Navigation Data Standard

Compiler Interoperability Specification

NDS Version 2.1.1.9 and 2.1.1.10.1 - 2.1.1.10.6

(Preliminary Version)

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.

Document version 2.0 (based on NDS versions 2.1.1.9 and 2.1.1.10.1 – 2.1.1.10.6)

Table of Contents

List of Figures

List of Tables

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	2009-04-24	Initial version for NDS version 1.8.2
1.1	2009-07-10	Bug 387: Replace filename extension CIDX and FIDX with CDX and FDX Replace "file extension" with "filename extension" Chapter *Profile Attributes* added Bounding Box definition added in Table 2-4 *Fields in the update region table* Chapter *POI Building Block* added [Bug 380] Information added to Section *A Selection Graph Definition Must be Complete* [Bug 353] Change Case Marker: Info added in Chapter *Name Building Block*, Section Change Case Marker on page 100 [Bug 379] *NamedObjectLevel* information: Section 7.4 Allocation of Named Objects to Basic Map Display and Routing Levels on page 93 enhanced [New] Information about instantiation of DataModelVersionTable added: – Section *ProductDatabaseTable* enhanced – Paragraph *Version Table* moved to Section 2.1.1 Product Database Structure on page 28 – Section on *DataModelVersionTable* added to Section 2.1 *File Structure of NDS Databases* on page 27 [New]Filename extensions in capital letters (decision in 8.3) [New] Chapter 10 POI Building Block on page 131 Enhanced definition of named object references (Section 7.6.4 Assigning Named Objects to Routing and Basic Map Display Tiles and Features on page 106 Section *House-to-Road Relations* (access at road in DataScript) for address retrieval changed in *Name Building Block* chapter

Version number	Published	Changes
1.2	2009-10-19	[Bug 433] Missing entry in Table 6-2 [Bug 449] DRM: added note about CTR start value Information on attribute layer and attribute points added to Chapter 5.9 *Handling Attributes* on page 82 Title of Figure 4-1 changed [Bug 462] Rename `isAdvancedAdminSearchAvailable to hasAdvancedAdminSearch` [Bug 434] Tables added to Figure 2-1 [New] Chapter *Full-text Search Building Block* added Section 6.7.1 *Filtering of Links by Road Classes* enhanced with Road Classes 5 - 7 Enhancement of the *POI Building Block* chapter Enhancement of Section *Handling Attributes of Aggregated Links* Enhancement of Section *Flexible Attribute Mechanisms* Changes and enhancements (new subchapters) in *POI Building Block* chapter [Bug 436] Define default values for flexible attributes [Bug 398] Info that a common prefix is allowed only for leaf nodes added to *Name Building Block* chapter [455] Info added to *Routing Building Block* chapter (new section above Table 6-3, Fig 6-7) [Bug 390] Info added to Section *Update Regions* in Chapter 3 *General Processes* [Bug 496] Added paragraph about `link2TileTable` in Section 6.3 [Bug 441] Section on range tables added to Chapter 3 *General Processes* Min scale denominator description in *POI Building Block* chapter changed [New] Chapter “Orthoimages” added [Bug 386] Named object feature class Signpost added to Section 7.6.5 [Bug 526] New paragraph in section 10.5 Chapter “Profile Attributes” moved to Routing BB chapter FTS BB chapter: Changes to metadata section; poiMortoncode column removed; reversed terms concept added [Bug537] SQLite Autovacuum allow Incremental in Interop Spec: Value 2 (incremental) added [New] Chapter on Data type mapping added to Chapter 3 General Processes [New] Chapter “Digital Terrain Model Building Block” added

Version number	Published	Changes
1.3 (for NDS 2.1)	2010-01-20	*Name Building Block* chapter: Major changes due to task force modifications *Routing Building Block* chapter: Major changes due to task force modifications [Bug 615] Use of modal verbs corrected and section *Use of Modal Verbs* added to *About this Document* chapter Use of online/compile time/runtime harmonized [Bug 605] Quotation marks corrected Table 15-1 in *Full-text Search Building Block* chapter: Error in first two rows corrected [Bug 601] *Height Map Data* section changed according to Bug 601 [Bug613] Range Tables start with 0, not with 1. Section 3.5 *Range Tables* changed accordingly [Bug 638] List of flexible attributes for Routing and assignment to BLOB/layer added in Appendix [Bug 530] Changed `ROAD_WIDTH` to `PHYSICAL_WIDTH_METRIC` in Section *Attribute Lists and References* [Bug 598] Row `buildingBlockId` added to table 15-1 [Bug 574] Explanation how to use/assign the data quality attributes added to section 6.6 [Bug 610] Added rule that NVC trees must have a unique entry point to section 7.5.2

Version number	Published	Changes
1.4 (for NDS 2.1.1 and 2.1.1.1)	2010-06-03	Updated Appendix A *Assignment of Flexible Attributes to Routing Tiles and Layers* on page 161 with new flexible attributes Added Appendix 17 *Using Flexible Attributes and Attribute Groups* on page 213 [Bug 625] Changes due to removal of `ENHANCED_ATTRIBUTION`; updated 5.6 *Assigning Data Quality Attributes* on page 76 [Bug 690] Added information on route guidance attributes in *Routing Building Block* chapter [Bug 677] Added information about assigning several junction view images to specific positions on a link to chapter 15 *Junction Views* on page 203 [Bug 424] Max. number of intersected tiles in geographical extent of named objects [Bugs 600 and 607] Created topic 13.3 *Handling Orthoimage Tiles* on page 192 [Bug 611] `isShort` option for NVC tree nodes [Bug 753] One or several signpost named objects allowed for modelling signposts with several names [Bug 707] Added information on handling lane marking attributes [Bug 618] Changed table 15-1, row `buildingBlockId`, description field [Bug 538] `nds.amd` restructured in DataScript – Changes done accordingly in Compiler Interoperability Specification for *Digital Terrain Model* and *Orthoimages* chapters [New] Section *Labeling Hints* added to *Basic Map Display Building Block* chapter [Bug 749] Info about NVC subtree IDs added to Section *Size and Partitioning* of the *Name Building Block* chapter [New] Section on Link Angles added to *Routing Bulding Block* chapter [Bug 468] Section „Junction View BB/POI BB – References to Routing“ added to Appendix B [Bug 616] Info about 3D Junction Views added to *Junction View Building Block* chapter [Bug 702] New `RouteLinkRefType` for POIs [Bug 676] New flexible attribute for physical lane dividers [Bug 495] AES-256 changed to AES-128 in *Encryption* chapter [Bug 503] Section on travel direction attribute and conditions added to Section 6.9 *Handling Attributes*

Version number	Published	Changes
Version 1.4 contd.		[Bug 561] Section 10.8 *Handling of Secondary POI Attributes* added to POI chapter [Bug 730] Added Section 7.6.5 *Administrative Area Named Object Reference* to *Name Building Block* chapter [Bug 777] Added information about 3D junction view arrows [Bug 682] Use case for toll gates added to Appendix B [Bug 697] Use of `isPseudo` flag for identifying split area features that belong together [Bug 813] `ColorTable` example added to Section on Building Block Component Version Table [Bug 841] Note about empty database tables reformulated according to bug [Bug 608 and others]: Added information about new versioning concept for relational tables and colums to be used, for example, for attribute and metadata updates. See 2.1 *Distribution of Tables over SQLite Files* on page 31 [Bug 893] Reference location for relative URIs - sentence added to note in Section 2.1 [Bug 876] and others: Integration of prerecorded voice, renaming of Phonetic Representation building block to Speech building block
1.5 (for NDS 2.1.1.2, 2.1.1.3)	August 2010	[Update] Added entries to list of indices in Appendix B *List of Indices for NDS Relational Tables* on page 249 [Bug 900] Changed the DataScript locations of the fixed attributes `startAngle`, `endAngle` and `averageSpeed` to `nds.routing.link` [Bug 792] Added glossary entry for profile [Bug 1050] Selection criteria in Name building block; statement that cities and city districts are mandatory instead of place
1.6 (for NDS 2.1.1.4, 2.1.1.5)	October 2010	[Bug 861] Added definition of south-west rule in Glossary [Bug 1035] Added a note regarding the rules for the number of lane definition to Appendix A.2 [Bug 1089] Added section on compression [Bug 1252] Added information to *Signpost* chapter in Appendix B that signpost information can be attached to links and road geometry lines, where the sign is positioned in reality. [Bug 1074] Added information about handling continued turn restrictions on higher level to Section 5.8 *Generalizing Features for Higher Levels*

Version number	Published	Changes
1.7 (for NDS 2.1.1.6)	2011-02-07	[Bug 620] Explained drawing order in more detail. Added information about drawing order of road geometry lines [Bug 826] Changed section on optimization options for NVC trees [Bug 834] Changed DataScript locations for Orthoimage metadata [Bug 1074] Added example of CTRs to Section 5.8 *Generalizing Features for Higher Levels*; adapted content accordingly [Bug 1137] Harmonized section 5.4 *Sorting links at Intersections* with Section 5.10 *Link Angles* [Bug 1283] Added information to Section 5.9 *Handling Attributes* that transitions must not be used for one-way streets, but one-way directions must be evaluated. [Bug 1318] Added section about Multiplexing of NDS dataase files to chapter 2 *Setting up the Database* [Bug 1352] Labeling line features removed from Chapter *Basic Map Display Building Block* as DataScript implementation is not yet available
1.8 (for NDS 2.1.1.7)	2011-06-01	[Bug 1341] Changed attribute names `BY_PASS` and `SHORT_CUT` to `BYPASS` and `SHORTCUT` in Appendix B. [Bug 1439] Replaced phonetic *representation* with phonetic *transcription* [Bug 1453] Renamed `inclusiveExclusiveFlag` to `isInclusive` [Bug 1504] Changed name of attribute `SINGLE_OR_DUAL_LANE` to `SINGLE_OR_DUAL_CARRIAGEWAY` in Appendix B.6 [Bug 1443] Changed description of extended postal code to relative geographic position [Bug 1564] SQlite version 3.7.4 min requirement [Bug 1604] SQLite version 3.7.4-nds3 min requirement [Bug 1584] Added sentence to Section 5.8.2 *Handling Attributes of Aggregated Links* that link angles on higher levels have to be equal to the corresponding angles on level 13 [Bug 1357] Changed section completeness and correctness of selectio graph definition [Bug 1307] Added Section 3.7 *Sorting Order for Signed and Unsigned Values* to Chapter 3 *General Processes* [Bug 1367] Changed Example in Table B-3 *Sample settings for* `LANE_CONNECTIVITY_ACROSS_INTERSECTIONS` to include example with a link without lane information [Bug 1230] Changed use of aggregation named objects in NVC trees to optional.

Version number	Published	Changes
1.8 ctd.		[Bug 1650] Added new section 9.10 *References to BMD Features* to the POI Building Block chapter. [Bug 1370] Added new section *References to 3D Objects* to POI building block chapter; cross-referenced this chapter also with Section 13.3 *References into 3D Objects Building Block* [Bug 1122] Renamed „polygon primitives“ to“polygon types“ [Bug 1406] Added section about reconstructing information from leaf and reference nodes in NVC trees to *Name Building Block* chapter [Bug 1249] Added/changed information in section *Version Table* (Chapter 2 *Setting up the Database*) [Bug 1545] Added note in Section 13.1 *Partitioning of Spatial Trees* that ID of root subtree is defined as 0. [Bug 1478] Removed `languageCode` from `IconCollectionTable` in Appendix C *List of Indices* [Bug 1385[Added information about chunk size to Section *Multiplexing/Striping for SQL Files* in Chapter 2 *Setting up the Database* [Bug 1356] Added subsection *Corner Cases of Name Strings* to Section 6.8 *Name Strings* [Bug 1453] Changed description of inclusive/exclusive flag in section 5.8 *Handling Attributes* [Bug 1477] Added info that Phythagorean theorem shall be used for calculation to Section 13.3 *References into 3D Objects Building Block* [Bug 1403] Added Section *Size Optimizations for NVC Trees* in *Name Building Block* [Bug 1406] Added Section *Reconstructing information from Leaf and Reference Nodes* to *Name Building Block* chapter. [Bug 1596] Changed link angle resolution to 5.6 [Bug 1489] Added subsections *Storage of Selection Graphs* and *Storage of Selection Criteria* to Section 6.3.2 and 6.3.3 of the *Name Building Block* chapter [Bug 551] Added Section 11.4 *Online Calculation of Coord Shift and Error Value for BDAM* to *Digital Terrain Model BB* chapter [Bug 1472] Added information that coins and bills can be used to Table B-10 Group of attributes attached to lanes 3 and 4 [Bug 1463] Updated Table B-6 *Lane Spearators* according to changes in DataScript

Version number	Published	Changes
1.9 (for NDS version 2.1.1.8)	2011-08-24	[Bug 1526] Added Section 8.3 *Calculation of Distances Between Locations* in *Traffic Information BB* chapter [Bug 1138] Enhanced note in Section 6.5.1 *Global Entry* [Bug 1685] Added Section B3 *Modelling Express Roads and Express Lines* to the Attribute Groups appendix. [Bug 1632] Added Section 3.8 *Angle Definitions* in *General Processes* chapter [Bug 1634] Changed max number of connected lanes to "16" in Section B.2 *Using Flexible Attributes for Lanes* [Bug 1665] Added information about `TILE_OF_CONTINUED_TURN_RESTRICTION` to subsection *Example 3: Aggregation of Link Attributes (Continued Turn Restriction* in *Routing BB* chapter [Bug 1685] Added use case for express lanes and roads in *Appendix A* [Bug 1671] Deleted `languageList` from Table 2-1 *Fields in product DB table* [Bug 1898] Added two explanatory figures to *Big Cities* section in *Name BB* chapter [Bug 1903] Enhanced documentation on toll attributes in *Appendix A* [Bug 1974] Changed file name NUL.txt to NUL.TXT in Section *File Structure of NDS Databases* [Bug 2068] Added section 3.1.1 *Tile Content Index* [Bug 1653] Updated *List of Indices* appendix [Bug 2031] Deleted recommendation in *Name Strings* section as it is out of scope for NDS [Bug 1544] Added information that ZIPVFS is used for NDS database compression in Section 2.5 *Compression*; deleted information about zlib Changed Heading of Appendix B to *Using Flexible Attributes*
2.0 (for NDS versions 2.1.1.9 and 2.1.1.10.1 - 2.1.1.10.6)	2012-01-09	[Bug 1806] Added subsection *Assigning Language Codes* to Section 6.8 *Name Strings* [Bug 2037] Made it more clear in Section *Looking up TMC Location Information* that both tables are required for decoding TMC messages. [Bug 2216] Deleted section B.11 *Modelling Fees between Toll Gates* as content is already contained in the new Toll Section (B.5 *Modelling Toll Systems*) Changed section 5.3 Splitting Road Features according to results from Docu Task Force meeting in Hildesheim, Augsut 4, 2011 (see meeting minutes) [Bug 2068] Reworked Tile Content Index section in General Processes chapter: Added mandatory and optional building blocks and clarified use

Version number	Published	Changes
2.0 ctd.		[Bug 2039] Added definition of T-junction to Section 13.7 *Avoiding T-Junctions* in *3D Objects Building Block* chapter [Bug 1712] Reworked Signpost Attributes section in Attribute Groups Appendix and added new signpost attributes [Bug 2051] Changed `locSubType` to `locSubtype` in Appendix *List of Indices* [Bug 1940] Enhanced description of `creationDateTime` in Table 2-3 *Fields in the version table* [Bug 1793] Table 2-8 *Pragma Commands*: Added rows `compress` and `uncompress` [Bug 1797] Table 2-4 *Fields in the update region table*: Removed row `urBoundingBox` [Bug 2200] Deleted Section 3.7 Sorting Order for Signed and Unsigned Integers [Bug 2039] Added explanation for T-junctions in Section 13-7 *Avoiding T-Junctions* [Bug 2037] Changed sentence in Section *Looking up TMC Location Information* to make clear that both tables are required as entry points [Bug 1975] Changed sentence in Section 2.4 *Multiplexing/ Stripping for SQL Files* to clarify on file extension for multiplexed db files. [Bug 1979] Changed entry *Possible settings for NDS* for `page_size` in the table *PRAGMA commands for compiling an NDS-compliant database* to correct inconsistency. [Bug 2032] Changed “reduce disambiguity” to “reduce ambiguity” in Section 6.8 *Name Strings*. [Bug 2038] Added link to Wikipedia for “texel” in Section 11.2 *Storing Height Map Data*. [Bug 2035] Added cross-reference in Section 7.5.1 *Defining the Overlap Section*. [Bug 2042] Reworked lists in *Example: Traffic Junction with Signposts* in Section 17.4. [Bug 2204] Added note in section 5.8.2 *Aggregated Links* stating that two links with different functional road classes shall not be aggregated [Bug 2299] Changed order of PK columns for `nvcTable` in *List of Indices* Appendix [Bug 2307] Added data type DateTime in Table 3-7 *Mapping of DataScript types to SQLite types* [Bug 2282] Added use case for disputed borders in Attribute appendix [Bug 1977] Added responsible party to `updateRegionID` in section *Product Database Structure*

Version number	Published	Changes
2.0 ctd.		[Bug 2040] Rephrased sentence on installation of POI database in section 15.2 *Relation between Full-text Search Data and POI Data* [Bug 1986] Section 6.7 *Speech Data* removed [Bug 2041] Replaced all occurrences of 3926 with 3826 in section 17.4 *Using Flexible Attributes for Signposts* [Bug 2230] Added section 3.2 *IDs According to Morton Order* [Bug 1985] Replaced paragraphs on spatial ordering in section 6.6.1 *Handling Objects and Object IDs* with reference to section 3.2 IDs According to Morton Order [Bug 2036] Rephrased sentence in section on *Correcting the Distance between Shape Points* [Bug 1563] Added figures *Attribute maps without grouping* and *Attribute maps with grouping* to section *Flexible Attribute Mechanism*. [Bug 1939] Rephrased section *Generating Tile IDs* and updated figure. [Bug 1996] Renamed section *Administrative Area Named Object Reference* to *Region Named Object Reference* in *Name Building Block* chapter and changed the content and the name of the flexible attribute accordingly [Bug 2331] Sentence added in About this Document chapter that examples used in the Specification documents are not binding. [Bug 1829] Added paragraph on generic standard POI category `POICAT_NDSGENERAL` to section 9.5 POI Standard Categories. [Bug 2077] Moved section *Change Case Marker* to NDS Format Specification. [Bug 1761] Added use case for roundabouts with middle traversal to Chapter 16 *Using Flexible Attributes and Attribute Groups* [Bug 2313] Described composed POI region ID: Changed sections about filling of metadata for integrated and non-integrated POI building blocks and section *Country and State Code References* in *POI Building Block* chapter [Bug 2303] Added a subsection for warning signs for speed limits in *Using Flexible Attributes and Attribute Groups* chapter [Bug 2378] Deleted `#define SQLITE_MULTIPLEX_MAX_CHUNKS 12` from *Size of File Chunks in Multiplexed Databases* subsection in *Setting up the Database* chapter [Bug 2428] Added sentence to *Height Map Tile Dimensions* section that interpolation is done at runtime and, thus, out of scope for NDS

Version number	Published	Changes
2.0 ctd.		[Bug 2313] Added description for composed region ID for POIs in Section *Filling the Metadata of Integrated POI Building Blocks* and Section *Country and State Code References* [Bug 2033] Reworked sections on BMD feature generalization in Format and Compiler Interoperability Specifications [Bug 1979] Corrected information on SQLite PRAGMA command `page_size` in Chapter 2 *Setting up the Database*

1.2 Purpose and Subjects

This document serves as a guideline for software engineers and software developers who design and implement compilers for navigation databases compliant to the Navigation Data Standard (NDS).

Data compilers have the task to *convert* input data to an NDS-compliant storage format. Compilers do not only have to produce product data in the right format, but must also fulfill additional requirements regarding content, accuracy, and correctness. This is to ensure that NDS applications are able to efficiently use the generated *NDS database*.

To this end, this document describes the rules that shall be followed to compile interoperable NDS databases, and gives recommendations for designing compiler processes where appropriate. This document does not contain best practices for compiling high-performance NDS databases, as this is the sole responsibility of each compiling company. The main task of this specification is to identify those issues that affect interoperability and to provide rules and recommendations for solving these issues.

Some of the parameter settings needed for the compiler processes may vary due to country specifics, properties of input data, customer requirements or other reasons. Most of these parameters are also stored as metadata of the NDS database in order to make them available for NDS applications. For this reason, complete and correct metadata is of crucial importance for the interpretation of NDS map data.

Examples used in the NDS Specification documents are intended to improve the comprehensibility of the respective topic and must not be seen as binding.

This document deals with the following subjects:

- Chapter 2 *Setting up the Database* on page 31. This chapter describes the physical structure of NDS databases, meaning the number and location of database files and the structure of the database tables. It also lists the SQLite parameters that can be set for NDS databases.
- Chapter 3 *General Processes* on page 45. This chapter describes general rules for the compilation of input data, which apply to all building blocks.
- Chapter 4 *Encryption* on page 63. This chapter describes compiler interoperability issues for the encryption of NDS databases.
- Chapter 5 *Routing Building Block* on page 71. This chapter describes compiler interoperability issues for the Routing building block.

