route link 2535 in tile 0201 in update region 1 has still the same flexible gateway attribute: gatewayNumber = 15
- The GlobalGatewayTable has the following entry:
Gateway 15; updateRegionId 1; tile 0201; level 13

- Route link 4656 in tile 0301 in update region 2 does not exist anymore. Instead, there is a new route link 6768 in tile 0303 in update region 2 with the following flexible gateway attribute:

gatewayNumber = 15

- The GlobalGatewayTable entry for update region 2 has changed to:

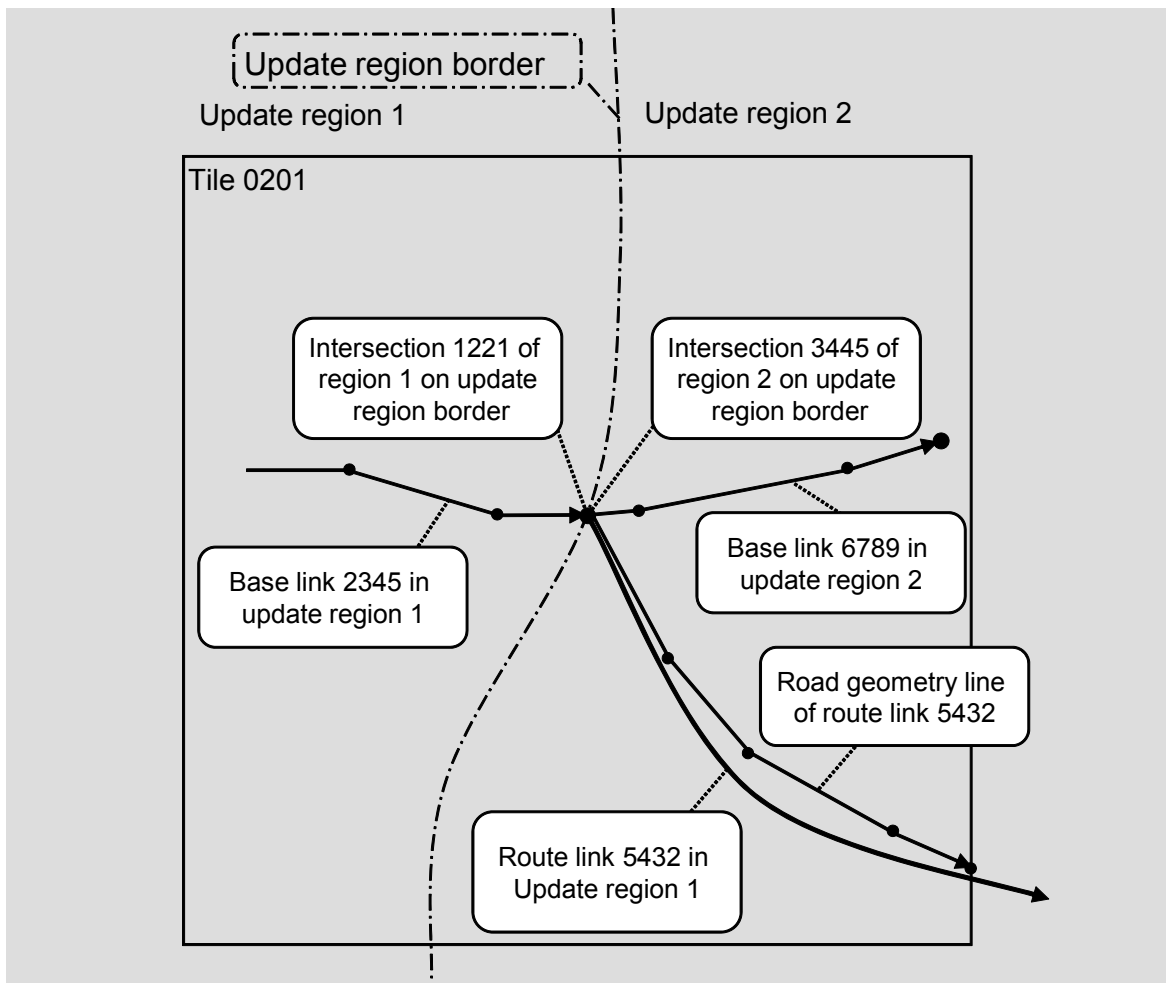
Gateway 15; updateRegionId 2; tile 0303; level 13

The references to the road geometry lines do not affect the GlobalGatewayTable. They are tile-internal changes of the routing data and are modified when the update region is updated. If only one update region is updated, a gap in geometry occurs at the tile border. The connectivity and consistency of the network, however, is still maintained. An application can retrieve the route link attributes from the updated update region.

6.2 Intersection Gateways

In most cases, an intersection gateway can handle update region borders more conveniently than a route link gateway, for example, if a real-world intersection is located directly on an update region border. Figure 6-4 shows an update region border within a tile. The road network of both update regions is connected at an intersection located directly on the update region border. The example shows a real intersection with three adjacent streets. The method described below, however, would also apply if the intersection on the update region border is a pseudo intersection caused by the delivery of two independent product databases with two adjacent streets.

Figure 6-4 Intersection as gateway between two update regions



Both update regions are compiled independently and the intersection therefore gets a different ID in each update region. The number of connected streets stored with the intersection refers to the number of links stored within the update region. This is why intersection 1221 in update region 1 has only one connected link (base link 2345) and intersection 3445 in update region 2 has two connected links (base link 6789 and route link 5432).

To enable the application to merge the intersections on both sides of the update region border, each of them has the flexible gateway attribute:

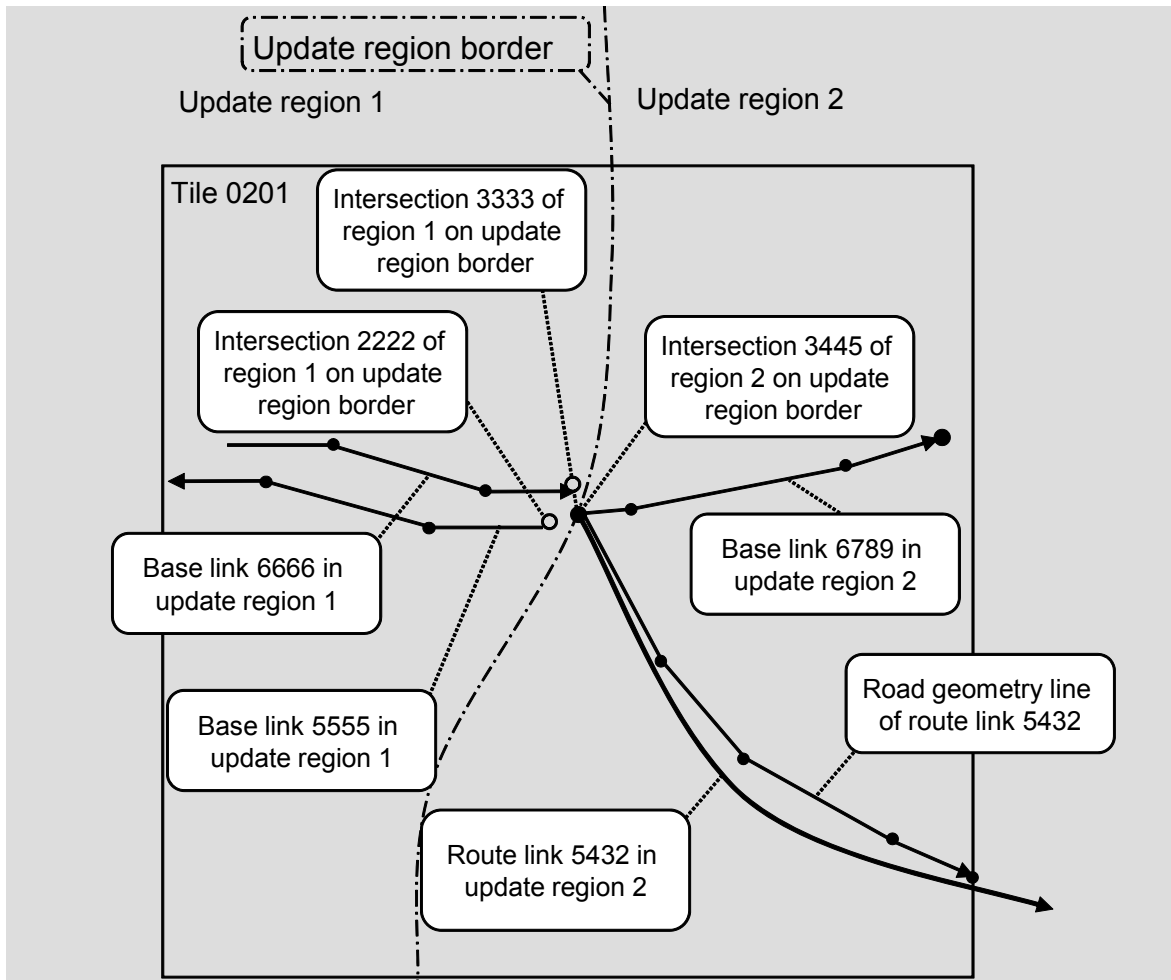
- Intersection 1221 in tile 0201 in update region 1 has the following flexible gateway attribute:
gatewayNumber = 15
- Intersection 3445 in tile 0201 in update region 2 has the following flexible gateway attribute:
gatewayNumber = 15
- The GlobalGatewayTable has the following entry:
Gateway 15; updateRegionId 1; tile 0201; level 13
- The GlobalGatewayTable has the following entry:
Gateway 15; updateRegionId 2; tile 0201; level 13

On the basis of this data, the application can use the GlobalGatewayTable to route from update region 1 to 2 and vice versa via gateway 15.