- Chapter 6 *Eco Routing* on page 93. This chapter contains the information, which is required to calculate the eco routing values at compile time and run time.
- Chapter 7 *Name Building Block* on page 129. This chapter describes compiler interoperability issues for the Name building block.
- Chapter 8 *Basic Map Display Building Block* on page 149. This chapter describes compiler interoperability issues for the Basic Map Display building block.
- Chapter 9 *Traffic Information Building Block* on page 169. This chapter describes how flexible attributes and global TMC lookup tables are used to reference NDS features to TMC locations and how TMC events are used.
- Chapter 10 *POI Building Block* on page 173. This chapter describes compiler interoperability issues for the POI building block.
- Chapter 11 *Speech Building Block* on page 185. This chapter describes compiler interoperability issues for the Speech building block.
- Chapter 12 *Digital Terrain Model Building Block* on page 187. This chapter describes compiler interoperability issues for the Height Map representation of the Digital Terrain Model.
- Chapter 13 *Orthoimages Building Block* on page 191. This chapter describes compiler interoperability issues for the Orthoimages building block.
- Chapter 14 *3D Objects Building Block* on page 195. This chapter describes compiler interoperability issues for the 3D building block.
- Chapter 15 *Junction Views* on page 203. This chapter describes compiler interoperability issues for integrating junction views into an NDS database.
- Chapter 16 *Full-text Search Building Block* on page 207. This chapter describes compiler interoperability issues for the Full-text search building block.
- Chapter 17 *Using Flexible Attributes and Attribute Groups* on page 213 collects some typical use cases for navigation systems and shows how to cover these cases by using the NDS attribute mechanisms.
- Appendix A *Glossary* on page 247 explains the most important terms and key concepts for Navigation Data Standard.
- Appendix B *List of Indices for NDS Relational Tables* on page 249 lists the relational tables available in NDS and their indices.

1.3 Target Audience

This document is provided for IT professionals who want to get familiar with interoperability issues for compilers for Navigation Data Standard. This can be, for example, software developers developing compilers for NDS databases, product managers specifying navigation applications or software engineers developing such applications.

The following knowledge is assumed:

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

Provision	Verbal form	Allowed alternative expressions
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
	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 Setting up the Database

An NDS database is a set of data complying with the Navigation Data Standard (NDS) format that is stored in a particular navigation system supporting NDS. An NDS database may consist of one or more *product databases*. A product database may contain the navigation data for one or more *update regions*.

Note	NDS database files shall be stored in a format that can be processed by SQLite 3.7.4-nds3. Due to the use of FAT file systems, NDS database files shall not exceed 2 GB.

NDS databases are stored on *data carriers*. Generally, a data carrier is a physical medium to store digital data persistently, such as a DVD, a solid state memory, or a hard disk. In the context of NDS, a data carrier is a medium for storing data complying with the NDS format. It can be installed permanently in an NDS-compliant system, for example on a hard disk, or it can be used to transport data to an NDS-compliant system, for example using a DVD.

This chapter describes the file and table structure of NDS databases. It also provides information on the SQLite PRAGMA commands that may be used to modify the structure of the database files and the behavior of the SQLite library. This chapter also contains information about compression of NDS databases.

2.1 Distribution of Tables over SQLite Files

The current *NDS – Physical Model Description* specifies one SQLite database schema that includes all data tables and metadata tables. This does not contradict the approach described in this section, because the baseline database schema is to be applied as a template for creating SQLite files. It is mandatory to instantiate all tables for all building blocks that are present in an NDS database. It is not mandatory to fill all tables with data.

This approach provides a well-defined way for an application to access an NDS database and allows system suppliers and NDS database suppliers to individually organize their data carriers. They may, for example:

- Use a hierarchical or a flat file structure
- Distribute the data to multiple SQLite files or store it in a single SQLite file
- Install additional product databases at a later point in time

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, for example the `ColorTable` between Basic Map Display and 3D Objects building block. The SQLite file in which a table is stored is specified in the `uri` field of the `BBlockCompVersionTable`.

Note For a given update region and building block using a `ColorTable`, the `BBlockCompVersionTable` uniquely specifies where the `ColorTable` is found:

If the `uri` field of `BBlockCompVersionTable` for the `ColorTable` entry is `NULL`, the `ColorTable` is located in the same SQLite file as the other tables of the respective building block. Otherwise, the `uri` field specifies the location of the `ColorTable`.

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.

See also *NDS – Update Specification*, 5.3 *NDS Version Tree* on page 41.

Example 1

In the following example, an NDS database is defined that consists of one product, one update region (UR), and a common SQLite file for the Basic Map Display, the Routing and the Name building blocks. Absolute URIs are used.

```
ROOT.NDS
productDbTable
product1 -> file:///ROOT.NDS
UrBuildingBlockVersionTable
UR1, BMD -> file:///ROOT.NDS
UR1, Routing -> file:///ROOT.NDS
UR1, Name -> file:///ROOT.NDS
BBlockCompVersionTable
UR1, Routing, GlobalGatewayTable -> <empty uri>
[… more tables here …]
```

Example 2

In the following example, an NDS database is defined that consists of one product in a subdirectory, two update regions (UR), each in a different subdirectory, and separate SQLite files for the Basic Map Display, the Routing and the Name building blocks. Relative URIs are used.

```
ROOT.NDS
productDbTable
product1 -> PRODUCT1/PRODUCT.NDS
UrBuildingBlockVersionTable
UR1, BMD -> PRODUCT1/UR1/BMD.NDS
UR1, Routing -> PRODUCT1/UR1/ROUTE.NDS
UR1, Name -> PRODUCT1/UR1/NAME.NDS
UR2, BMD -> PRODUCT1/UR2/BMD.NDS
UR2, Routing -> PRODUCT1/UR2/ROUTE.NDS
UR2, Name -> PRODUCT1/UR2/NAME.NDS
BBlockCompVersionTable
UR1, Routing, GlobalGatewayTable -> PRODUCT1/PRODUCT.NDS
UR2, Routing, GlobalGatewayTable -> PRODUCT1/PRODUCT.NDS
 [… more tables here …]
```

Note In example 2, the `GlobalGatewayTable` for both Routing building blocks in UR1 and UR2 is located in the file `PRODUCT.NDS` of product 1.

Related Topics:

- *NDS – Format Specification*, 3.2 *NDS Database Structure* on page 52

2.2 SQLite Parameters

SQLite uses PRAGMA commands to modify the structure of the database files and the behavior of the SQLite library. For NDS, the persistent PRAGMA commands influencing the data storage are especially important. PRAGMA commands that are set temporarily and modify the behavior of the SQLite library are not in the focus of NDS, as these commands are set by the applications working with NDS-compliant databases. For a detailed list of all available PRAGMA commands, refer to the SQLite website (*http://www.sqlite.org/pragma.html*).

Note The values of the persistent PRAGMA commands are stored directly in the database file. If a database compiler is not to use the default values, it has to change the PRAGMA values before writing data to the database. Once the first table in the database has been created, the persistent PRAGMA commands cannot be changed any longer.

When the SQLite library opens an SQLite file, it first reads the persistent PRAGMA values from the first page of the database file. Based on these settings, the library is able to read the database.

Table 2-1 lists the PRAGMA commands that are relevant for compiling an NDS-compliant database.

Table 2-1 PRAGMA commands for compiling an NDS-compliant database

Command	Description	Possible settings for NDS
`auto_vacuum`	Standard SQLite PRAGMA command Controls reclaiming of unused database space created by deletion of data.	`0` (off) 2 (incremental)
`encoding`	Standard SQLite PRAGMA command Sets the character encoding of the database.	UTF-8
`page_size`	Standard SQLite PRAGMA command Sets the page size of the database.	Any value in bytes that is a power of two greater than or equal to 512 Byte, and less than or equal to 32768 Byte
`default_cache_size`	Standard SQLite PRAGMA command Sets a default fixed cache size for the maximum number of database disk pages that SQLite will hold in memory simultaneously.	Use of this command is forbidden
`compr_type`	NDS-specific PRAGMA command Sets the compression type of the database.	None (default) `zlib`
`compress`	NDS-specific PRGAMA command Compresses database using method defined in `compr_type`	Integer [1..9] 6 (default)
`uncompress`	NDS-specific PRAGMA command. Decompresses database using method defined in `compr_type`	None
`pct_free`	NDS-specific PRAGMA command Defines how much space has to be left free at the end of each compressed page (in percent). For more information, refer to Section 2.4 *Compression* on page 37.	Integer [0, ..., 100] Default: `10`

Note This document only contains information that is relevant for NDS databases. For detailed information on the standard SQLite PRAGMA commands, refer to *http://www.sqlite.org/pragma.html*.

Parameter `page_size`

NDS databases may have different page sizes. Page sizes may even vary between the individual database files of the same NDS database, provided that the page size is a power of two greater than or equal to 512 Byte, and less than or equal to 32768 Byte.

NDS databases supplied on rotating media, such as hard disks or DVDs, should use a relatively large page size, whereas NDS databases supplied on flash media should use a smaller page size. Applications working with NDS databases must be able to process different page sizes. If a standard SQLite library is used by such an application, this library provides for correct processing of database files with different page sizes.

Note The total memory consumption of the SQLite library is not only determined by the page size, but also by the `cache_size` PRAGMA command. This command is a session parameter, and its value can be set by the application using the database. The total memory consumption equals the product of the `page_size` and `cache_size` parameter values.

Example `page_size`: 1024 byte

`cache_size`: 2000

Total memory consumption: 1024 x 2000 = 2048000 byte (about 2 Mbyte)

Parameter `default_cache_size`

Compilers for NDS databases must not use this parameter, meaning it is not allowed to define a default fixed cache size. The cache size is set by the application by means of the temporary PRAGMA command `cache_size`.

Parameter `compr_type`

NDS databases may be compressed using the zlib compression library. Each SQLite page is compressed individually. For detailed information about compression, see Section 2.4 *Compression* on page 37.

Parameter `pct_free`

This parameter controls how much space is left free at the end of each compressed database page. This space can be used during updates, for example, for inserting new records. If the available space is not sufficient, compressed pages must be moved in the SQLite file after an update. The higher the value of the `pct_free` parameter and thus the percentage of free space, the more likely it is that updated pages do not need to be moved. On the other hand, the size of the original database increases accordingly, as additional space is added at the end of each page.

2.3 Multiplexing/Striping for SQLite Files

The maximum size of NDS database files is restricted to 2 GB. This size restriction is due to a common limit of 32-bit architectures where files have to fit in 2^31 byte, the equivalent to 2 GB. In navigation databases, however, the data of some building blocks, such as the 3D Objects or Orthoimages building blocks, may exceed this limit. The database files for these building blocks must, therefore, be split into several files.

For splitting database files into several file chunks, NDS uses the multiplexing shim solution, which is available as of SQLite version 3.7.4-nds3. SQLite uses shims to optimize the hardware access without affecting the logic of database tables or the external API.

SQLite's multiplexing shim enforces a multiplexing of the database files by partitioning a single database file into several chunk files. In this way, the overall database file size can exceed the file size limit of the underlying file system. The chunk files are binary compatible to classic monolithic database files. The files are split without changing the content. Thus, it is also possible to create a valid multiplexed database by splitting and renaming a big database file manually. This also works vice versa: If the underlying operating system allows file sizes of more than 2 GB, a multiplexed database can be easily converted to a monolithic database by concatenating the chunks together.

Multiplexing only affects database files exceeding the 2 GB size limit. As the database file itself is not different to database files without multiplexing, smaller database files can be read with or without the multiplexing shim.

Use of the multiplexing shim must be specified at compile time by means of parameters defining the chunk size and the naming convention for the chunk files. It is not possible to switch multiplexing on or off during runtime. Also, the database metadata does not contain information about the multiplexing parameters. For this reason, the multiplexing parameters must also be set in the SQLite programs and drivers using the compiled database. Thus, these applications are restricted to work with databases, which are created with identical multiplexing parameters.

The size of the chunk is fixed to 2 GB (maximum file size). For a page size with a maximum size of 64 Bytes, the chunk size is calculated as follows: 2097152 – 64 = 2097088 KB.

Multiplexed databases are addressed by the name of the first chunk file, which keeps the name of the original database file. The other chunk files are numbered in the file name extension according to the naming convention defined in the parameters. The file name extension is uses as a three digit counter (`.001`, `.002`, ...); the regular extension `.NDS` is not used in this case. As the multiplexing shim handles these additional files, the application does not need to know these names.

Example Name of the original SQLite file (first chunk): `UPDATERG.NDS`

Name of the second chunk: `UPDATERG.001`

Name of the third chunk: `UPDATERG.002`

Size of File Chunks in Multiplexed Databases

The SQLite multiplexing shim solution limits the size of file chunks to 2 GB minus 1. The operating systems, in which the database is used, must support database files with a maximum size of 2 GB. In order to achieve the best possible performance, the size of the file chunks should be set close to the limit set by SQLite. Also, the chunk size should be a multiple of the defined max. page size of 64 kB for NDS databases.

Based on these parameters, the optimal size of the file chunks is calculated as follows: 2 GB – 64 kB = 2147418112 Bytes.

To ensure that the size of file chunks is the same in all NDS databases, the following `define` statements have to be made during compilation:

- `#define SQLITE_MULTIPLEX_EXT_OVWR`
- `#define SQLITE_MULTIPLEX_EXT_FMT    "%03d"`
- `#define SQLITE_MULTIPLEX_EXT_SZ     3`
- `#define SQLITE_MULTIPLEX_CHUNK_SIZE 2147418112`

2.4 Compression

Compression may be applied to reduce the storage space required for NDS databases.

NDS databases are compressed by means of the ZIPVFS module of SQLite. For detailed information, see the documentation of the respective ZIPVFS module version. The corresponding compression algorithm, which shall be used for compression of NDS databases, is defined in the NDS SQLite Header.

Note	The concept of ZIPVFS compression for NDS is still under discussion. More detailed information will be available in a later version of the NDS Specification documents.

2.4.1 Page Level Compression

SQLite reads and writes data to pages of a certain size.

When page level compression is enabled, a compression or decompression module compresses or decompresses each page that is read or written before the page is further processed by the higher-level modules of the SQLite library. Compression is handled by the low-level SQLite modules, that is the operating system layer. The higher-level modules, such as the B-tree, the virtual machine, and the parser module, deal with decompressed pages. This enables applications to access the data using standard SQL.

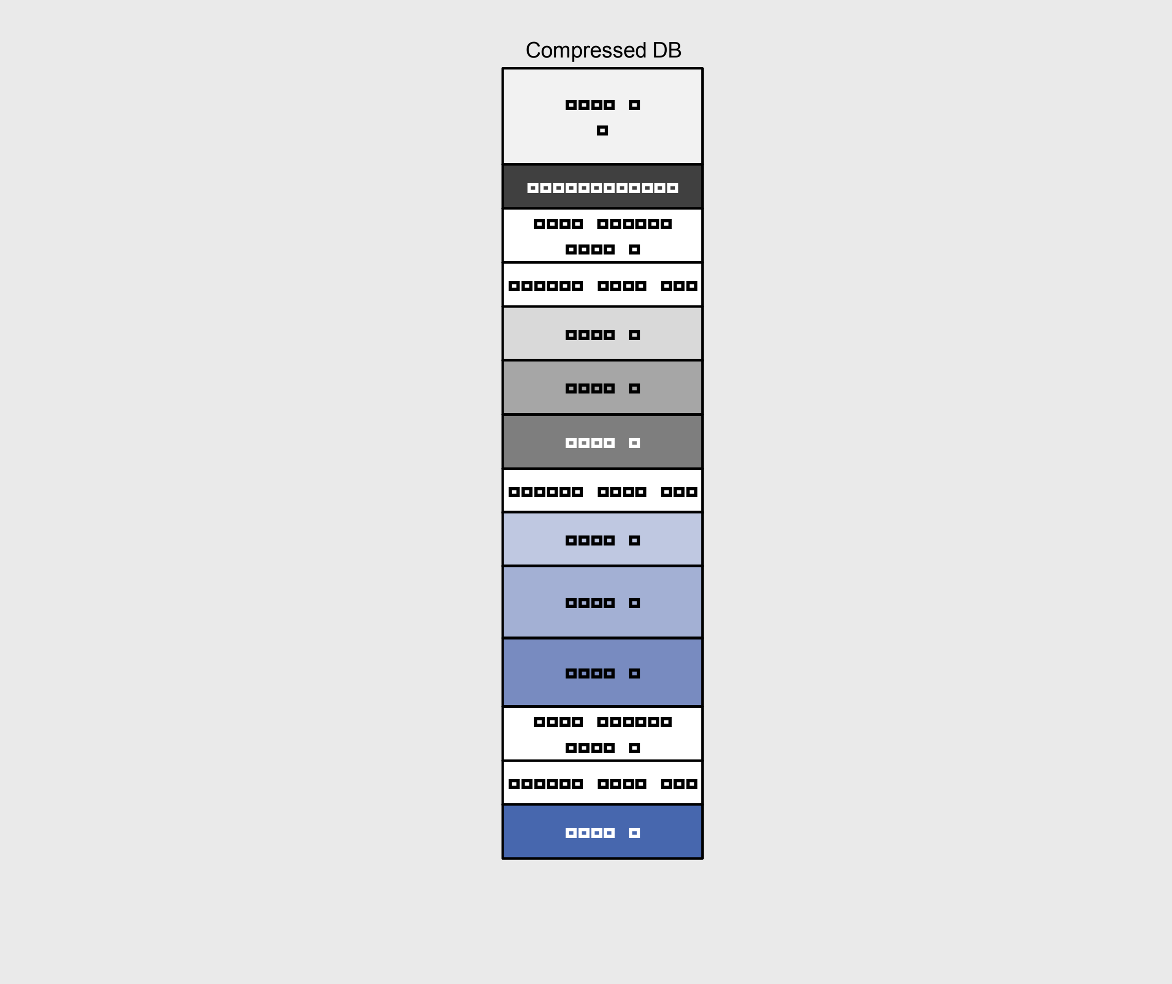
In an uncompressed SQLite file, the starting point of a page within the database file can always be computed directly by multiplying the page number with the page size.

In a compressed SQLite file, this method is not applicable. In this case, page numbers need to be mapped explicitly to their starting points in the database file. The mapping is stored in the SQLite database itself in the so called offset-list.

Structure of Compressed Databases

Figure 2-1 shows the structure of a compressed NDS database.

Figure 2-1 Structure of a compressed NDS database



The compression header follows page 1 of the SQLite database. The structure of the compression header is described in Table 2-2.

Table 2-2 Structure of the compression header

Byte range	Byte size	Descript
1..16	16	The first 16 bytes of the compression header contain the UTF-8 encoding of the string „SQLite3 for NDS“, followed by a single null-terminator character.

Byte range	Byte size	Descript
17..20	4	Number of compressed pages
21..24	4	Number of offset lists of one main offset list
25..28	4	Number of page offsets and free space offsets of one offset list
...29	1	Compression method: – 0 = none – 1 = zlib – 2 = lzma – 3 = lzo1f

The main offset list follows the compression header. It contains the address of the next main offset list, and a predefined number of addresses to offset lists. The predefined number is stored in the compression header of the SQLite database.

This list is used to reduce the number of read operations required to find a compressed page. Table 2-3 shows the structure of the main offset list.

Table 2-3 Main offset list in a compressed NDS database

Byte range	Byte size	Description
1..8	8	Offset to next main offset list For the last offset list, the offset is `0`.
9..16	8	Offset to first offset list
16..24	8	Offset to second offset list
x...x+7	8	Offset to nth offset list

Note In the example depicted in Figure 2-1, the number of offset lists contained in one main offset list, and the number of page offsets in one offset list is rather low. This can increase the number of seek operations required to reach a page.

Values between, for example, 512 and 8192 are recommended to improve performance.

The offset list contains a predefined number of addresses and sizes of compressed pages, and an equal number of addresses and sizes of free space. The predefined number is stored in the compression header of the SQLite database. This list contains the actual offsets and sizes of the compressed pages.

Offset and page size are defined as follows:

- Offset 48 bits, size of 16 bits for a page size of 32 KB or smaller
- Offset 47 bits, size of 17 bits for a page size of 64 KB

The free space part of the offset list starts at: `(8 * <number of page offsets>)`.

Table 2-4 and Table 2-5 show the structure of page offsets and their respective sizes.

Table 2-4 Offset list: Page offset and sizes

Byte range	Byte size	Description
1..8	8	Offset to and size of page k
9..16	8	Offset to and size of page l
17..24	8	Offset to and size of page m
x..x+7	8	Offset to and size of page n

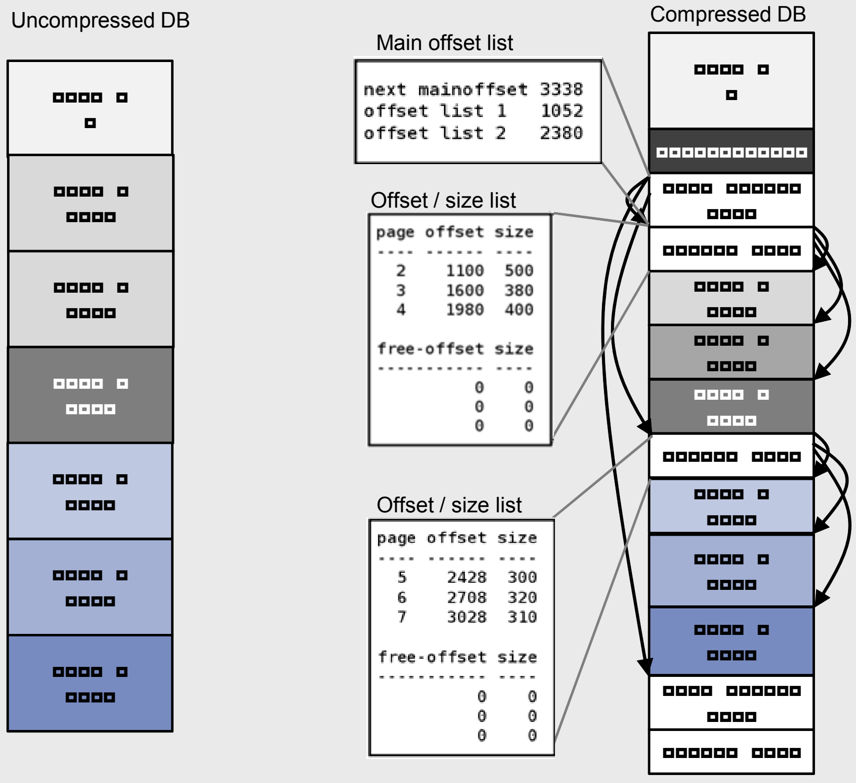
Table 2-5 Offset list: Free space offsets and sizes

Byte range	Byte size	Description
1..8	8	Space o: Offset and size of free space
9..16	8	Space p: Offset and size of free space
17..24	8	Space q: Offset and size of free space
x..x+7	8	Space r: Offset and size of free space

Handling of Page Updates

The following figures describe the handling of page updates for compressed NDS databases. Figure 2-2 shows the compressed database before the first update.