6.2.1 Changes at Intersection Gateways

Figure 6-5 shows the example from Figure 6-4 with the road in update region 1 extended to a dual carriageway. This leads to two new route links (5555 and 6666) and two new intersections at the end of this road (2222 and 3333). The intersections are drawn a bit aside for clarity, but are both still located on the update region border.

Figure 6-5 Changes at an intersection gateway



To maintain the topology between the two update regions, the following changes are necessary:

- Intersection 2222 in tile 0201 in update region 1 has the following flexible gateway attribute:
gatewayNumber = 15
- Intersection 3333 in tile 0201 in update region 2 has the following flexible gateway attribute:
gatewayNumber = 15
This means, that both new intersections which replace the old intersection need to keep the gateway number of the predecessor.
- The GlobalGatewayTable remains unchanged.

6.3 Gateway Concept vs. Raw Data Specifications

The major map suppliers deliver intersections on database borders twice: One intersection in each intersecting database. An NDS compiler can rely on this and mark both via intersection gateways.

The situation differs for links at database borders. In this case, the application cannot rely on the map suppliers to be consistent with each other and must account for discrepancies between different numbers of links at the borders.

6.4 Installation of Update Regions

The application can check if the GlobalGatewayTable has been changed by checking its version in the BBlockCompVersionTable. In case of changes, the GlobalGatewayTable can be updated with the following SQL commands:

```
DELETE from GlobalGatewayTable
WHERE updateRegionId = N
```

N is the ID of the update region whose gateways have been changed.

```
INSERT INTO GlobalGatewayTable
SELECT * FROM UpdatedB.GlobalGatewayTable
WHERE updateRegionId = N
```

N is the ID of the update region whose gateways have been changed. UpdatedB is the NDS database containing updates.

In this way, an update region can be added without changing other existing update regions.

This approach allows flexible combinations of update regions. An update region of Germany could, for example, be combined with an update region of Poland or with an update region of East Europe (containing Poland). For the update region of Germany, both is possible.

A Glossary

This chapter explains the most important terms and key concepts that are necessary to understand the update scenarios and the concept.

Baseline Map

A baseline map denotes an NDS map release from a particular NDS database supplier that is used as single basis for compatible product databases and updates.

Data Carrier

Generally, a data carrier is a physical medium to persistently store digital data, for example, a DVD, a solid state memory stick, or a hard disk. In the context of this document, a data carrier is a medium to store data complying with the NDS format. A data carrier may be permanently installed in an NDS-compliant system, for example, a hard disk, or be used to transport data into an NDS system, for example, a DVD.

Gateway

A gateway establishes a topological connection from the Routing building block of one update region to the Routing building block of another update region by assigning a permanent and unique ID to the fragments at both sides.

NDS Database

A set of data complying with the NDS format, which is stored in a particular navigation system. It includes one or more product databases. For example, two product databases can be installed in system X: *Navigation Data Europe 2008.01* and *Points of Interest – Travelguide 2008*.

NDS Database Supplier

An NDS database supplier is any party that converts map data into NDS format. The NDS standard provides a worldwide unique identification of NDS database suppliers, see *NDS – Format Specification*, B *NDS Database Supplier IDs* on page 369.

Product Database

A set of navigation and other data complying with the NDS format and created by a single NDS database supplier. It underlies a unique version control, and it may be subject to updates. Product databases may contain one or more building blocks and cover a geographic area. The geographic area may be further divided into several update regions. For example, there may be a product database *Navigation Data Europe 2008.01* from vendor A, and another product database *Points of Interest – Travelguide 2008* from vendor B.

Update Region

A product database is further subdivided into one or more update regions. An update region represents a geographic area of a product database that may be subject to a regional update. Update regions do only overlap at their borders, their interiors are disjoint.

Update Method

Update methods describe how updates are executed: by exchanging the complete database, by exchanging relational tables, or by replacing a subset of records. To cover the update scenarios defined by NDS, different update methods may be used, see *2.4 Update Methods Used for Update Scenarios* on page 20.

Update Scenario

Update scenarios describe what needs to be updated or added to a database, for example, a country that needs to be added to a product database. The update scenarios may be realized by one of the available update methods, see *2.4 Update Methods Used for Update Scenarios* on page 20.

Versioned Item

A versioned item represents a portion of a product database that underlies version control. A versioned item shall always be updated as a whole in one update transaction. This ensures consistency and structural integrity of an updated database. A versioned item may be a logical NDS element, such as country and region, or a physical NDS element, such as building block and BLOB. It can depend on other versioned items, meaning that an update of a versioned item can require a previous update of another versioned item.

B Index

A

architecture
 elements for update 13
 products and update regions 23
 attribute update 19

B

BLOBs, versioned items 33
 business version data 37

C

changes at intersection gateways 59
 changes at route link gateways 55
 concept
 elements for updates 13
 products and update regions 23
 versioning 39
 content, adding to database 18

E

elements for update 13
 example
 changes at intersection gateways 59
 changes at route link gateways 55
 patch update versioning 45
 versioning concept 46
 expansion, map 17

F

full update 14

G

gateway
 assigning numbers 51
 definition 51
 for ferries, tunnels, bridges 31
 intersection 57
 route link gateways 53

geographical extent of update regions 24

I

incremental update 14
 product database 15
 update region 16
 intersection gateway 57

M

map expansion (update scenario) 17
 metadata update 35

N

numbers, assigning to gateways 51

O

ocean and ferry update region
 definition 30
 gateways 31

P

parameter
 versioning 35
 partial update 14
 partially filled update region
 consistency at update 27
 definition 27
 update example 28, 29
 patch update
 example for versioning 45
 update scenario 18
 version control 45
 patch update versioning, example 45
 POI
 update 19
 versioned item 33
 product database
 adding content (update scenario) 18
 definition 23

incremental update 15
introduction 13

R

route link gateway 53

S

scenario
 transmission scenario 21
 update scenario 14
shared tables 34
Subversion 40

T

technical version data 35
tile
 at update region border 25
 versioned item 33
transmission scenarios 21
type of update region 27

U

update
 attributes 19
 elements for update 13
 full (update method) 14
 incremental (update method) 14
 metadata 35
 methods 14
 of update regions 25
 partial (update method) 14
 patch 18
 POIs 19
 scenarios 14
 transmission scenarios 21
update method
 definition 14
 for update scenarios 20
 full update 14
 incremental update 14
 partial update 14
update region
 definition 23

gateways 51
geographical extent 24
implementation 25
incremental update 16
installation 60
introduction 13
ocean and ferry 30
partially filled 27
tiles at update region border 25
type 27
update 25
world overview map 32
update scenario
 adding content 18
 attribute update 19
 definition 14
 incremental update 15, 16
 map expansion 17
 patch update 18
 POI update 19
 update methods 20

V

version control
 definition 40
 patches 45
version data
 business 37
 technical 35
version tree
 definition 41
versioned item
 atomic 33
 definition 33
 introduction 14
 non-atomic 33
versioning
 business version data 37
 concept 39
 design considerations 39
 example for patch 45
 examples 46
 parameters 35
 technical version data 35
 version control 40

version tree 41

W

world overview map 32

C Index of DataScript Terms

A

AttributeMapList 33

B

BBlockCompVersionTable 34, 60

BmdTile 33, 46

BmdTilePattern 33

buildingBlockId 36

BuildingBlockTable 36

C

ColorTable 34

compilerConfiguration 37

compilerVersion 37

copyright 37

creationDateTime 36

D

dataModelVersion 36

DataModelVersionTable 36

DtmSurfaceTile 33

G

gatewayNumber 54, 55, 56, 60

GlobalGatewayTable 54, 55, 56, 58, 60

H

HeightMapData 33

I

isDirty 41, 43

isPartiallyFilled 25, 27

N

NamedObject 33

NamedObjectPhoneticTranscriptionListBlob 33

NdsDatabaseSupplierTable 35

NdsDbSupplierId 35

NvcSubTreeBlob 33, 47

O

OrthoImageTile 33

P

PoiCategoryTable 35

PoiMetadataTable 35

ProductDbTable 37

productName 37

R

RoutingGeoTile 33, 46, 48, 49

RoutingNewlyOpenedLinkTable 43

RoutingTile 33, 46, 48, 49

RoutingTileTable 43

U

UpdateRegionTable 36

UrBuildingBlockVersionTable 25, 27, 36

V

versionId 25, 36

VersionTable 36, 37