Figure 2-2 Handling of page updates – Starting point



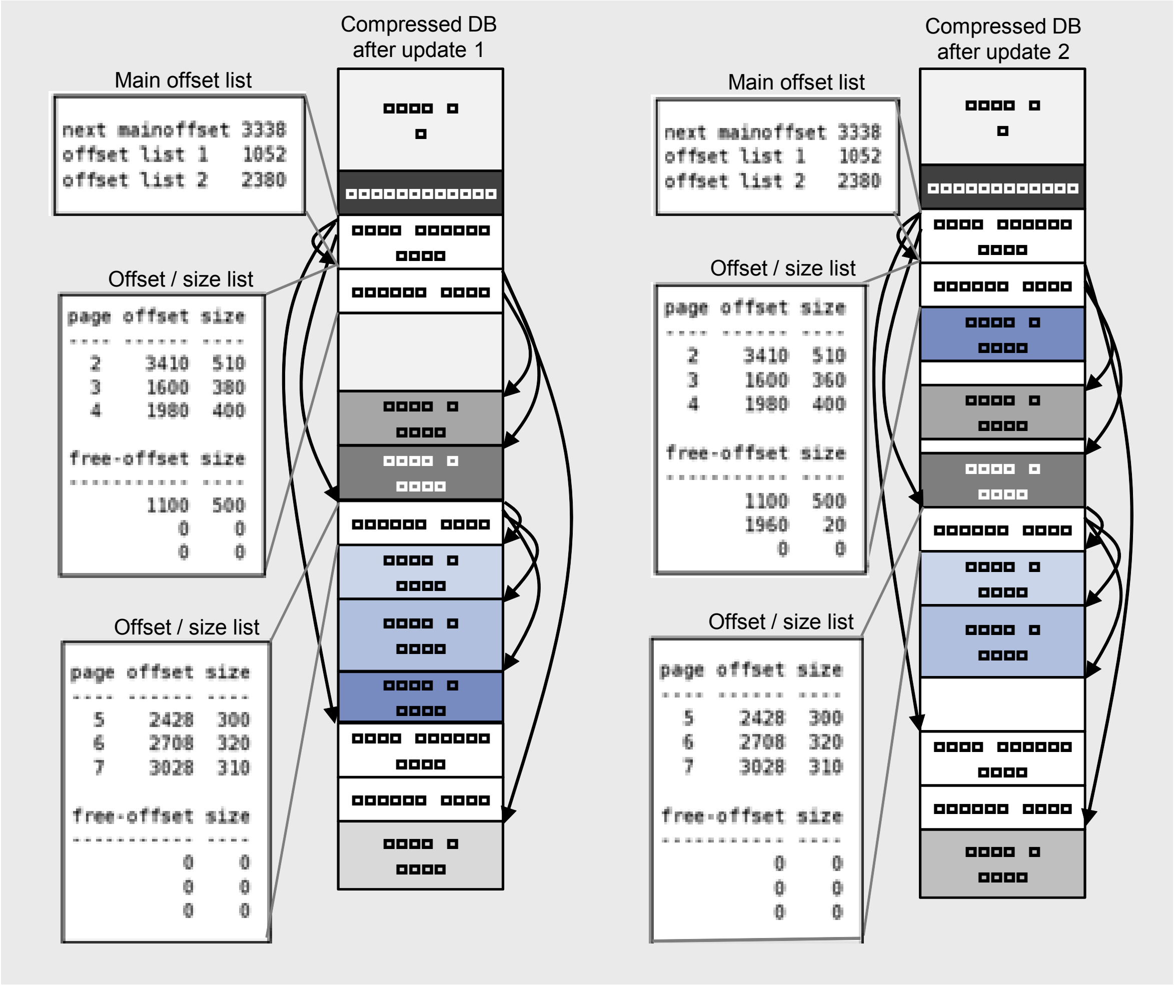
```
next mainoffset 3338
offset list 1   1052
offset list 2   2380
```

```
page offset size
---- ------ ----
  2    1100  500
  3    1600  380
  4    1980  400

free-offset size
----------- ----
          0    0
          0    0
          0    0
```

```
page offset size
---- ------ ----
  5    2428  300
  6    2708  320
  7    3028  310

free-offset size
----------- ----
          0    0
          0    0
          0    0
```

In the update scenario in Figure 2-3, the size of page 3 shrinks during the second update from 380 to 360 bytes. The page is written to its original location, and the position (1960) and the free space (20) are written to the free space list.

Figure 2-3 Handling of page updates – Update scenario 1



```
next mainoffset 3338
offset list 1   1052
offset list 2   2380
```

```
page offset size
---- ------ ----
  2    3410  510
  3    1600  380
  4    1980  400

free-offset size
----------- ----
       1100  500
          0    0
          0    0
```

```
page offset size
---- ------ ----
  5    2428  300
  6    2708  320
  7    3028  310

free-offset size
----------- ----
          0    0
          0    0
          0    0
```

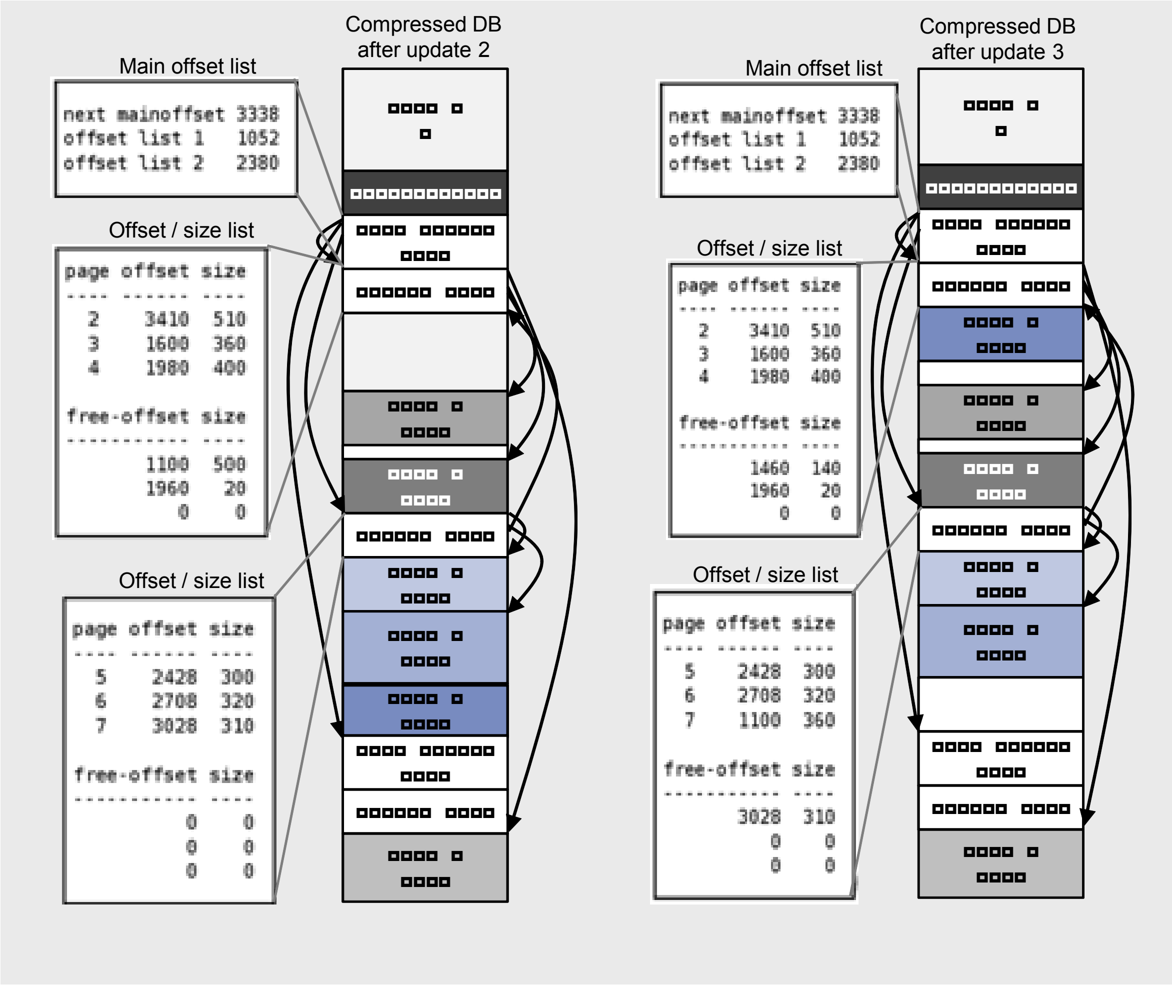
```
next mainoffset 3338
offset list 1   1052
offset list 2   2380
```

```
page offset size
---- ------ ----
  2    3410  510
  3    1600  360
  4    1980  400

free-offset size
----------- ----
       1100  500
       1960   20
          0    0
```

```
page offset size
---- ------ ----
  5    2428  300
  6    2708  320
  7    3028  310

free-offset size
----------- ----
          0    0
          0    0
          0    0
```

During the next update (depicted in Figure 2-4), the size of page 7 grows from 310 to 360 bytes. The page is now too large for its original location and is, therefore, written to a new location with enough space (position 1100). The free space entry for position 1100 with size 500 is changed to 1460 with size 140. The original position of page 7 (3028) and the size (310) is written to the free space list.

Figure 2-4 Handling of page updates – Update scenario 2



```
next mainoffset 3338
offset list 1   1052
offset list 2   2380
```

```
page offset size
---- ------ ----
  2    3410  510
  3    1600  360
  4    1980  400

free-offset size
----------- ----
       1100  500
       1960   20
          0    0
```

```
page offset size
---- ------ ----
  5    2428  300
  6    2708  320
  7    3028  310

free-offset size
----------- ----
          0    0
          0    0
          0    0
```

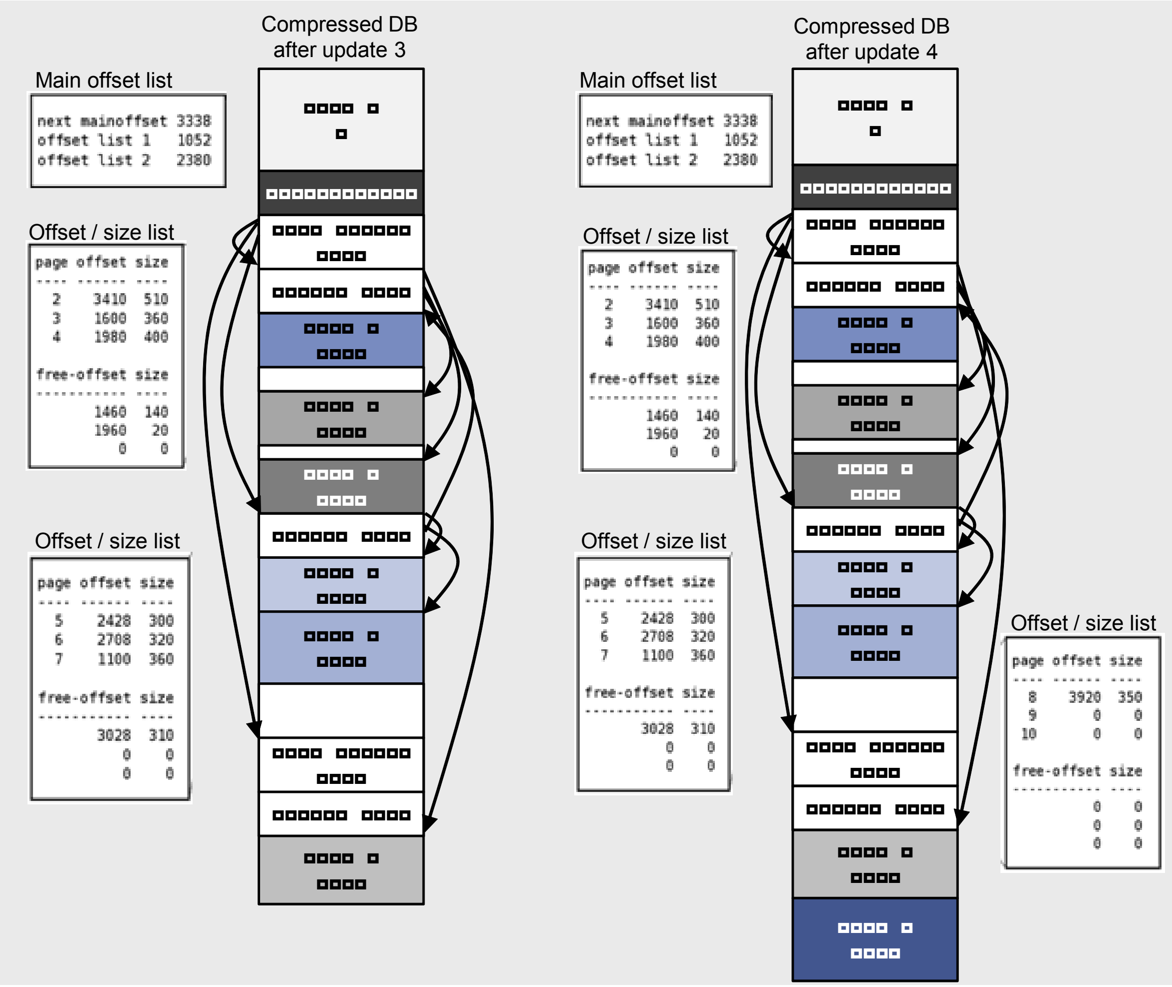
```
next mainoffset 3338
offset list 1   1052
offset list 2   2380
```

```
page offset size
---- ------ ----
  2    3410  510
  3    1600  360
  4    1980  400

free-offset size
----------- ----
       1460  140
       1960   20
          0    0
```

```
page offset size
---- ------ ----
  5    2428  300
  6    2708  320
  7    1100  360

free-offset size
----------- ----
       3028  310
          0    0
          0    0
```

In the next update, a new page 8 is added to the database. This is depicted in Figure 2-5. As the free space list shows that there is not enough space for page 8, the page is added at the end of the database at position 3920.

Figure 2-5 Handling of page updates – Update scenario 3



```
next mainoffset 3338
offset list 1   1052
offset list 2   2380
```

```
page offset size
---- ------ ----
  2    3410  510
  3    1600  360
  4    1980  400

free-offset size
----------- ----
       1460  140
       1960   20
          0    0
```

```
page offset size
---- ------ ----
  5    2428  300
  6    2708  320
  7    1100  360

free-offset size
----------- ----
       3028  310
          0    0
          0    0
```

```
next mainoffset 3338
offset list 1   1052
offset list 2   2380
```

```
page offset size
---- ------ ----
  2    3410  510
  3    1600  360
  4    1980  400

free-offset size
----------- ----
       1460  140
       1960   20
          0    0
```

```
page offset size
---- ------ ----
  5    2428  300
  6    2708  320
  7    1100  360

free-offset size
----------- ----
       3028  310
          0    0
          0    0
```

```
page offset size
---- ------ ----
  8    3920  350
  9       0    0
 10       0    0

free-offset size
----------- ----
          0    0
          0    0
          0    0
```

2.5 Data Carrier-specific Options

Note Interoperability issues related to data carrier-specific options will be added as soon as prototyping experiences are available.

3 General Processes

This chapter describes general rules and requirements for the compilation of input data, which apply to all building blocks.

General Process Flow

Level 13 is the base level with full detail information for routing and basic map display. All routing and basic map display features created from the input data must be assigned to exactly one of the tiles of level 13.

During the subsequent generalization processes, features can be simplified and/or aggregated and assigned to tiles on higher levels. Alternatively, they can simply be left out.

The transformations performed by the compiler can be interwoven and highly interdependent. To ensure correct processing results and adequate behavior of the applications using the generated NDS database, some of these processing steps follow a specific sequence. If a sequence of steps is given in this document, this sequence is to be considered as a working assumption, meaning that a real compiler may deviate from this sequence, provided that it produces the same results as the given processing sequence.

Structure Integrity

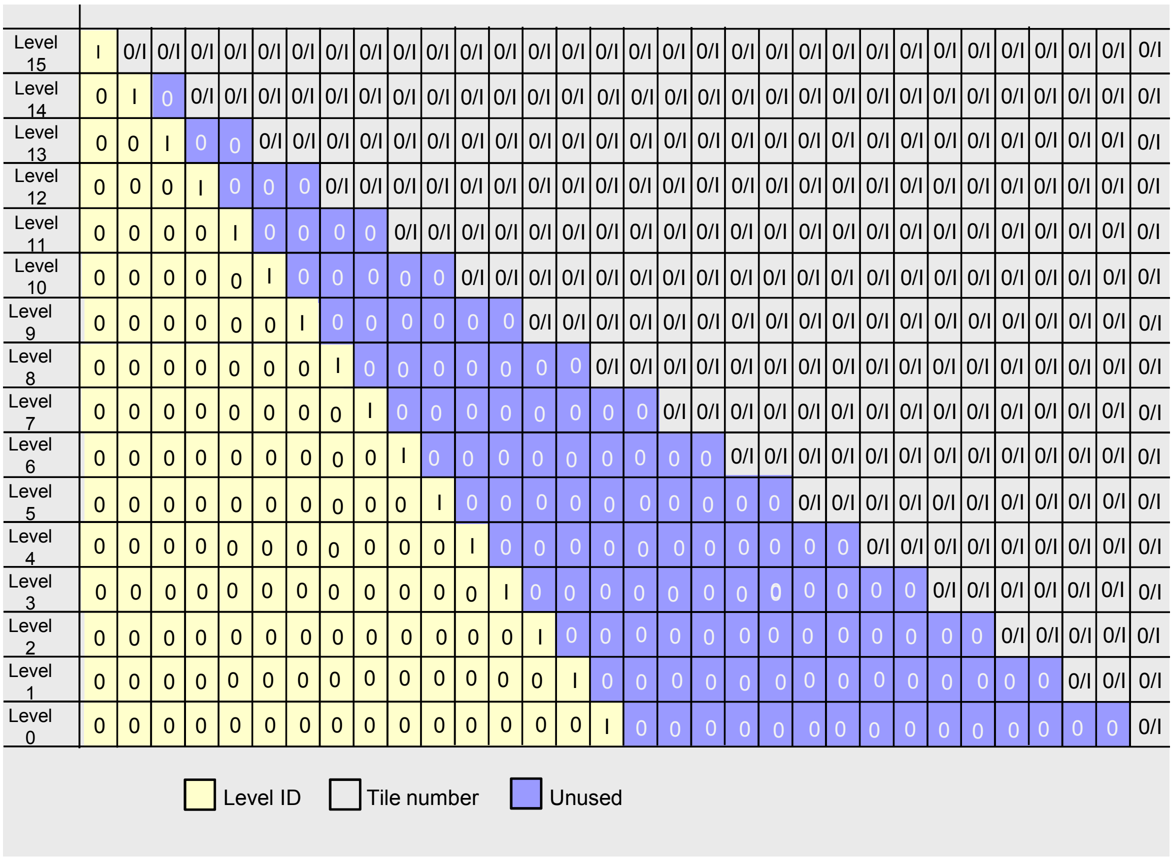
The compiler must ensure that the generated NDS database has reliable basic properties. That means, for example, that all references must point to existing features, also after an update. This also means that the NDS database must be compiled in a space-saving way. Empty lists, for example, shall not be stored if the entire list is an optional member and can be omitted.

3.1 Generating Tile IDs

The data in several building blocks is partitioned according to the NDS tiling scheme. The underlying tiling scheme is regular and hierarchical. Tiles are identified by a combination of level number (encoded as a level ID) and tile number as shown in Figure 3-1. The combination of level number and tile number constitutes the tile ID.

The number of tiles on level 0 is two and increases by the factor of 4 with each level. This is why the tile number can be encoded by one bit on level 0 and additional two bits for each lower level. So level 1 needs 3 bits while 31 bits are needed on level 15 for encoding the tile number. The remaining bits are used to encode the level. Encoding the level is necessary because the ranges of tile numbers overlap between levels.

Figure 3-1 Compact tile ID



Level 15	I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I
Level 14	0	I	0	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I
Level 13	0	0	I	0	0	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I
Level 12	0	0	0	I	0	0	0	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I
Level 11	0	0	0	0	I	0	0	0	0	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I
Level 10	0	0	0	0	0	I	0	0	0	0	0	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I
Level 9	0	0	0	0	0	0	I	0	0	0	0	0	0	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I
Level 8	0	0	0	0	0	0	0	I	0	0	0	0	0	0	0	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I
Level 7	0	0	0	0	0	0	0	0	I	0	0	0	0	0	0	0	0	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I
Level 6	0	0	0	0	0	0	0	0	0	I	0	0	0	0	0	0	0	0	0	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I
Level 5	0	0	0	0	0	0	0	0	0	0	I	0	0	0	0	0	0	0	0	0	0	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I
Level 4	0	0	0	0	0	0	0	0	0	0	0	I	0	0	0	0	0	0	0	0	0	0	0	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I	0/I
Level 3	0	0	0	0	0	0	0	0	0	0	0	0	I	0	0	0	0	0	0	0	0	0	0	0	0	0/I	0/I	0/I	0/I	0/I	0/I	0/I
Level 2	0	0	0	0	0	0	0	0	0	0	0	0	0	I	0	0	0	0	0	0	0	0	0	0	0	0	0	0/I	0/I	0/I	0/I	0/I
Level 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0/I	0/I	0/I
Level 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0/I

🟨 Level ID □ Tile number 🟦 Unused

The tile numbers are created according to Morton code. The origin of the coordinate system (longitude = 0, latitude = 0) is identical to the origin of the WGS 84 coordinate system.

The tile number on level 0 consists of one bit, which is set according to the most significant bit of the longitude of the south-west corner of a tile.

For each lower level, the tile number is extended by two bits. The tile number is built from the bit(s) of the parent tile number and the nth bit of the latitude and longitude of the south-west corner of the tile, with n = level number and bits counted from the most significant as bit 0. Thus, the tile number is identical to the (2n+1) most-significant bits of the Morton code of the south-west corner of the tile.

To generate the tile ID, combine tile number and level ID by a bitwise or-operation. The level ID for a referenced level n is encoded as 2^{16+n}, which is 65536 = 2^{16} for level 0.

Related Topics:

– *NDS – Format Specification*, 7.3 *Tiling Scheme* on page 100

3.1.1 Generating IDs According to Morton Order

To speed up access to specific data elements, it is necessary to generate IDs for these elements according to Morton order. Such elements are, for example: named objects, POIs, and POI service locations.

When generating IDs according to Morton order, consider the following:

- The generation process may have to take levels into account.
- The generation process shall generate IDs with 32 bits.
- The generation process has to consider updates to avoid avalanche effects.

Recommended procedure

1. Identify a reference point for each data element.

   For data elements with a geographic extent, typical reference points are either the center of the bounding box or the most western of the most southern points belonging to the data element.

2. Identify a level for each data element.

   Feature type or POI category can determine the level to use. It can also be the level in which the item is visible for the first time. For items with geographic extent, it can be sensible to use one of the next two upper levels if the data element is then contained completely in the corresponding tile. If no levelling is necessary, use level 13 as default for all data elements.

3. Attach a tuple to each data element that consists of:
   - the data element's level
   - the Morton code of the data element's reference point
4. Sort all data elements according to their tuples.
   1. Sort all data elements by level.
   2. Sort data elements with the same level by Morton code.
   3. Choose some arbitrary (but consistent) order for data elements with identical level and Morton code. For POIs, for example, this may be according to POI category value.
5. Define an ID k as start ID.

6. Choose a gap n between successive IDs (with, e.g., n = 10 or n = 16).

   The value of n determines how many new data elements can be inserted by update operations between two existing data elements, before IDs of existing data elements have to be modified.

7. Assign each data element in increasing order its 32-bit ID. Start from k using n as gap between successive IDs.

In addition to gaps between individual successive IDs, it can be necessary to use additional gaps between successive data elements on different tiles. It can be sensible to reserve some IDs for empty tiles. If levels play an important role, either larger gaps between levels or defined ID ranges for each level can be advisable. When selecting those parameters for the ID generation, take number and distribution of data elements into account to make sure that all generated IDs fit into the desired 32-bit range.

3.2 Tile Content Index

The `TileContentIndex` describes the availability of tiles in a building block in form of a matrix. In the matrix, the availability of a tile is indicated by the following values:

- 1 = set

  If value 1 is set for a matrix cell, the building block contains one or more tiles in the area described by the matrix cell in all levels lower or equal to the level defined by the south-west tile.

- 0 = not set

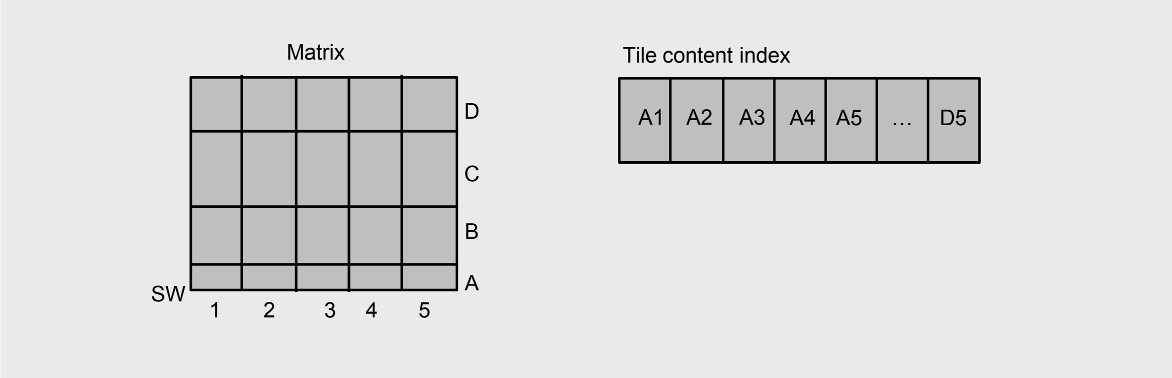
  If value 0 is set for a matrix cell, no tiles are available in the area described by the matrix cell for this building block.

The tile content index is optional.

The starting tile of the matrix is located in the south-west corner. The `deltaLatTileNumber` specifies the width of the matrix and the `deltaLongTileNumber` specifies the height. Thus, the matrix spans `deltaLatTileNumber * deltaLongTileNumber` tiles.

The ordering rule for the matrix is as follows: The ordering is per row, starting with (and including) the matrix‘s south-west tile. This is depicted in Figure 3-2.

Figure 3-2 Matrix for tile content index



It is possible to create a matrix for one or more levels. There should be at least one tile content index for level ≤ 13. Tile content indices for levels 14 or 15 can be added. NDS recommends to create a matrix for level 8, because this level gives the best trade off between matrix size and hit/miss ratio of the matrix. The hit/miss ratio is defined by the number of times an application has to look into this update region to find (or not find) a tile for this building block for a specific level.

There should be at least one level in the tile content index table covering one or more levels of the respective building block. If, for example, the building block only contains level 10, the tile content index table should contain the levels ≤ 10.

If an entire row and/or column is not set (value 0), a new `TileContentIndex` matrix can be created to avoid that the matrix contains unset rows or unset columns. An update region with an island, for example, contains one tile content index for Basic Map Display (ocean tiles) and two tile content indices for routing (if no ferry links are present).

3.3 Reducing Coordinate Precision

Coordinate precision determines the numeric precision with which spatial data is stored in NDS. The maximal coordinate resolution in NDS is defined as $360°/2^{32}$ which results in a size of about 0.9 cm per coordinate unit along the equator or the meridians.

To store coordinates with maximum resolution, 32-bit values are required. To save space, however, NDS allows a maximum of 16 bit for encoding coordinates within tiles. For this reason, coordinate precision must be reduced on all levels including level 13. This is done using the shift value parameter that is stored for the different levels as a global metadata item for all building blocks. For more information on global metadata, refer to *NDS – Format Specification*, 20.2 *Common Data* on page 352.

The shift value parameter is specified by the number of bits and must be used as follows: When an NDS application reads a coordinate value from the database, it shifts it to the left by the number of bits specified as the shift value. After this, the coordinate value is interpreted within the NDS coordinate system based on 360°/2^{32}. Accordingly, a compiler must shift full-resolution coordinate values to the right by the defined shift value before writing the coordinates to the database.

A shift value of 3 bit, for example, means that each number read from the database has to be shifted 3 bits to the left (that is, multiplied by 8). Thus the resolution is effectively reduced to one-eighth, which corresponds to approximately 7.2 cm per coordinate unit along the equator or the meridians. Using a shift value of 3 makes it possible to encode level 13 tiles within the allowed range of 16 bit.

This shift value, however, still produces a resolution that is higher than the common accuracy of the input data. It is, therefore, recommended to specify a shift value of 5 or 6 for level 13, which results in a resolution of about 30 or 60 cm per coordinate unit along the equator or the meridians.

The shift value is typically – but not necessarily – increased for each higher level. It is not allowed to decrease the value for a higher level. Considering the NDS tiling scheme, increasing the shift value for each higher level by 1 is a natural choice. This results in a coordinate resolution that decreases by factor 2 for each level.

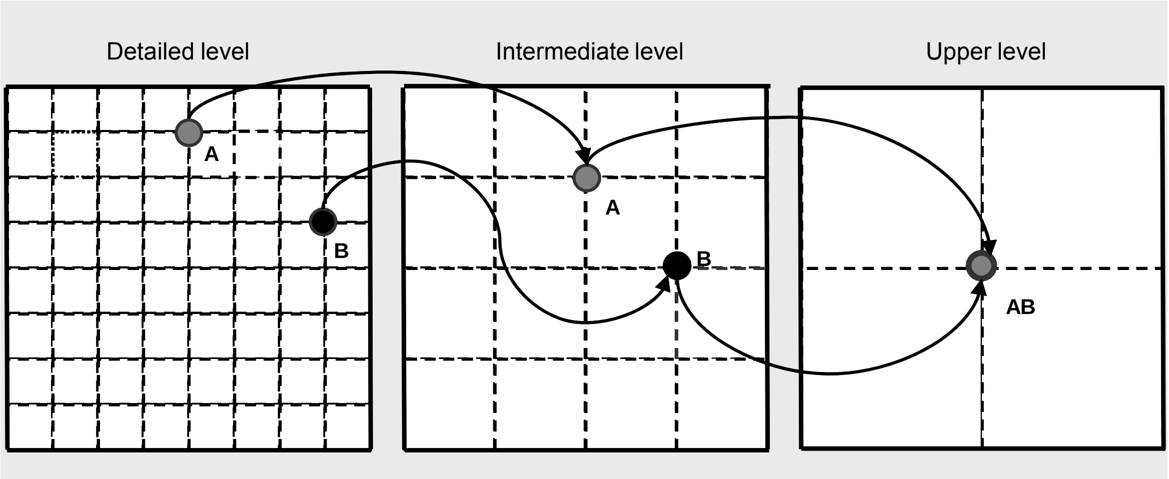
Changing the shift value results in different coordinate values for the same point on different levels, with the effect that the point "moves". The distance between two points calculated by means of the difference in coordinates can thus differ between the levels. A special case is that two different points are mapped to identical coordinate values. It is, for example, possible that (with a shift value increased by 1 per level) two points which are apart from each other by 160 meters on level 13 collapse to a single point on level 4. Compilers must consider these side effects of precision reduction and handle them accordingly. Handling can be different depending on the building block.

For more details, refer to the following information:

- *NDS – Format Specification*, 7.2 *Coding of Coordinates* on page 97
- Adjusting coordinates for routing features (see Section 5.7 *Adjusting Coordinates* on page 76)
- Adjusting coordinates for basic map display features (see Section *Adjustment* on page 153)

Figure 3-3 illustrates how points on different positions are moved to identical coordinates on higher levels.

Figure 3-3 Shifting of points due to coordinate precision reduction



The coordinates of the input data are adapted to the coordinate grid by truncation and not by mathematical rounding. If, for example, the shift value is progressively increased by 1 on each higher level, the least significant bit of a coordinate value is dropped. The reason for using truncation instead of rounding is that rounding could move points located near the eastern or northern tile border directly onto the border. According to the rules on assigning features to tiles, a point feature would then be assigned to a neighboring tile. Truncation ensures that point features are always assigned to the same tile.

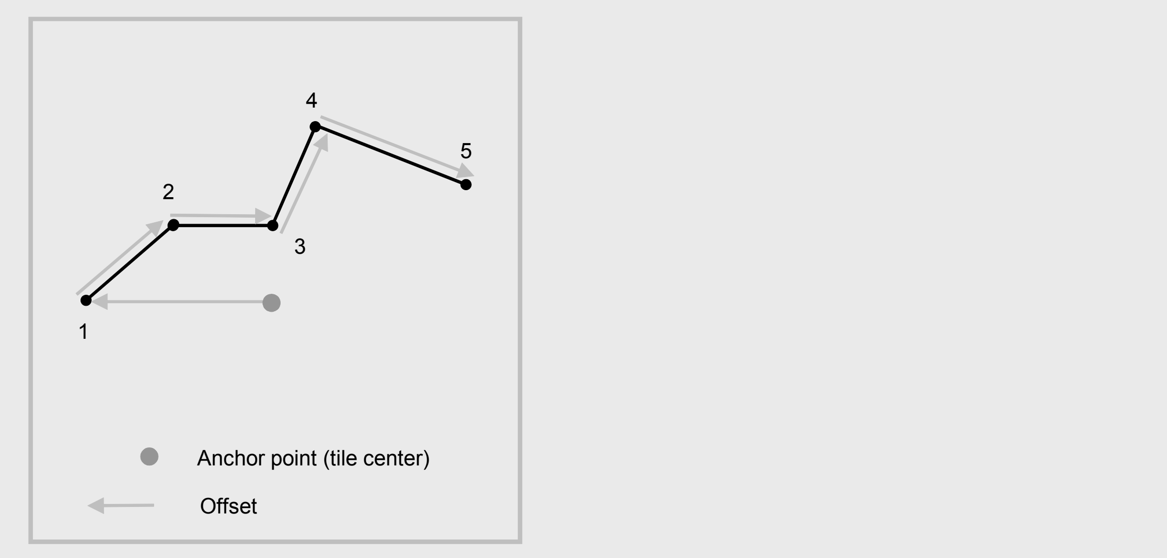
3.4 Difference and Offset Encoding

To save space, coordinates in a tile are encoded relatively to the tile anchor, which is located at the tile center. An offset is a pair of numbers that indicate the differences in x- and y-coordinate values (where x corresponds to the longitude and y to the latitude). The coordinates of single points or the first point of a line are encoded by offsets from the anchor point of the tile they belong to. Subsequent points of lines are encoded by offsets from the predecessor point. The offsets are added component-wise. After adding the offsets, the resulting coordinate for each point of a line must lie in the same tile (meaning in the inner of the tile or on the western or southern border).

For more information, refer to *NDS – Format Specification*, 7.2 *Coding of Coordinates* on page 97.

Encoding offsets for coordinates outside a tile, though syntactically possible, shall be avoided in any case. Offsets between a line point and its predecessor point shall have at least one component different from zero. Otherwise, the line point would be a duplicate of its predecessor and therefore redundant. This can occur as a result of precision reduction and shall be suppressed by the compiler. The offset between points 2 and 3 in Figure 3-4 illustrates the need for offsets where one component is zero.

Figure 3-4 Difference and offset encoding



3.5 Flexible Attribute Mechanisms

Attributes describe the specifics of the different features. For each feature class, a set of attribute types is provided. Attributes may be defined as fixed or flexible attributes.

Fixed attributes are stored with each feature and can be coded very compactly, as the attribute/ value structure is predefined in the feature classes themselves. For a fixed attribute, a value must always be set.

Flexible attributes, on the other hand, describe feature properties for which no fixed structures are predefined. Flexible attributes are mandatory for those features at which information must be available if it applies. An example of a mandatory flexible attribute is the transition mask of an intersection if there is at least one turning restriction. Other flexible attributes are optional and are present if according information is available from the data source.

Note Storage space is always needed for the fixed attributes, whereas flexible attributes consume space only when they are explicitly assigned to a feature.

For more information, refer to *NDS – Format Specification*, 3.3.2 *Attributes* on page 60.

Attribute Groups

Flexible attributes can be combined to form a so-called *attribute group*. Attribute groups enable the modelling of complex traffic regulations, for example, and can also be used to form restricted attributes by combining a restriction with a condition like a time range or vehicle type.

Attribute Lists and References

Flexible attributes that can be derived directly from the input data or can be created using logical and/or mathematical functions, are collected for each tile and each attribute type. They are then compiled into lists according to the *NDS – Format Specification*. Separate attribute lists must be set up for each attribute type and for each reference type of features used in a tile. An attribute list contains all used attribute values together with references to the corresponding feature(s).

Some attributes describe properties of line features (route links, base links, road geometry lines, map lines) for both directions, for example, the `PHYSICAL_WIDTH_METRIC` attribute. Other attributes are specific for one direction, for example, the `SPEED_LIMIT` attribute, and some attributes can apply to one or both directions. Names, for example, often apply to both sides, but can also be different for both sides. The following reference types reflecting directions are available (DataScript location: `nds.common.flexibleattributes`):

- `ROUTING_LINK_DIRECTED`: Refers to route or base links; the direction flag in the reference must be regarded
- `ROUTING_LINK_BOTH_DIRECTIONS`: Refers to route or base links; the direction flag in the reference must be ignored
- `ROUTING_ROAD_GEO_LINE_DIRECTED`: Refers to road geometry lines; the direction flag in the reference must be regarded
- `ROUTING_ROAD_GEO_LINE_BOTH_DIRECTIONS`: Refers to road geometry lines; the direction flag in the reference must be ignored
- `BMD_LINE_FEATURE_DIRECTED`: Refers to road geometry lines; the direction flag in the reference must be regarded
- `BMD_LINE_FEATURE_BOTH_DIRECTIONS`: Refers to road geometry lines; the direction flag in the reference must be ignored

An attribute value (general value) may be assigned to all features of a specific type within a tile. This is achieved by grouping the attribute with the flexible attribute `FEATURE_TYPE` and the reference type `ALL_TILE_FEATURES`. In this case, no attribute references need to be stored with the individual features. References only need to be stored for features that do not use the general value. This reference mechanism should be used if more than 60% of the features use the general value. It shall only be used for attributes that will not be grouped.

Note For side-dependent attributes refer to the

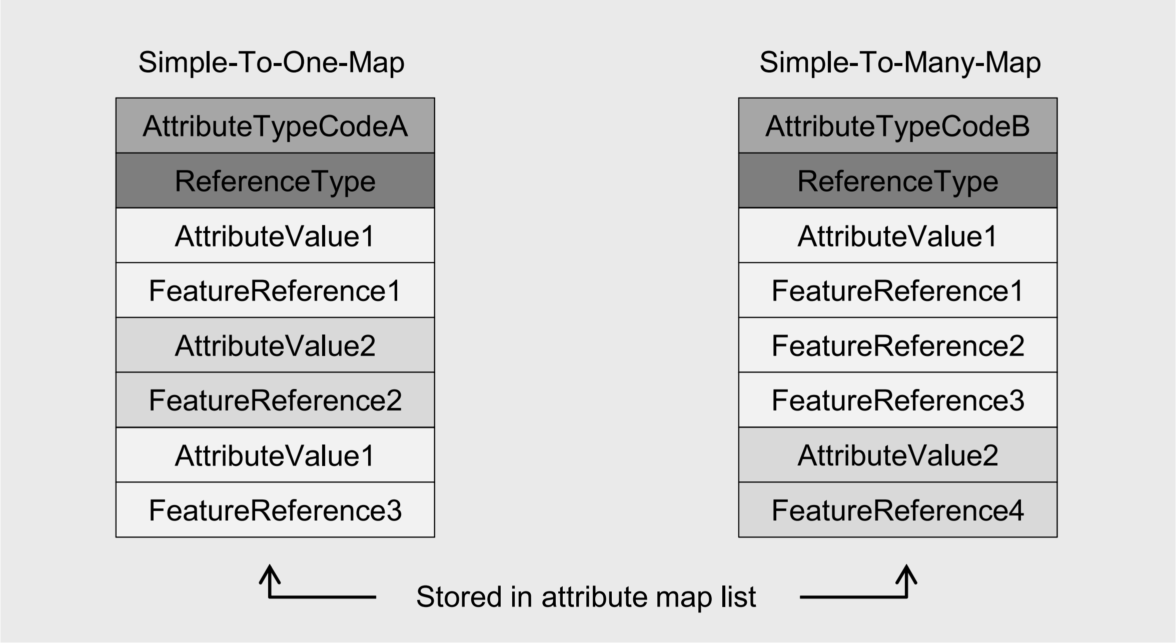
- right side seen from start to end node for forward direction
- right side seen from end to start node for reverse direction

For details on data types and value ranges of the available attribute types, see the DataScript definitions of the attribute types (DataScript location: `nds.common.flexibleattributes`).

In addition, NDS provides alternative reference mechanisms, which make references from attributes to features more compact. The following reference mechanisms may be used for attribute maps without grouping:

- Simple-To-One-Map: An attribute value is assigned to exactly one feature by a single reference.
- Simple-To-Many-Map: An attribute value is assigned to several features by a given number of references following the value.

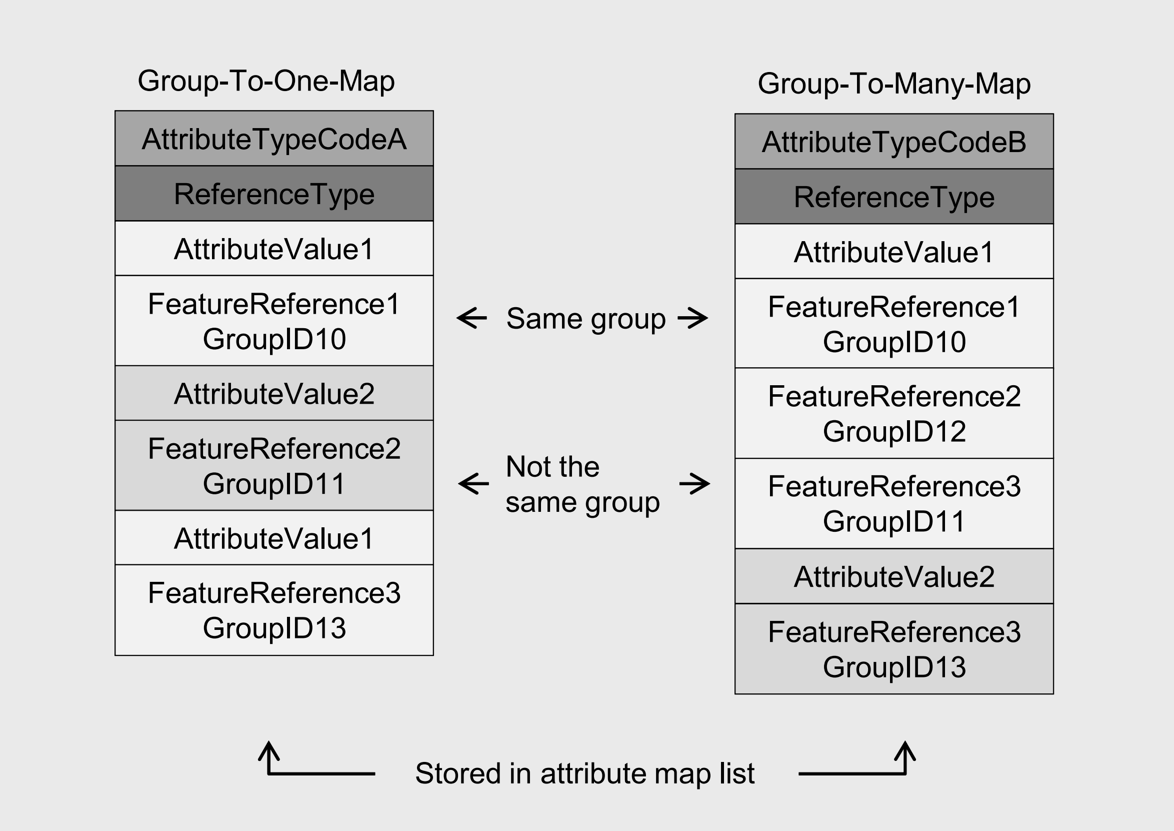
Figure 3-5 Attribute maps without grouping



The following reference mechanisms may be used for attribute maps with grouping:

- Group-To-One-Map: An attribute value is assigned to a single feature, but is grouped with one or more other attribute values of the same or different attribute types. In this case, the reference is paired with a group number.
- Group-To-Many-Map: An attribute value is assigned to several features with a number of references to features. Additionally, each reference is paired with a group number in order to group the attribute value with other attributes, so that each of the features is assigned an additional attribute group.

Figure 3-6 Attribute maps with grouping


Group-To-One-Map
AttributeTypeCodeA
ReferenceType
AttributeValue1
FeatureReference1 GroupID10
AttributeValue2
FeatureReference2 GroupID11
AttributeValue1
FeatureReference3 GroupID13
Same group
Not the same group
Group-To-Many-Map
AttributeTypeCodeB
ReferenceType
AttributeValue1
FeatureReference1 GroupID10
FeatureReference2 GroupID12
FeatureReference3 GroupID11
AttributeValue2
FeatureReference3 GroupID13
Stored in attribute map list

The compiler decides which referencing mechanism is most space-efficient for a specific use case.

3.6 Range Tables

Range tables are introduced into NDS to keep references consistent between features which are located in different tiles of one building block or between features which are located in different building blocks. References between features can be used directly, if the respective features are compiled together in one version. In this case, the range table has only one entry.

In a later version, a single building block or several tiles may be compiled anew, whereas others are not compiled. In this case, feature IDs can change due to deletions and insertions of features. This would lead to inconsistent feature references. Range tables ensure the consistency of the references in this case.

Range tables are modeled as structures which are located in the building block containing the features that are referenced. Range tables are available for the Basic Map Display and Routing building blocks. For each feature type, a separate range table exists per tile. A routing tile, for example, contains a range table for links. See also *NDS – Physical Model Description*, `nds.routing.main` > `RoutingIdRangeList`.

In the initial delivery, the range table for a routing tile with 20 links contains only one entry. References from other building blocks or tiles can be directly mapped to the feature ID of the referenced link. This is illustrated in Table 3-1.

Table 3-1 Reference feature IDs and local indices for a tile with 20 links (initial delivery)

Reference feature ID	Local index
0	0
1	1
2	2
3	3
...	...
19	19

Table 3-2 Range table after initial delivery

Start ID	End ID
0	19

Table 3-3 illustrates the feature IDs used for references and the local indices as well as the according range table after delivery of a minor update of the Routing building block. For this update, the links with the local indices 7, 15, 16 are removed and new links are added with local indices 17 and 18.

Table 3-3 Reference feature IDs and local indices after minor update

Reference feature ID	Local index
0	0
1	1
2	2
3	3
4	4
5	5
6	6
8	7
9	8
10	9
11	10
12	11
13	12
14	13
17	14
18	15
19	16

Reference feature ID	Local index
20	17
21	18

Table 3-4 Range table after minor update

Start ID	End ID
0	6
8	14
17	21

Table 3-5 shows the feature IDs used for references and the local indices as well as the according range table after a second minor update of the Routing building block. In this update, the base links with the local indices 7, 14, 17 are removed and new base links with local indices 16, 17, 18, 19 are added.

Table 3-5 Reference feature IDs and local indices after second update

Reference feature ID	Local indices
0	0
1	1
2	2
3	3
4	4
5	5
6	6
9	7
10	8
11	9
12	10
13	11
18	12
19	13
21	14
22	15
23	16
24	17
25	18

Table 3-6 Range table after second update

Start ID	End ID
0	6

Start ID	End ID
9	14
18	25

For major releases, for example, if a complete database is compiled and delivered, the local indices and the feature IDs are aligned and the range table has again only one entry.

Note

- For incremental updates, new routing or BMD features must be added at the end of the respective array of features of a given type.
- Compiler complexity increases. Effort must be spent for managing the features within the BLOB correctly across releases and for creating range tables.
- The application shall be able to handle range tables. The calculation of the correct feature offsets after an update could be a performance issue.

Update Scenarios

As shown in Table 3-1 to Table 3-5, minor updates with changes within the referenced building blocks result in adjusted range tables. As a result of inconsistent partial updates, referenced features could be missing in the new database. This shall be avoided as it would cause dangling references. The missing features are, however, represented by gaps in the range table and can thus be easily identified by the application.

The ID range for features is limited, for example, 2^{15} for link features. If the end of the ID range has been reached and an update requires new feature IDs, all affected tiles must be recompiled as for major updates in order to reorganize the tile IDs.

For major updates, an ID alignment is performed and the range table has only one entry. The feature offsets in the referenced BLOBs and the IDs used in the references are identical. It is assumed that all building blocks of a major release are installed.

Note A building block of a major release cannot be used with, for example, the Routing building block of an older minor release. Due to the ID alignment, the IDs would no longer fit.

For more information regarding range tables in different update scenarios, refer to *NDS – Update Specification*, 2.3 *Update Scenarios* on page 14.

3.7 Mapping of Data Types

Navigation Data Standard (NDS) uses DataScript, a formal language for modelling binary data types, bit streams, or file formats. The NDS e.V. enhanced the DataScript language with relational extensions (Relational DataScript (RDS)) and, thus, enables the definition of hybrid data models: Whereas high-level access structures are modeled as relational tables and indices in RDS, the low-level bulk data is stored in BLOBs in single columns with a format defined in DataScript.

For more information, refer to Section *NDS – Format Specification*, 3.4 *Introduction to NDS DataScript* on page 61.

The relational extensions allow the definition of relational database tables and indices in DataScript. For each table column, a DataScript data type is specified. The automatically generated source code for encoders and decoders can use the relational tables in the same way as binary data.

The database files for NDS databases shall be stored in a format that can be processed by SQLite 3.7.4-nds3. For SQLite databases, a reference implementation of such a code generator for Java encoders and decoders is available (see *http://www.sqlite.org/datatype3.html*). Whereas most SQL database engines use static typing (data type of a value is determined by its container, that is, the particular column the value is stored in), SQLite uses a more dynamic data type system: The data type of a value is associated with the value itself, and not with the container in which it is stored.

The following SQLite data types (so-called *storage classes*) are available for each value stored in or manipulated by an SQLite database engine.

- `NULL`: Specifies that the value is a `NULL` value
- `INTEGER`: Specifies that the value is a signed integer; depending on the magnitude of the value the signed integer can be stored in 1, 2, 3, 4, 6, or 8 bytes
- `REAL`: Specifies that the value is a floating point value which is stored as an 8-byte IEEE floating point number (see *http://en.wikipedia.org/wiki/IEEE_754-2008*)
- `TEXT`: Specifies that the value is a text string, stored using the database encoding (UTF-8, UTF-16BE, or UTF-16-LE)
- `BLOB`: Specifies that the value is a data BLOB, stored exactly as it was entered

During code generation, the DataScript types are mapped to the SQLite data types. The current code generator for relational DataScript (RDS 0.30) contains such a mapping. It leaves, however, several possibilities to map the DataScript types to SQLite data types. This can cause interoperability problems if compilers and applications of different companies use different mappings.

To avoid this, NDS defines the mapping described in Table 3-7 as mandatory for any implementation of the NDS standard.

Table 3-7 Mapping of DataScript types to SQLite types

DataScript data type	SQLite data type
uint8	INTEGER
uint16	INTEGER
uint32	INTEGER
uint64	BLOB
int8	INTEGER
int16	INTEGER
int32	INTEGER
int64	INTEGER
bit:1 … bit:63	INTEGER
bit:n, n≥64	BLOB
int:1 ... int:64	INTEGER
int:n, n≥65	BLOB
enum of <basetype>a)	Same as <basetype>
stringb)	TEXT
sequencec)	BLOB
union	BLOB
choice	BLOB

DataScript data type	SQLite data type
Fixed/variable/implicit length array of uint8, uint16, uint32, uint64, int8, int16, int32, int64, bit:<num>, int:<num>, compound types[d]	BLOB
sql_integer	INTEGER
DateTime[e]	DATE

a) Mapping enum of all integers and bit:1 … bit:63 into INTEGER. This is helpful in several ways, but mainly to have a relational representation of flags and other Boolean values when used as table columns, for example for the POI building block and the update region and other version metadata. It is also consistent with the mapping of the corresponding basic integer and bit types which are also mapped towards INTEGER.

b) Renaming VARCHAR to TEXT is a terminology change to have a conceptually clean mapping into an SQLite type. The TEXT type is always interpreted as UTF-8, as the PRAGMA command encoding must be set to UTF-8 (see Table 2-1).

c) The sequence nds.common.Utf8String, for example, is mapped to an SQLite BLOB

d) Mapping **all** arrays to BLOB. Treating the various arrays of uint8 as TEXT is a problem for the following reasons: First, the arrays can be any sequence of bytes. SQLite, however, expects TEXT to be one of the following formats: UTF-8, UTF-16BE, or UTF-16LE. Anything in SQLite that depends on a comparison of such a column possibly does not work correctly. Second, such an array of bytes can have zero values. Zero values in an UTF string, however, are an end of string. It is therefore not possible to store a TEXT column with embedded 0 values as text. Instead, it is stored as a BLOB. Finally how would one get an array with embedded 0 values into SQLite, assuming that it would accept such a thing. In C code one would prepare a statement, bind the array to a column, and then execute the statement. But the functions that bind a data to a TEXT column, such as sqlite3_bind_text accept a const char * parameter, so the imbedded 0 would act as end of text and truncate the input. One would probably have to bind the array as a BLOB, and rely on SQLite to store the input as a BLOB (this is the default behavior for a column with TEXT affinity and a data type that does not convert to text). Therefore any array needs to be treated as BLOB in the first place.

e) The subtype DateTime is used to store date and time in a readable string. The subtype is defined as a subset of the ISO-8601 definition and the SQLite type DATE (see *http://en.wikipedia.org/wiki/ISO_8601* and *http://www.sqlite.org/datatype3.html*). An explicit time zone should not be set in the string; the value should either be in LOCAL TIME or in UTC. The comment where the column/value is defined must state if it is LOCAL TIME or UTC.

3.8 Angle Definitions

NDS uses angles in various building blocks. Angle definitions are used, for example, for the rotation of 3D object templates, the rotation angle for icons in Basic Map Display, the tangential angles for clothoids (ADAS), and for link sectors in Routing.

In NDS, angles always have the same reference direction and orientation: The reference direction is always north and the orientation is always clockwise. This means that for the rotation angle for icons, the value 0 refers to the north. The increasing values then lead in clockwise direction from the north to northeast, east, southeast, south, southwest, west, and northwest, and finally back to the north.

The units and ranges of values, however, may differ for each of these angle definitions.

4 Encryption

NDS supports the encryption of data stored in a navigation database to prevent unauthorized use of map data.

The following options for applying encryption are specified for NDS:

- Encryption of database files containing the data for a particular building block of an update region. The database file is defined by the `uri` field in the `UrBuildingBlockVersionTable`.
- Encryption of BLOBs containing tile data or NVCs for a particular building block. BLOBs are defined by the `blob` field in the building block tile table, for example `BmdTileTable`.

For more information on encryption for NDS databases, refer to *NDS – Format Specification*, 8 *Encryption* on page 109.

This chapter introduces interoperability issues that are related to the encryption and decryption of NDS databases. It explains the concept of encrypting NDS databases with the AES-128/CTR cipher algorithm, as well as security issues linked with the use of encrypted NDS databases and recommendations on how to deal with these issues.

4.1 Use of the Cipher Algorithm

NDS uses the Advanced Encryption Standard (AES) for the symmetric cipher algorithm. Each AES cipher has a 128-bit block size, with key sizes of 128, 192 and 256 bits supported. NDS uses the AES-128 block cipher.

To extend the block cipher for longer sequences, and to provide unvarying transformation of original data, NDS additionally uses the AES Counter (CTR) mode (also known as Integer Counter Mode (ICM) and Segmented Integer Counter (SIC) mode). The general idea is to use the block cipher for generating a stream of pseudo-random bits (so-called key stream) which in turn is combined with the original data via the bitwise exclusive-or operation to get the encrypted data. By the properties of the exclusive-or operation, decryption is performed the identical way by re-computing the key stream.

For more information on AES-128/CTR, refer to *http://www.rfc-archive.org/getrfc.php?rfc=3686*.

To encrypt NDS data with AES-128/CTR, proceed as follows:

1. Divide the data (D) into blocks of 128 bit/16 byte. The last block may have less than 128 bit.
2. For each block D_i (`i = 0` ... `NumberOfBlocks–1`) compute the initialization vector IV_i with 128 bit/16 byte as described in Section *Initialization Vector (IV) Computation* on page 64.
3. Encrypt IV_i with the symmetric key K and XOR the result with D_i.

This approach has the following advantages:

- The data size is not limited to being a multiple of the block size. Pad bits for the last block may therefore be cut off.
- No dependency on previous blocks; the data block D_i can therefore be efficiently encrypted or decrypted without encrypting/decrypting data block D_j
- Robust against bit errors

Initialization Vector (IV) Computation

For each data block D_i, an initialization vector IV_i shall be computed that complies with the following rules:

- The initialization vector shall be 128 bit/16 byte.
- The initialization vector shall be an integer used once (so-called nonce), meaning that the IV_i must not be the same as any other IV_j for the symmetric key `K`.
- The initialization vector does not need to be secret.
- The initialization vector shall be unpredictable.

To support the last rule and to prevent certain kinds of cryptographic attacks, for each symmetric key `K` a 2-byte random number RND_K is computed during compilation.

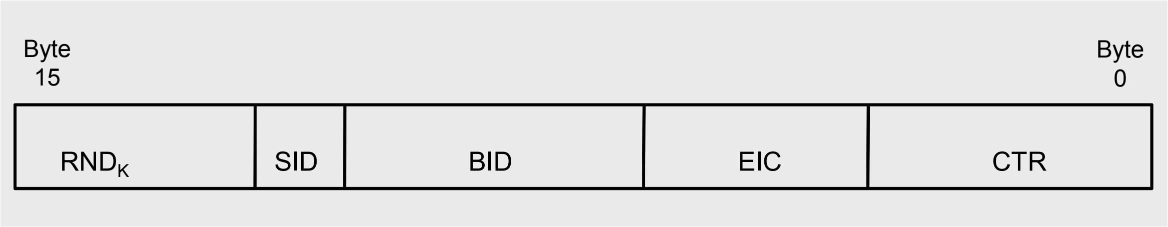
Note The random number RND_K must be delivered to a user together with the symmetric key `K`.

Based on these rules, the following approach for computing the initialization vector IV_i during compilation (by the NDS compiler) is used for a BLOB B to be encrypted with key `K`:

1. Use the 2-byte random number RND_K for key `K`.
2. Concatenate the following values with the random number:
   - Supplier ID (SID) of 1 byte as given in the `ndsDbSupplierId` field of the `nds.overall.version.NdsDatabaseSupplierTable` table
   - Encryptable Item Code (EIC) of 2 bytes taken from the `eic` field of the `EncryptableItemCodeTable` using column name and table name for the given BLOB as the `columnName` and the `tableName` respectively.
   - BLOB ID (BID) of 6 bytes as given by the concatenation of fields constituting the primary key of that table according to the order in the primary key definition and padding with zeroes if necessary.
   - BLOB version (BV) of 2 bytes as given in the `versionId` field of the row containing the BLOB. If the BLOB is not versioned, then BV is `0`.
3. Concatenate with a 3-byte counter (CTR) denoting `i`.

   Note The counter (CTR) starts with `0`.

Figure 4-1 Structure of the initialization vector



For the encryption of database files, the procedure is largely the same. The difference is that the EIC code is taken from the `eic` field of the `UrBuildingBlockVersionTable` in the row that contains the URI denoting the file to be encrypted. BID is set to `0` in this case.

Note	The first 100 bytes of a database file shall not be encrypted. This part of the database represents the database header containing important database properties such as the well-known 16-byte sequence, page size, and so on.

A counter of 3 bytes allows 2^{24} blocks with a maximum of 16 byte each. This results in a key stream of a maximum of 256 MByte. Thus, it is possible to encrypt data of up to 256 MByte.

For computing initialization vectors during runtime (by the application), the same procedure is used with the exception that the RND is not generated but taken as provided by the data provider.

4.2 Compilation of Encrypted NDS Databases

Encrypted NDS databases are compiled in three steps:

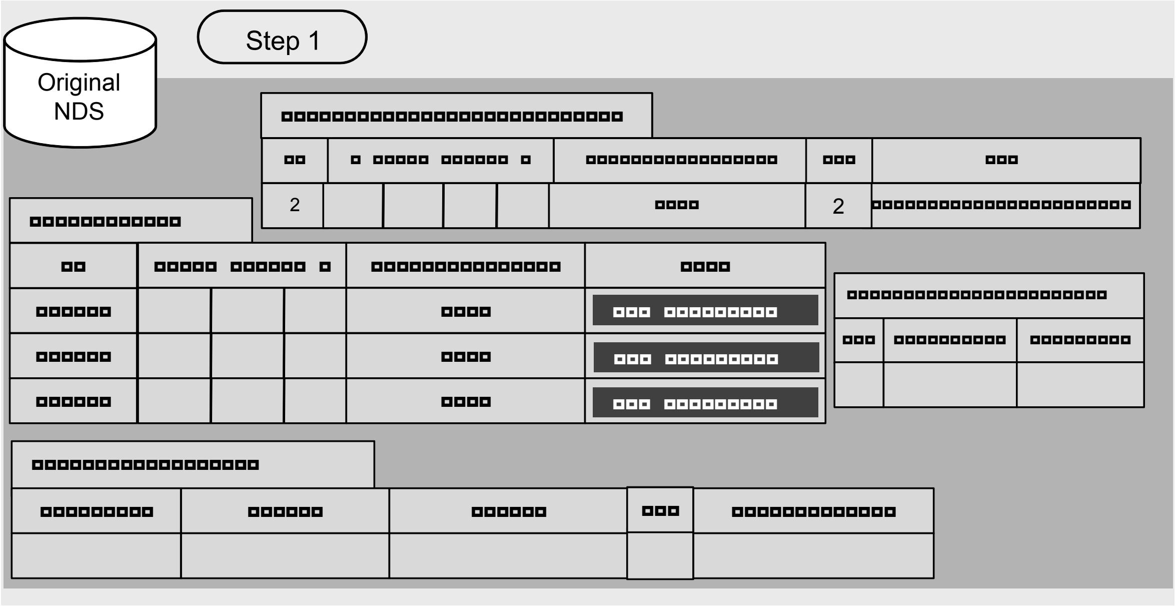
1. Compilation of unencrypted data
2. Encryption of NDS data using symmetric keys
3. Preparation of information about the symmetric keys for delivery to a user

The following sections give a detailed description of these steps.

Step 1: Compilation of Unencrypted NDS Data

In the first step of the compilation process, NDS data is compiled without encryption, meaning that all tiles or databases are unencrypted. The related symmetric key info values are set to `NULL`. The `CipherKeyInfoTable` and the `EncryptableItemCodeTable` are empty. This is depicted in Figure 4-2.

Figure 4-2 Unencrypted NDS data

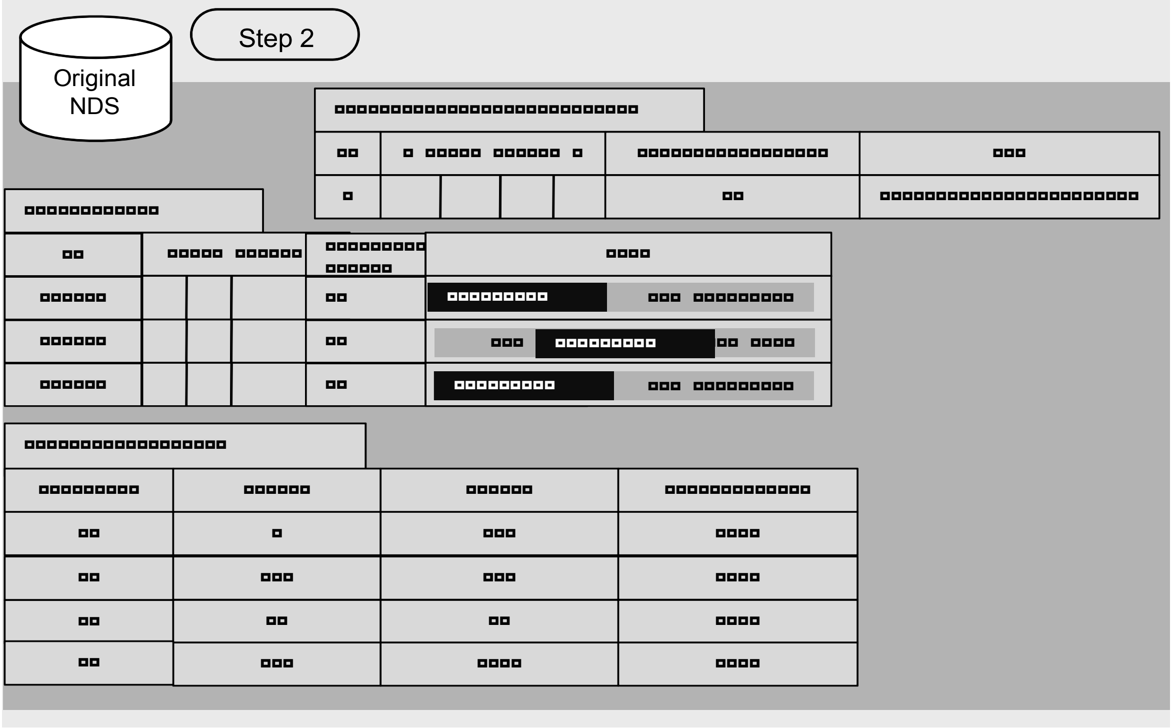


Step 2: Encryption of NDS Data Using Symmetric Keys

In this step, the symmetric key info ID values are set for each BLOB or database. The ID offset and length values are set in the `CipherKeyInfoTable`. The `EncryptableItemCodeTable` is populated with `eic` codes of encrypted fields. The `UrBuildingBlockVersionTable` is populated with cipher key IDs and `eic` codes of the encrypted files. The `cipherKeyData` values are not set, also the `rnd` values from the `CipherKeyInfoTable` are not set. The NDS database is now ready for distribution. Whereas users are now able to copy the NDS database, it is not possible to use the NDS database, because the symmetric keys are missing.

Figure 4-3 illustrates encrypted NDS data without symmetric keys.

Figure 4-3 Encrypted NDS data without symmetric keys



Step 3: Preparation of Information about the Symmetric Keys for Delivery to a User

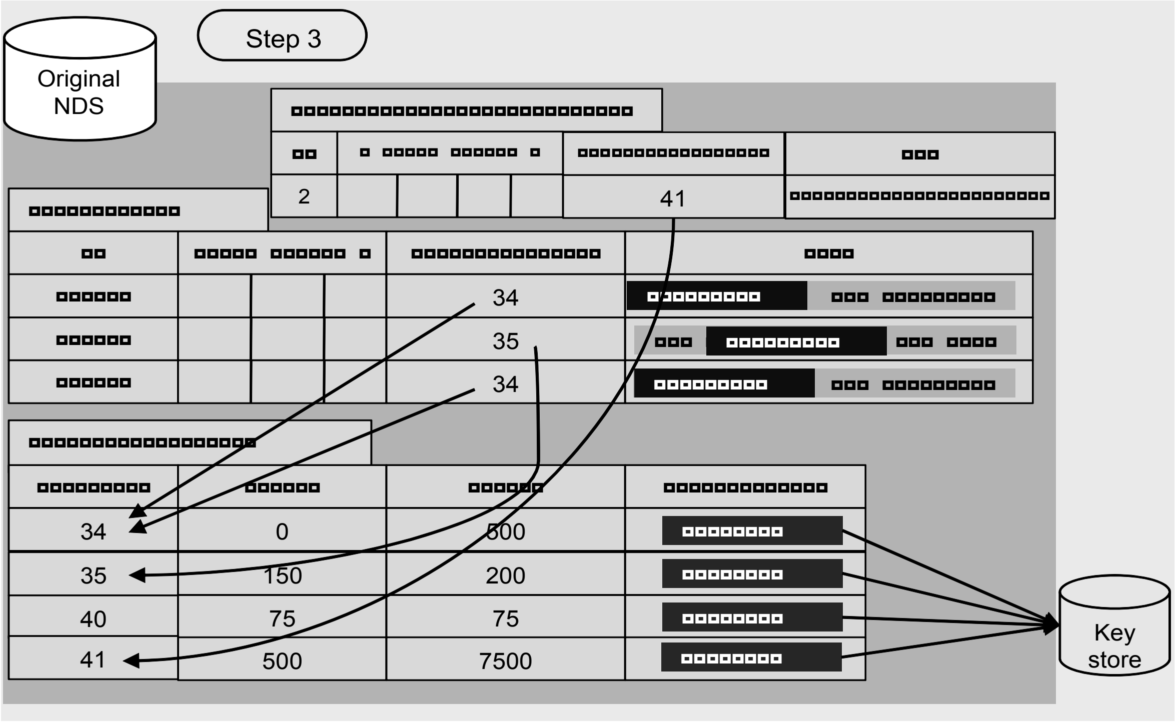
In this step, the symmetric keys are prepared for a secure delivery of the NDS data to the user. The delivery package contains:

- Symmetric keys
- Identifier for each symmetric key
- Random part of the CTR initialization vector (RND) for each symmetric key (see Section 4.1 *Use of the Cipher Algorithm* on page 63)

By receiving the package, the application can populate the `cipherKeyData` and `rnd` fields of the `CipherKeyInfoTable`. The application can, for example, store a reference to the symmetric key in the key store.

Figure 4-4 depicts an encrypted NDS database that can be decrypted and used by the user.

Figure 4-4 Encrypted NDS data ready for use



4.3 Decrypting NDS Data

NDS databases with encrypted data (databases or BLOBs) shall be decrypted before use. For decryption, the following information is required:

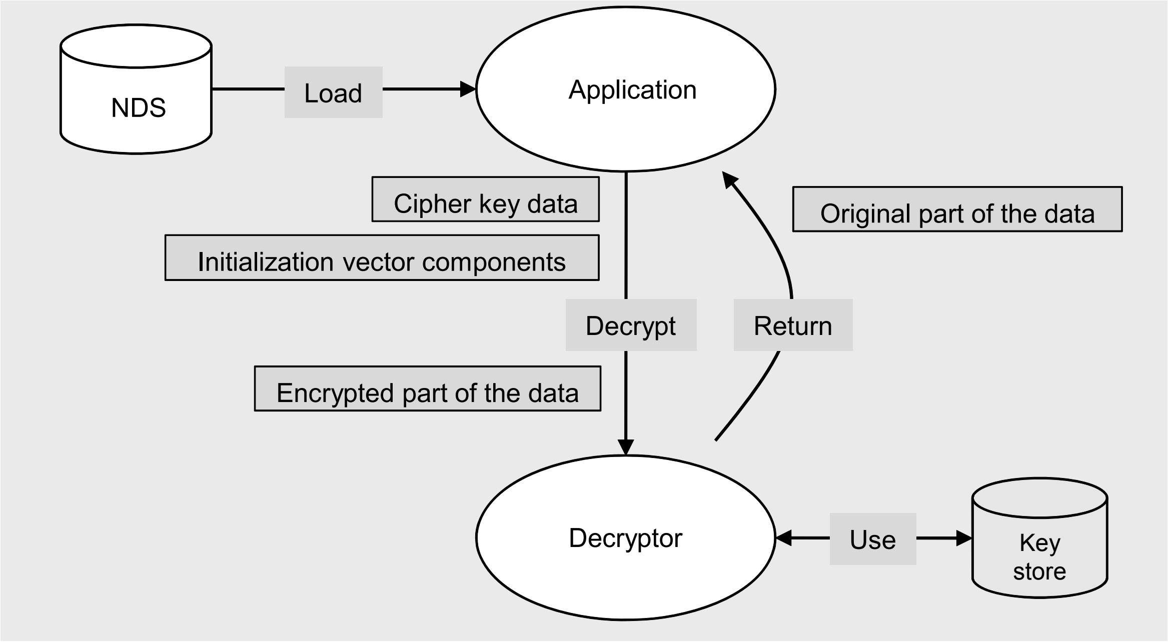
- A software component (Decryptor) that implements the AES-128/CTR algorithm
- Information about the symmetric keys; this information is stored in the `cipherKeyData` field of the `CipherKeyInfoTable`
- Initialization vector components; this is used to construct the initialization vector for decryption as described in Section 4.2 *Compilation of Encrypted NDS Databases* on page 65

The reference to the symmetric key that is needed to decrypt an NDS database is stored in the `useFileEncryption` field of the `UrBuildingBlockVersionTable` of the corresponding database. The reference to the symmetric key that is needed to decrypt a given BLOB is stored in the `cipherKeyInfoId` field in the BLOB table for the corresponding BLOB (for example the `BmdTileTable`).

Figure 4-4 shows an example of an NDS database that is ready for decryption. In this example, tile 331187 and tile 331189 of the BMD building block are encrypted with key 34, and tile 331188 of the Basic Map Display building block is encrypted with key 35. The POI database is encrypted with key 41.

The application decrypts the data by loading the encrypted part of the data, the related symmetric key data, and the initialization vector components from the NDS database. These input parameters are passed to the decryptor. If the decryption is successful, the decryptor returns the original (meaning unencrypted) data. The application then merges the encrypted data with the other data. This process is depicted in Figure 4-5.

Figure 4-5 Decryption process


NDS
Load
Application
Cipher key data
Initialization vector components
Original part of the data
Decrypt
Return
Encrypted part of the data
Decryptor
Use
Key store

4.4 Security Issues

There are two possible security issues with regard to encrypted NDS databases:

- One symmetric key for all users
- Partial encryption of data

These issues as well as recommendations for possible solutions are described in the following section.

One Symmetric Key for all Users

The NDS data supplier sends an encrypted NDS database to a user. The navigation system reads the symmetric key ID for a given tile and sends a request to the NDS supplier to obtain the corresponding symmetric key. To obtain the symmetric key, the navigation system uses asynchronous encryption. The navigation system stores the symmetric key and applies the key to decode the data.

In this process, one symmetric AES-128 key is provided for all users. Any exposure of the symmetric key would compromise the encryption scheme and, for example, allow sharing of data between multiple customers who did not pay for the content.

Even if the symmetric key is well protected, there is still the risk of the scheme being compromised if a user has unrestricted access to the hardware utilizing the data. The hardware should be protected to avoid that the symmetric key can be extracted from the navigation system. This can be done by using trusted computing elements such as Technological Protection Measures (TPM), Digitial Rights Management (DRM), AEGIS, etc.

Another problem can be caused by storing the unencrypted data in the navigation system. If the data is stored on a flash drive, reading the chip would expose decrypted content data. Pirate firmware of the navigation system could accept unauthorized data without encryption, and allow sharing of the content with parties who did not obtain the navigation data from the data provider.

Therefore, all precautions must be taken to ensure that it is not possible to extract unencrypted symmetric keys or the unencrypted files from the navigation system.

Partial Data Encryption

In case of small data changes between versions of navigation data, partial encryption can facilitate decryption or guessing of data. This is especially the case if older data is available. To avoid this, it is recommended to alter the semantic order of objects in a tile between versions to keep the similarities between data with older and newer versions low.

Recommendations

To manage the risks caused by these issues, the following solutions are recommended:

- Using trusted computing technologies, such as technological protection measures (TPM), digital rights management (DRM), and AEGIS, ensures that the navigation system using the map content is not manipulated. The navigation system should not operate, if hardware changes are detected and should not allow non-trusted access to the plain map data.
- Storing the symmetric key in an inaccessible location, for example, in a self-destructible chip (offered by some versions of TPM)
- Variable communication protocol for obtaining the symmetric key, for example, by using nonces (integers used once) to prevent man-in-the-middle scenarios and replay attacks
- Storing the data at least partially encrypted on the navigation system to prevent simple reading of unencrypted data
- Reordering tile data for each new version to prevent that the new data content can be guessed based on the old content

5 Routing Building Block

In order to ensure reproducible NDS data from input data, it is required to follow a specific sequence when compiling data for the Routing building block. Some of the steps described below are optional, others are mandatory. The steps are briefly described below; for detailed information refer to the individual sections of this chapter.

Step 1: Adjusting Direction of Links (Optional)

Road elements have a start point and an end point defining their direction. This direction can be arbitrary in the source data. The compiler may adjust the directions in a uniform way as described in Section 5.1 *Adjusting Direction of Links* on page 72.

Step 2: Reducing Bivalent Intersections (Optional)

Source data can contain road elements that belong to the same road between two intersections in the real world, but are split at artificial intersections in the map data. To reduce the overall data size and to minimize the number of links to be considered by a route calculator, it is recommended to merge this type of road elements before further processing. For more information, refer to Section 5.2 *Reducing Bivalent Intersections* on page 72.

Step 3: Splitting Road Elements (Mandatory)

Road elements intersecting tile borders must be split with respect to the NDS tiling scheme and according to the rules that apply to routing features. Elements to be split are intersections, links, and road geometry lines. For more information, refer to Section 5.3 *Splitting Road Features* on page 73.

Step 4: Sorting Links at Intersections (Mandatory)

Intersections contain a list of all link IDs connected to them. The list entries shall be sorted clockwise. For more information, refer to Section 5.4 *Sorting Links at Intersections* on page 75.

Step 5: Assigning Routing Features to Tiles (Mandatory)

Routing features created from input data are assigned to exactly one tile. For more information on assigning intersections to tiles, refer to Section 5.5 *Assigning Intersections to Tiles* on page 75.

Step 6: Adjusting Coordinates (Mandatory)

Steps 1 to 5 must be carried out with the full coordinate resolution available from the input data. Coordinates of intersections and shape points must be adjusted to the resolution specified for level 13 using the according coordinate shift value. For more information, refer to Section 5.7 *Adjusting Coordinates* on page 76.

Step 7: Generalizing Data for Higher Levels (Mandatory)

Once the tiles on level 13 have been compiled, they contain all routing features for the road network (base links, route links, road geometry lines, and intersections). These features form the basis for the next compilation step – generalization. During generalization, the higher level features are created by removing unimportant features and aggregating the important features. On higher levels, only route links and intersections exist. For more information, refer to Section 5.8 *Generalizing Features for Higher Levels* on page 77.

5.1 Adjusting Direction of Links

Road elements have a start point and an end point defining their direction. This direction can be arbitrary in the input data. A compiler may adjust the direction of all road elements. Direction adjustment means that the most southern end point is defined as the start point and the other end of the link as the end point. If both ends of a link are on the same latitude, the most western point is chosen as the start point.

5.2 Reducing Bivalent Intersections

Source data can contain road elements that belong to one road between two intersections in the real world, but are split at artificial intersections because attributes or names change at the corresponding split points. Compiling the input data unaltered will produce a large number of intersections that have two adjacent road elements only. To reduce the overall data size and to minimize the number of links to be considered by a route calculator, it is recommended to merge this type of road elements before further processing.

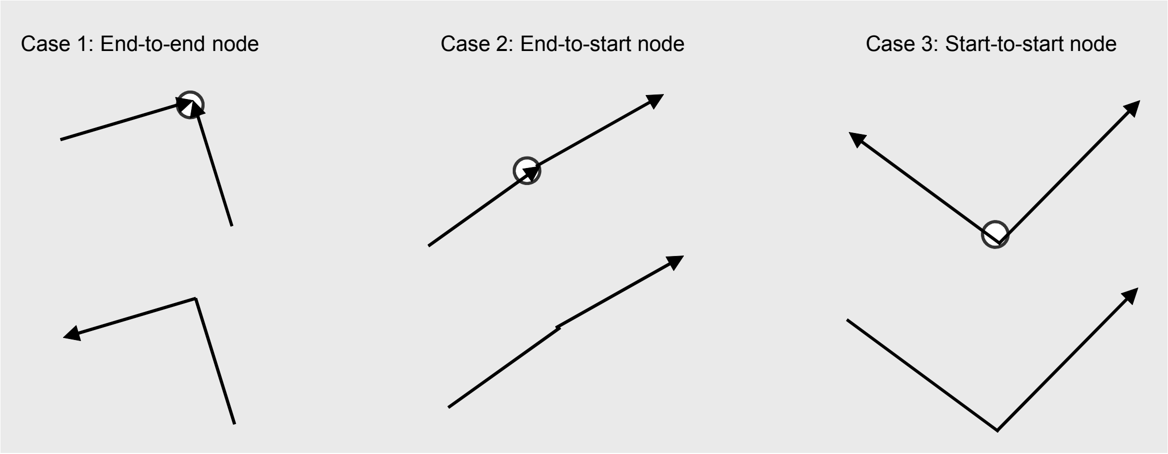
Nodes of source data connecting only two road elements may be deleted. The road elements are then merged into one.

Nodes that represent artificial intersections can connect road elements in one of the following ways:

- End-to-end node (see case 1 in Figure 5-1)
- End-to-start node (see case 2)
- Start-to-start node (see case 3)

Figure 5-1 shows the road elements before processing (with artificial intersections) and after processing. In these examples, the optional compiler step of adjusting the link direction has been applied. If the compiler does not adjust the link direction, the resulting links can have either direction. Note that in case 1 and 3 the direction of the joined road elements differs from the direction of its original parts.

Figure 5-1 Merging of links



The splitting node becomes a shape point of the joined road elements. Common attributes of the original road elements are assigned to the joined road elements, while different attributes are assigned to the parts which are specified by validity ranges in a later process.

For more information regarding attribute handling, refer to Section 5.9 *Handling Attributes* on page 82.

Note The reduction of bivalent intersections can lead to so-called *tear drops*, meaning links forming a loop. The compiler shall ensure that links do not start and end at the same intersection.

5.3 Splitting Road Features

A road feature whose start point, end point and all its shape points (if existing) lie within one tile is not split, but is stored as a base link in the corresponding tile.

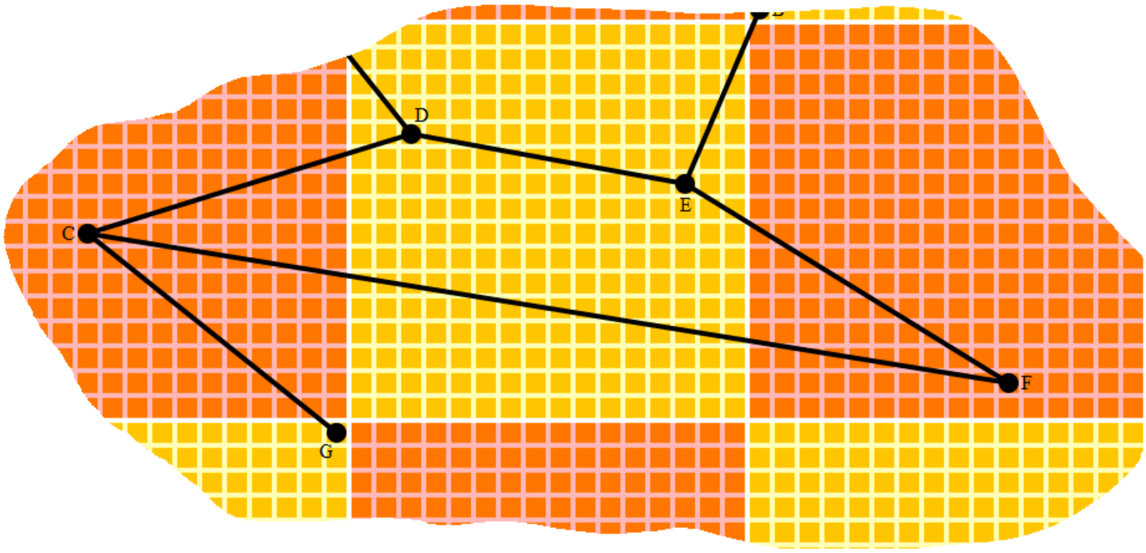
A tile is a matrix consisting of cells (discrete locations) identified by row and column. The number of rows and columns is equal. All tiles located on one level have the same number of cells. See also *NDS – Format Specification*, 7.3.1 *Tiles and Levels* on page 101.

If a road feature has it points spread over more than one tile, its geometry has to be split. The resulting parts are stored as road geometry lines in the different tiles. The split road feature is then stored as one route link in the tile where its start point is located. The split point is always located in row or column 0 and is part of both road geometry lines.

Note If one of the resulting road geometry lines consists of only one point, the road geometry line must be omitted.

Figure 5-2 illustrates a road feature, which road geometry line end points extend tile borders.

Figure 5-2 Splitting road features



After splitting a feature, its parts may be assigned to tiles other than those of the original features. For more information on referencing, refer to Section 5.12.1 *References between Features on Different Tiles* on page 86.

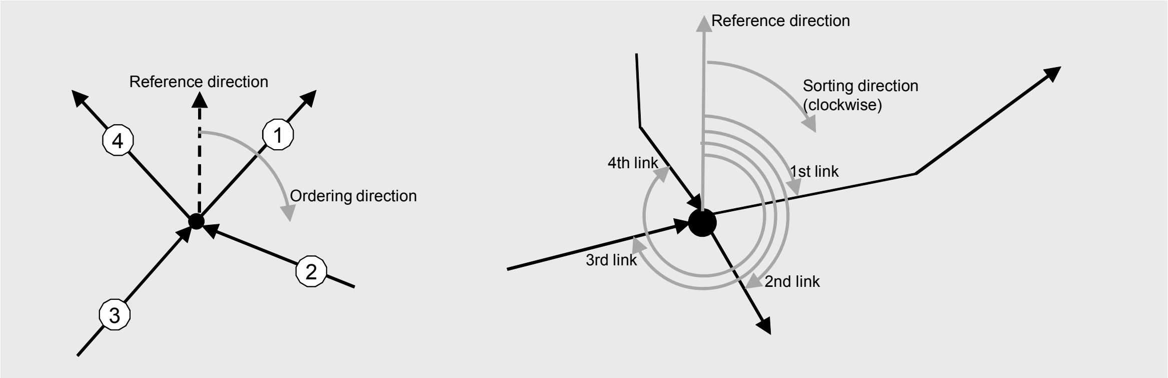
Attributes that were assigned to the original feature, must now be assigned to its split parts. For more information on attribute handling, refer to Section 5.9 *Handling Attributes* on page 82.

Road features intersecting more than 255 tiles would cause an overflow in the `Link2TileList` table (see Section 5.12.1 *References between Features on Different Tiles* on page 86). This table contains the references between the route link and the tiles where the road geometry lines are stored. This can occur, for example, for ferry lines that are longer than 200 km. To avoid this problem, the route link shall be split into two route links connected by a bivalent intersection. It is recommended to do this at shape points where attributes change, or in the middle of the original road feature if no shape points are available.

5.4 Sorting Links at Intersections

Intersections contain a list of all link IDs connected to them. The list entries shall be sorted according to the following ordering scheme: Starting from North in clockwise direction (that is, in mathematical negative direction; see Figure 5-3).

Figure 5-3 Sorting of links at intersections



The angle of the link arising from the intersection is relevant for sorting. The angle is determined in the same way as described in Section 5.10 *Link Angles* on page 85.

This sorting procedure has the following advantages:

- Physical data becomes predictable and reproducible
- Determination of maneuver recommendations for intersections is simplified

5.5 Assigning Intersections to Tiles

Intersections are represented by exactly one point, defined by a pair of coordinates. This point is assigned to the tile on which it is located if one of the following conditions is met:

- The point lies within the tile, meaning not on one of its borders or vertices.
- The point lies on the south-west vertex of the tile.
- The point lies on the western tile border, but not on the north-west vertex.
- The point lies on the southern tile border, but not on the south-east vertex.

Navigation Data Standard defines a tile as follows: If (x1, y1) is the south-west corner of a tile and (x2, y2) is its north-east corner, then all points (x, y) with x1 ≤ x < x2 and y1 ≤ y < y2 are uniquely assigned to the tile.

5.6 Assigning Data Quality Attributes

The data quality of a road is defined by means of the flexible attributes DIGITIZATION_STATUS and ATTRIBUTION_STATUS.

Three values are available to specify the **digitization status** of a road. A default value for the digitization status is assigned to a routing tile. A different value can be assigned to individual links in the tile, for example, because these links belong to another country or area. The value assigned to an individual link overrides the default value.

Example The default value FULLY_DIGITIZED is assigned to all links of tile 2574. The links 4, 5, and 9 of tile 2574 belong to another area with a different digitization status. The value INTERCONNECTING_ROADS is therefore assigned specifically to these links.

The three values for the **attribution status** are usually assigned to single links. If all or nearly all links of a tile have the same attribution status, this value can be assigned as a default value for all links of the tile. Exceptions to this value can still be assigned to single links and override the default value.

Note The attribution status cannot be used to specify whether ADAS attributes are available for specific roads. ADAS attributes are stored in the ADAS attribute layer. Consequently, an application needs to check this layer.

Note Only if a default value is set for the whole tile, a different value can be set for a link. It is not allowed to assign more than one value of each, attribution and digitization status, to one link.

For more information, refer to *NDS – Format Specification*, *Flexible Attributes for Specifying Data Quality* on page 349

5.7 Adjusting Coordinates

Steps 1 to 5 of the processing sequence are carried out with the full coordinate precision available from the input data. In the next step, coordinates must be adjusted to the precision specified for level 13 using the according shift value.

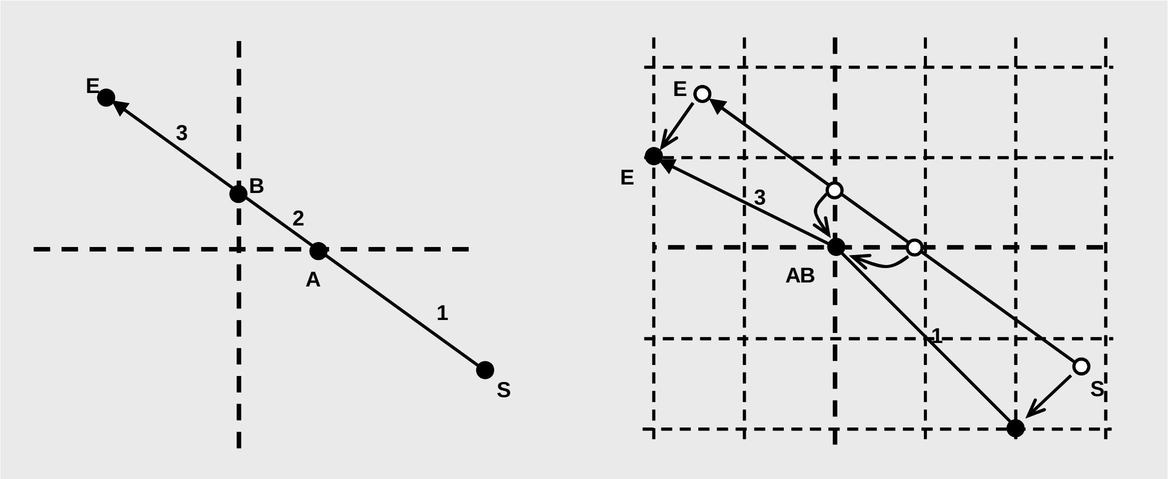
For general information on coordinate precision, refer to Section 3.3 *Reducing Coordinate Precision* on page 49.

As a consequence of precision reduction, it is possible that two points which are apart from each other by 160 meters on level 13 (with an increasing shift value of 1 per level) collapse to a single point on level 4. A route link that connects these points must still be present on level 4, although its start and end point has identical coordinates. In addition, the original link length that is stored on level 13 must be propagated as an attribute to the corresponding links on the higher levels through to level 4.

Figure 5-4 illustrates the adjustment of coordinates. The bold dashed lines show the borders of the tiles, the non-bold dashed lines represent the coordinate grid.

Points S, A, B, and E are moved to the next southern and western line of the grid. As a result, A and B lie on top of each other. The road geometry line between A and B collapses to one point and will be deleted. The resulting geometry consists of the two road geometry lines 1 and 3. Line 1 runs from point S to point B and is located in the lower right tile; line 3 runs from point A to E and is located in the upper left tile.

Figure 5-4 Adjusting coordinates



5.8 Generalizing Features for Higher Levels

Once the tiles on level 13 have been compiled, they contain all routing features of the road network (base links, route links, and intersections). These features form the basis for the next compilation step – generalization. During generalization, the higher level features are created by removing unimportant features and aggregating the important features. On higher levels, only route links and intersections exist.

During generalization, references between the features of different levels must be maintained for the construction of uplink and downlink tables. For more information on referencing, refer to Section 5.12 *Referencing* on page 86.

A simplification of road geometry is not applied, because on higher levels there are only route links that have no shape points.

Note	The positions of the start and end nodes are not changed.

5.8.1 Filtering of Links by Road Classes

The functional road class defines the importance of a road within the road network. Smaller numbers describe roads of higher importance, whereas higher numbers indicate lower importance. 0 denotes the highest class, 7 the lowest class. An NDS database shall contain the classes 0 … 4; the classes 5 … 7 may be available in an NDS database if provided with the input data.

Data providers shall classify all roads according to the road class scheme in Table 5-1.

Table 5-1 Functional road classes

Functional road class	Description
0	Roads of international importance
1	Roads of national importance
2	Roads of regional importance
3	Road of high local importance
4 … 7	Roads of local importance

The functional road class values are evaluated for generalization on higher routing levels. To this purpose, links are filtered on the basis of the classification scheme in Table 5-2[1].

Table 5-2 Filtering of road classes on levels

Level in Routing building block	Description
4	Contains roads marked in the source with the functional road class 0 Roads with functional road class 1 or lower will be left out.
6	Contains roads marked in the source with the functional road classes 0 … 1 Roads with functional road class 2 or lower will be left out.
8	Contains roads marked in the source with the functional road classes 0 … 2 Roads with functional road class 3 or lower will be left out.
10	Contains roads marked in the source with the functional road classes 0 … 3 Roads with functional road class 4 or lower will be left out.
13	Contains all roads

[1] Levels subject to change according to prototyping experiences

5.8.2 Aggregating Links

After less important roads have been filtered out, the data will contain a high number of intersections with two adjacent links only. These intersections may be deleted, and the adjacent links may be merged. The resulting new links are called *aggregated links*. The length of an aggregated link corresponds to the sum of all links it is composed of.

Note	To maintain the correct topology of the routing network, the two intersections of a link shall always be distinct. In other words, a link shall not be a loop. Two links shall not be aggregated if the result would form a loop.

Note	Two links shall **not** be aggregated if they are of different functional road classes.

5.8.3 Handling Attributes of Aggregated Links

An aggregated link must inherit all attributes of its components that are relevant for routing.

Fixed attributes of the lower level features that are common to all components shall be transferred as fixed attributes to the aggregated feature on the higher level. Link angles on higher levels must be equal to the corresponding link angles on level 13. Fixed attributes that are not valid for the entire aggregated link shall be assigned by using flexible attributes that indicate the different values.

Flexible attributes of the components shall also be stored as flexible attributes of the aggregated feature if they are relevant for routing.

If the components of an aggregated link have the same value for an attribute type, the values shall be accumulated in the aggregated link. The same applies to attribute groups: Values shall be summarized if the grouped attribute types and their values are identical.

Rules for Propagating Attributes to Aggregated Links

The following rules apply to propagating attributes together with their values to the aggregated links on higher levels:

- If several values are defined for fixed attributes, the least value is transferred for the fixed attribute to the aggregated link on the higher level. Differing values due to length dependencies are assigned additionally as a corresponding flexible attribute grouped with `LENGTH_ALONG_LINK`.
- Flexible attributes on level 13 are transferred as flexible attributes and grouped with `LENGTH_ALONG_LINK`.

- Attributes describing passing restrictions are propagated to the aggregated links. If they build attribute groups, they are assigned to the aggregated link as a group.
- Conditional attributes forming a group are assigned to the aggregated link as a group. It is not possible to merge several groups into one group on higher levels.

Note Additional note for link type: If different link types have been assigned to the links on level 13, the aggregated link is assigned the link type `NO_SPECIAL`.

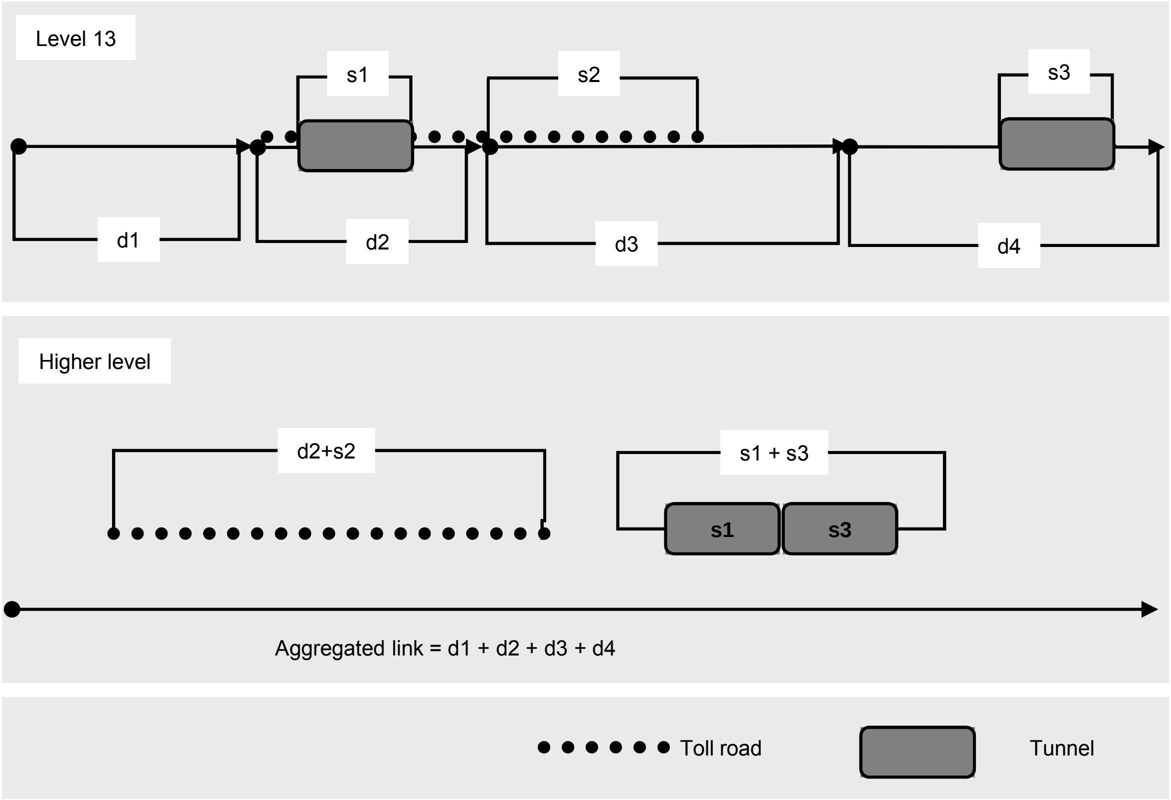
The following attributes of lower level links are examples for attributes that are not relevant for aggregated links. They shall not be propagated:

- Maneuver codes
- Signposts
- Lane markings
- Lane separators

Example 1: Aggregation of Link Attributes (Toll and Tunnel)

Figure 5-5 shows an example.

Figure 5-5 Aggregation of link attributes



On level 13, the following attributes are assigned to the link sequence d1-d4:

- Link d2 is assigned the fixed attributes TOLL and TUNNEL and the flexible attributes TUNNEL and VALIDITY_RANGE.
- Link d3 is assigned the fixed attribute TOLL and the flexible attributes TOLL and VALIDITY_RANGE.
- Link d4 is assigned the fixed attribute TUNNEL and the flexible attributes TUNNEL and VALIDITY_RANGE.

On a higher level, the links in this example are aggregated and become one link.

The aggregated link is assigned the fixed attributes TOLL and TUNNEL, indicating that there is at least one tunnel and one toll segment along the link.

Further flexible attributes must be provided to enable the application to determine the total extent of the properties. For this purpose, the flexible attributes TUNNEL and TOLL are grouped with the flexible attribute LENGTH_ALONG_LINK. This attribute group describes the length of the tunnel which requires toll.

Example 2: Aggregation of Link Attributes (Continued Turn Restriction)

The flexible attribute CONTINUED_TURN_RESTRICTION specifies a series of links, which are not allowed to travel in the given sequence. This type of attribute must be propagated to upper levels of routing data to ensure that the restriction is consistently regarded during route calculation.

If the series of links crosses tile borders, the flexible attribute TILE_OF_CONTINUED_TURN_RESTRICTION is added. This flexible attribute uses an external tile ID index indicating where the the original continued turn restriction can be found.

The link carrying the flexible attribute CONTINUED_TURN_RESTRICTION on a given level, and all links referenced by the attribute must exist on the next higher level or must at least be part (child) of a link on the next higher level. If this requirement is fulfilled, the parent of the given link also gets the flexible attribute CONTINUED_TURN_RESTRICTION with a list of links containing all parents of the links listed in the attribute on the lower level.

Note This may result in a shorter list, because several links listed in the attribute on the lower level may result in a common link on the higher level.

The last link listed in the attribute on the lower level may be represented on the higher level by a link generated also from other links than those listed in the lower level.

5.9 Handling Attributes

How different types of attributes are created from input data, depends on format, content, and quality of the input data, which is not part of the NDS standard. For more information on data types and value ranges for the available attribute types, refer to the *NDS – Physical Model Description*.

For the creation and storage of flexible attributes, the following general rules apply:

- The large number of flexible attribute types for routing features is distributed to different attribute layers. These attribute layers are partitioned into tiles in the same way as other routing data. The attribute layers are stored in separate columns of the routing SQLite tables.

  For more information on attribute layers, refer to *NDS – Format Specification*, 6.2 *Attribute Layers* on page 90.
- If a flexible attribute is not valid for the entire link or road geometry line, the flexible attribute `VALIDITY_RANGE` must be assigned. Attribute points must be used, if a validity range is not bounded by either the start or end node, or by shape points of a base link or road geometry line.

  For more information refer to Section *Rules for Attribute Point Use* on page 83 and *NDS – Format Specification*, 6.2 *Attribute Layers* on page 90.
- If at least one maneuver is not allowed between links adjacent to an intersection, an appropriate pointer to a mask attribute (`TRANSITION_MASK_2_4_VAL`, `TRANSITION_MASK_5_VAL`, or `TRANSITION_MASK_6_8_VAL`) must be assigned tot he intersection.
- If a maneuver is conditionally not allowed, a flag must be set in the transition mask, as if the maneuver is always prohibited. Conditions can be, for example, specific periods or specific vehicle types. The conditions under which the maneuver is allowed are defined by means of flexible attributes.
- If the passage of a link is prohibited conditionally in one or both directions, the value of the fixed attributes must be set to „prohibited" in analogy to the maneuvers above.
- A prohibited passage into a one-way street must not be stored as a transition. Instead, one-way directions must be used. The format does not ensure by means of transitions that driving into one-way streets is blocked.

For more information on flexible attributes, refer to Chapter 3.5 *Flexible Attribute Mechanisms* on page 52.

The following examples illustrate how to model typical real world situations by means of attributes describing characteristics grouped with condition attributes.

- Prohibited passage for trucks on Saturdays and Sundays:
PROHIBITED_PASSAGE grouped with FREQUENTLY_USED_VEHICLE_TYPES (value = isTruck) and DAYS_OF_WEEK (value = isSunday, isSaturday)
- Prohibited passage for all motorized vehicles except emergency cars
PROHIBITED_PASSAGE grouped with FREQUENTLY_USED_VEHICLE_TYPES (value = isAllMotorizedVehicle) and PUBLIC_SERVICE_VEHICLES (value = isEmergency, isInclusive = FALSE)
- Prohibited passage for all vehicles from the 1st to the 10th day of a month, except between 6 and 10 o'clock:
PROHIBITED_PASSAGE grouped with DAYS_OF_MONTH (value = isDay1… isDay10, isInclusive = TRUE) and TIME_RANGE_OF_DAY (value = startTime=6:00, endTime=10:00, isInclusive=FALSE)
- Overtaking prohibition for trucks, cars with trailers, and busses from 6 to 20 o'clock, modeled by 2 attribute groups:
First group: OVERTAKING_PROHIBITION grouped with FREQUENTLY_USED_VEHICLE_TYPES (value = isTruck, isBus) and TIME_RANGE_OF_DAY (value = startTime=6:00, endTime=20:00, isInclusive=TRUE)
Second group: OVERTAKING_PROHIBITION grouped with FREQUENTLY_USED_VEHICLE_TYPES (value = isPersonalCar) and EQUIPMENT_AND_OCCUPANCY (value = hasTrailer) and TIME_RANGE_OF_DAY (value = startTime=6:00, endTime=20:00, isInclusive=TRUE)
- Signpost to A-city via link xy located at position mn valid for the right two of four lanes and right hand traffic:
SIGNPOST_TOWARD_INFO_REFERENCE (value = NamedObjectId of A-city) grouped with SIGNPOST_LINK_REFERENCE (value = IntOrExtDirectedLinkReference of link xy) and VALIDITY_RANGE (value = shape or attribute point of position mn) and NUM_LANES (value = normalLanes=4, exitLanesRight=0, exitLanesLeft=0) and LANE_MASK (value = isFirstLane, isSecondLane)

Rules for Attribute Point Use

The following rules apply for the use of attribute points in NDS:

- Attribute points are separately stored per attribute layer as ungrouped flexible attributes and are assigned to the features for which they are used. The same attribute point can be used for different types of flexible attributes which are stored in the same attribute layer.
- Attribute points are identified by an ordinal number with respect to a base link or road geometry line. The start node has the ordinal number 0 and the first shape or attribute point has the ordinal number 1, and so on. The attribute point position is defined by a coordinate difference to the preceding shape point or the start node if there is no preceding shape point. Only the larger value of difference in latitude and longitude is needed, because attribute points are always located on a straight line between two shape points or the start/end node and a shape point.

Travel Direction Attribute and Conditions for Travel Directions

The travel direction is modeled by means of a fixed attribute indicating in which direction a link can be used by vehicles. The travel direction attribute is used to model one-way streets. When set to `IN_NO_DIRECTION`, the travel direction attribute can also be used to indicate blocked roads.

To assign conditions to a travel direction, such as time-dependent road blockings, flexible attribute groups are used. It is, for example, possible that the travel direction for a link changes during a day: Positive direction is open between 9 am and 2 pm, and negative direction is open between 2 pm and 9 am. This is modeled as follows:

The fixed attribute `travelDirection` is set to `IN_NO_DIRECTION`, that is, the link is closed in both directions. The attribute `PROHIBITED_PASSAGE` is grouped with a conditional attribute for a time interval, for example, 9 am to 2 pm for one direction. The attribute `PROHIBITED_PASSAGE` is also assigned to the other direction of the link, but grouped with a conditional attribute for the time interval 2 pm to 9 am.

The same mechanism applies for other conditions, such as vehicle type or weight.

Handling Time Zone Attributes

The time zones applicable for an update region are defined in the time zone list stored in the global metadata (DataScript location: `nds.overall.metadata.RegionMetadata`). If more than one time zone applies for an update region, the flexible attribute `TIME_ZONE_REFERENCE` is used to assign a time zone to a tile or specific routing features.

The time zone attributes are attached to tiles and/or base or route links on all levels of the Routing building block. Each tile has an index into the time zone list determining the time zone it is predominantly located in. Features located in a different time zone than the tile have their own time zone attribute. It is necessary to split features at time zone borders and reconnect the split links by means of bivalent intersections.

Handling Route Guidance Attributes

Flexible attributes relevant for route guidance shall be stored in the Routing geo tiles, see also *NDS – Format Specification*, *Routing Tiles and Routing Geo Tiles* on page 118.

Examples for using route guidance attributes:

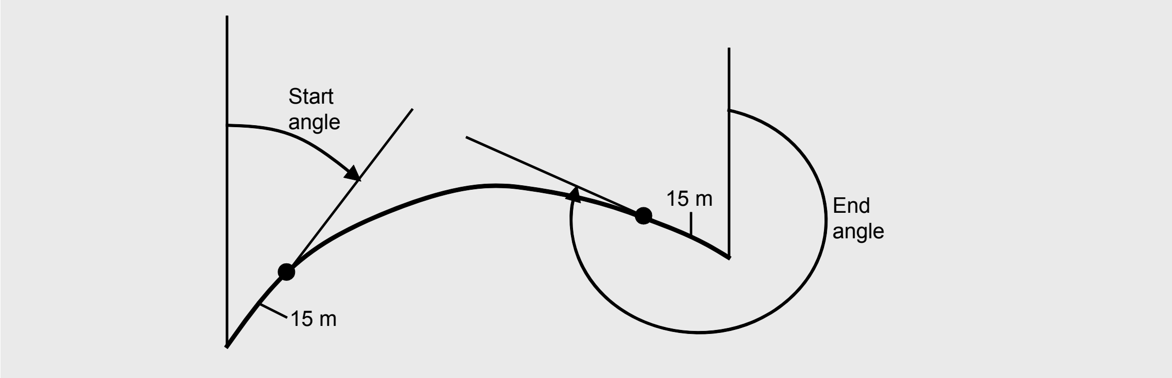
- **Maneuver instructions**: In some situations, the route guidance instructions to be given at an intersection cannot be derived unambiguously from the transition assigned to the intersection. In this case, a flexible attribute of type `SPECIAL_TRANSITION_CODE` may be assigned to the transition. Based on the transition code, the driver can be instructed to stay on the main road, for example . Alternatively, route guidance may be deactivated for a transition.
- Route guidance regarding **routes with high frequency of accidents**: To mark links that belong to a route with high frequency of accidents, the flexible attribute `FREQUENT_ACCIDENTS` is provided.

5.10 Link Angles

Links have a start angle and an end angle. The angles are modeled by means of the fixed attributes `startAngle` and `endAngle` (DataScript location: `nds.Routing.link`).

The angles are stored in 64 sectors, with the first sector starting from North in clockwise direction. This results in a resolution of 5.6°. For calculating the angles, a point located 15 meters from the start/end of the link is used. This is depicted in Figure 5-6. If a link length is less than 15 meters, the last point shall be used.

Figure 5-6 Link angles



5.11 Handling Urban Indicator, Administrative Road Class, and Traffic Sense

Urban Indicator

The urban indicator provides information on whether a link is located in an urban area. This information can, for example, be used to estimate an average speed for a route.

If an urban indicator or any flag is provided in the input data indicating that a road feature is within an urban area, the urban indicator shall be set in the fixed attribute set of links.

Administrative Road Class

The administrative road class is a fixed attribute. It is available in all links and road geometry lines. The attribute value `0` is reserved for roads with unknown administrative road class. All other classes are ordered from 1 (highest hierarchy level) to 7 (lowest hierarchy level).

A company compiling a database is responsible for mapping the classification scheme of the data provider to the seven classes. If less than seven classes are delivered, some values will not be used. If more than seven classes are delivered, classes have to be merged.

If databases contain several countries, different numbers of administrative road classes may be used in the different countries. In this case, roads with comparable importance shall have the same administrative road class value assigned. This requires leaving values unused in countries with a smaller hierarchy.

A road can have more than one administrative road class, for example, Europastraße (E) and Autobahn (A) in Germany. In this case, the class with higher importance from the view of the driver or on signposts shall be used. For Germany this would apply to the Autobahn (A), whereas for other countries it might apply to the Europastraße (E).

Traffic Sense

The traffic sense, meaning right or left hand traffic, is generally common for large regions. The predominant traffic sense for a region is defined by `hasRightHandDriving` (DataScript location: `nds.overall.metadata` > `RegionMetadata`). The attribute is part of the region-specific metadata and may have the following values:

- `TRUE` if right hand traffic is predominant
- `FALSE` for left hand traffic

For all links in a region with a different traffic sense than the one defined in the region-specific metadata, the flexible attribute `NON_DEFAULT_TRAFFIC_SENSE` (DataScript location `nds.common.flexibleattributes`) must be attached.

5.12 Referencing

After splitting features where necessary and generalizing data for higher levels, references must be set up between features on different tiles and on different levels. Additionally, links to other building blocks (Name and Basic Map Display) must be set up.

5.12.1 References between Features on Different Tiles

After splitting features, the split parts belonging to other tiles must be moved to these tiles. References then have to be defined in the original tiles. A reference is made up of the ID of the remote tile and the ID of the split part of the feature in the remote tile.

For more information, refer to *NDS – Format Specification*, 9.5.1 *Inter-Tile References* on page 136.

References between route links and the corresponding road geometry lines must be stored in the `Link2TileList` (DataScript location: `nds.routing.main`).

The list contains references to tiles, in which the road geometry lines belonging to a route link are located.

The list of tiles is ordered from the start to the end of the link.

The following applies for route links:

- The last entry in the list contains the tile ID where the link ends. This does not apply, if the route link starts and ends at intersections in the same tile.
- The list does not contain the tile ID where a route link starts, because the start intersection in this tile and even the end intersection can be found, if the route link also ends in this tile.
- If a route link intersects a tile several times, the tile is stored in the list only at the first occurrence with one exception: If a tile is not the start tile, but the end tile, and if this tile is intersected by the route link before, it occurs a second time at the last position.

The external tile list (`externalTileIdList`) is a collection of all referenced tiles in which the external feature IDs can be translated to indices of the inbound reference tables of the remote tiles. This step can be performed at the same time as building references to tiles on other levels.

5.12.2 References between Features on Different Levels

The references to features from higher to lower levels and vice versa can be set up during the generalization process. Links on lower levels sharing at least one intersection with a link on the next higher level must have a reference to this link in the `RouteUpLinkList` (DataScript location: `nds.routing.ref`) of the tile on the lower level.

For more information, refer to *NDS – Format Specification*, 9.5.3 *Inter-Level References* on page 140.

Links on upper levels must have references to all corresponding links on the next lower level in the route downlink table of their tile (DataScript location: `nds.routing.ref` > `RouteDownLinkList`). The references to the links on the lower level are sorted according to the direction of the referencing link from start to end. If the direction of a referenced link coincides with the direction of the referencing link, the referenced link is stored with positive direction. Otherwise, it is stored with negative direction.

To save space, some lower level links may be dropped from this list as long as their successors can be uniquely determined as follows:

- Each lower level link listed explicitly defines a sequence of lower level links with the same functional road class as the first one.
- The sequence is terminated when one of the following conditions is true:
  - The last link in the sequence ends at the target intersection of the corresponding upper level link.
  - The last link in the sequence ends at an intersection connected to more than two links of the given road class or with at least one link of a higher class.

5.12.3 References to Features in Other Building Blocks

The following sections describe how references to features in other building blocks are set.

For more information, refer to *NDS – Format Specification*, 9.5.2 *Inter-Building Block References* on page 137.

References to Features in the Basic Map Display Building Block

The compiler must deliver the geometry of base links and of road geometry lines and their map-relevant attributes to the higher levels of the Basic Map Display building block.

On higher levels, these features may be simplified by deleting, for instance, shape points. They may also be aggregated by merging features with identical map-relevant attributes to lines that can be drawn in one step.

During this process, it shall be tracked which road network features become part of which basic map display features in order to build references between the route links and map features in a later linking process.

References to Features in the Name Building Block

References to the named object features in the Name building block shall be provided. These references shall exist on all levels of the Routing building block. Otherwise, navigation applications could create route lists only on level 13 of the Routing building block.

If road features with different names have been aggregated to one link, the name references shall follow the direction of the aggregated link.

5.13 Processing Links at Update Region Borders

An NDS database may be split into several update regions to manage data sizes and to be more flexible with regard to update scenarios. This requires full connectivity across update region borders. This is guaranteed by so-called *gateways*.

For information on the NDS update region and gateway concept, refer to *NDS – Update Specification*, 3.2 *Update Region* on page 23 and *NDS – Update Specification*, 6 *Gateways Between Update Regions* on page 51.

It is recommended to use the partitioning schema of the input data from major map suppliers for the definition of update regions. For Europe, for example, the map data is typically partitioned based on countries. The update regions would therefore consist of one or more countries.

The major map suppliers deliver intersections on the borders of database partitions the same way, that is, duplicated. However, they deliver links at database partition borders differently: Whereas one supplier delivers links at database borders only in one database, the other supplier delivers them twice.

For processing links at database borders, the following aspects have to be considered:

- To avoid negative side-effects on route guidance, link duplicates at database borders shall be deleted if input data is partitioned, and is compiled and delivered in one NDS database. This also applies if input data of different update regions of the same NDS database is compiled separately.
- To avoid gaps, duplicated links must be kept if the input data is compiled separately per update region. This is particularly important if the update regions are delivered independently of each other. De-duplication is only possible during runtime.

For information on the NDS gateway concept, refer to *NDS – Update Specification*, 6 *Gateways Between Update Regions* on page 51.

5.14 Generic Profile Attributes

Navigation Data Standard provides generic profile attributes to represent properties of features that change over time. Generic profile attributes can be associated to links and to transitions of intersections in the Routing building block. This chapter introduces the topics that need to be considered to ensure interoperability.

For more information, refer to *NDS – Format Specification*, 9.7 *Generic Profile Attributes* on page 153.

5.14.1 Generalization

The concept for generic profiles is designed for level 13 of the Routing building block. For links on a higher level that have been aggregated from multiple level 13 links, the individual profile attributes of those level 13 links have to be mathematically combined during compilation.

To generalize speed profiles, all individual speed profiles must be resampled, the timely order and the proportional length must be considered, and then a single new speed profile can be created.

For more information on generalization, also refer to Section 5.8 *Generalizing Features for Higher Levels* on page 77.

5.14.2 Time Zone

Time-related information which is stored in profile attributes always reflects local times, including daylight saving time regulations of the time zone in which the corresponding road element is located.

5.14.3 Time Completeness

To ensure that the time period for a profile attribute of a link or a transition is complete, an NDS database supplier has the following possibilities:

- Delivery of detailed profile attributes for time periods where available
- Delivery of single average values for time periods where no detailed profiles are available

5.15 Advanced Driver Assistance (ADAS)

NDS allows two possibilities to store ADAS data:

- By means of shape points and ADAS attributes (`Curvature`, `SLOPE`, `PHYSICAL_WIDTH_METRIC`, and `PHYSICAL_WIDTH_IMPERIAL`)
- By means of an enhanced clothoid model with grade lines

The two concepts for storing ADAS data may coexist. NDS does not allow to mix the concepts, except for `SLOPE` attribute points which may be stored together with clothoid geometry. Enhanced grade lines must not be stored without enhanced clothoid geometry.

For more information, refer to *NDS – Format Specification*, 9.6 *Data Structures for Advanced Driver Assistance (ADAS)* on page 142.

Rules for Clothoid Approximation

There are no rules for clothoid approximation with regard to tolerances, approximation method, or approximation quality specified for NDS.

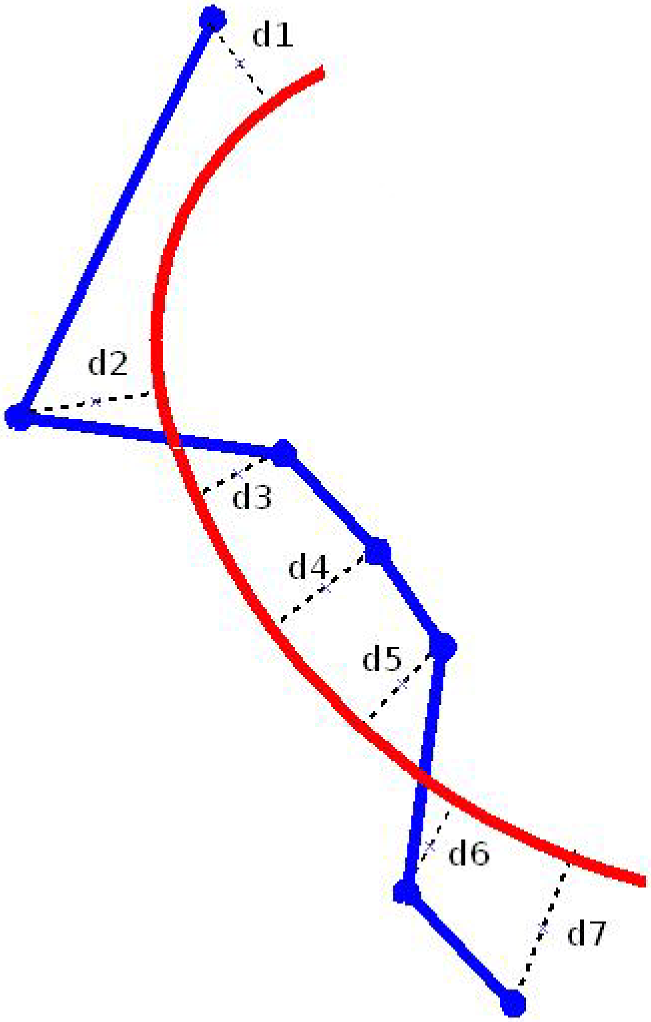
Rules for Grade Line Approximation

There are no rules for grade line approximation with regard to tolerances, approximation method, or approximation quality specified for NDS.

Standard Deviation and Degree of Freedom

The calculation of standard deviation is described in the *NDS – Format Specification*, 9.6.2 *Enhanced ADAS Geometry* on page 145. The figure below shows how to measure d_i values of shape points. The degrees of freedom in the example is `f = n–1 = 6`.

Figure 5-7 Perpendicular distances of shape points to the road center line (plane view)



Clothoid and Grade Line Consistency

ADAS only needs relative paths in front of the vehicle and a short track behind the vehicle. Therefore, an exact fitting of the whole road network is **not** required for clothoids and grade lines. If needed, inconsistencies must be compensated by positioning or by the horizon provider.

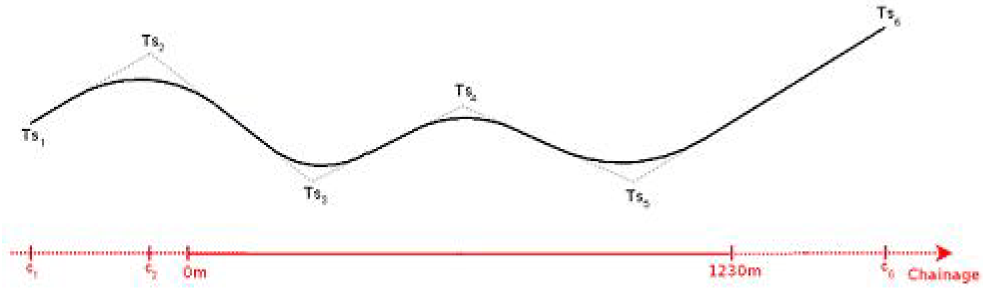
Chainage on the extension of a Road Center Line

For grade lines with vertical curves at their beginning or end, tangent intersection points on the extensions of the road geometry are required for a complete definition of the grade line.

The example in Figure 5-8 shows a road center line with a length of 1230 meters. To define the complete grade line, three tangent intersection points on the extension of the road center line are required: Ts_1, Ts_2, and Ts_6. To store the complete grade line definition, set a negative chainage offset in the grade line metadata: The offset is exactly the negative chainage c_1 of Ts_1.

Note A valid plane geometry on the extensions of a road center line is not required.

Figure 5-8 Grade line with tangent intersection points (profile view)



6 Eco Routing

Eco routing offers drivers an option to calculate an eco friendly and economic route that minimizes fuel consumption and emissions. This section introduces the topics that must be considered to ensure interoperability for eco routing. It contains information about eco routing calculations and explains its concepts and principles.

All data used for eco route calculation must be present in the Routing building block on the levels relevant for routing, meaning level 13 and upper levels. Some of the required data is already available in the database, such as the average speed. Other data, which is specific for eco routing, is stored for each link. Besides the costs related to link-based data, it must be possible to model the extra fuel or energy consumption caused by intersections. This concept is also used in fastest route calculations where penalties at intersections reflect the special obstruction effects there. The necessary information must be available on all routing levels from the most detailed up to higher aggregation levels.

For more information about the NDS data structures for eco routing data, refer to *NDS – Format Specification*, 9.9 *Data Structures for Eco Routing* on page 159.

Compiled databases should contain general attributes and not be vehicle-specific. Vehicle-specific details may be included at runtime using vehicle-specific parameters and lookup tables. This is described in the following sections.

6.1 Eco Route Calculation

Eco route calculations should avoid highly scientific and complex calculations, be adaptable to variations in the data availability, and be as general as possible. The calculations use the same principles as are used for finding the traditional fastest or shortest route. They are based on the following factors:

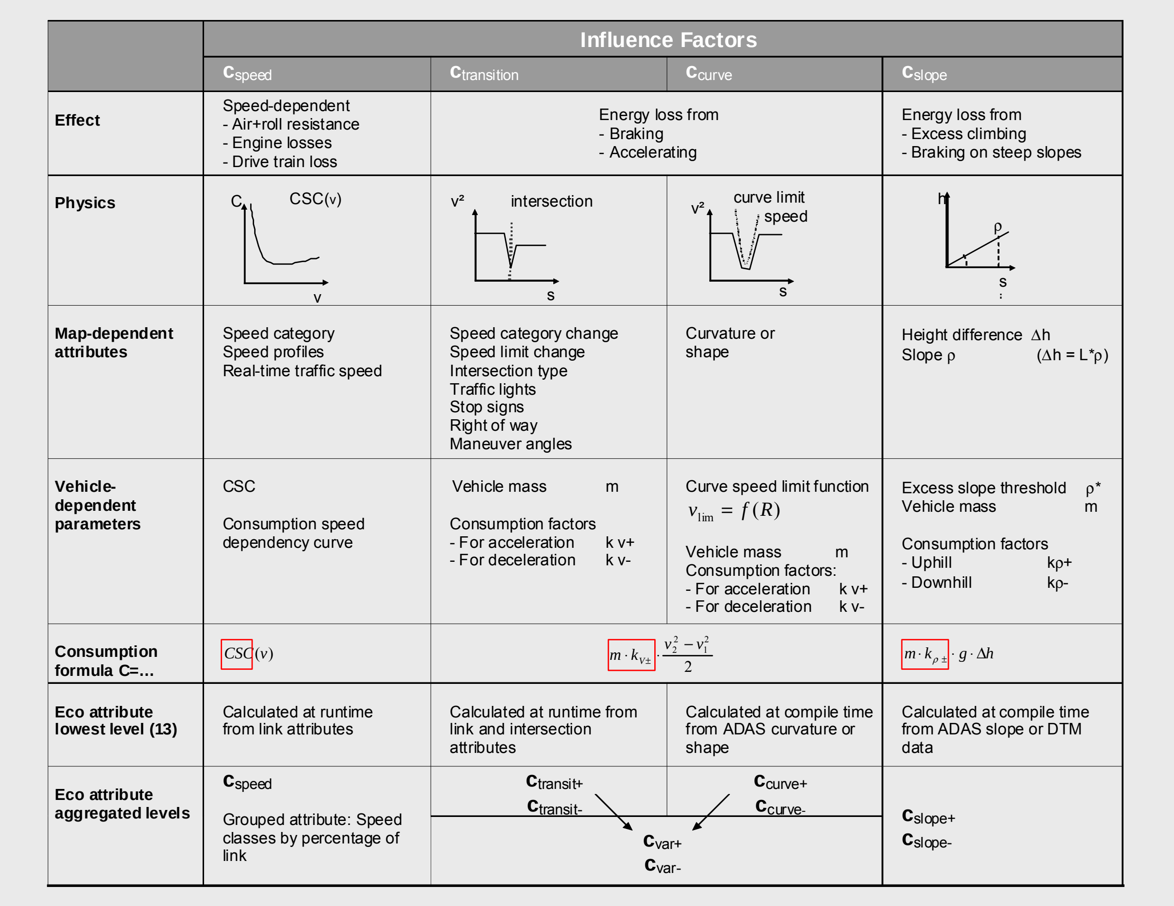
- Consumption Speed Curve (CSC): The speed dependency of fuel or energy consumption when driving at a specific average speed at a constant height and with normal traffic fluctuations.
- Speed variations: The influence of acceleration and deceleration on consumption due to transitions of road class, speed limits, sharp bends, curves or right-of-way regulations, such as traffic lights.
- Slope: The effects of ascending or descending slopes.

The data for these factors may come from one of the following sources:

- Map data: This data is generated by the compiler at compile time.
- Vehicle-specific data: This data is added by the application at runtime by using, for example, files containing vehicle-specific parameters.
- Driver-specific data: The application provides this data at runtime by using, for example, parameters that are deducted from the driving behavior over time.

The columns in Figure 6-1 show an overview of the considered influence factors causing fuel and energy consumption. The rows show the underlying physics, the map attributes, and the vehicle-dependent parameters that are used. The consumption formula row shows the principle of how the effect is calculated at runtime from map data and vehicle-specific parameters (vehicle-specific = highlighted by red boxes). The last two rows show how the attributes are handled on level 13 and on aggregated routing levels.

Figure 6-1 Overview of fuel and energy consumption effects



	Influence Factors			
	C_{speed}	$C_{transition}$	C_{curve}	C_{slope}
Effect	Speed-dependent - Air+roll resistance - Engine losses - Drive train loss	Energy loss from - Braking - Accelerating		Energy loss from - Excess climbing - Braking on steep slopes
Physics	C CSC(v) v	v² intersection s	v² curve limit speed s	h ρ s
Map-dependent attributes	Speed category Speed profiles Real-time traffic speed	Speed category change Speed limit change Intersection type Traffic lights Stop signs Right of way Maneuver angles	Curvature or shape	Height difference Δh Slope ρ (Δh = L*ρ)
Vehicle-dependent parameters	CSC Consumption speed dependency curve	Vehicle mass m Consumption factors - For acceleration k v+ - For deceleration k v-	Curve speed limit function $v_{\lim} = f(R)$ Vehicle mass m Consumption factors: - For acceleration k v+ - For deceleration k v-	Excess slope threshold ρ* Vehicle mass m Consumption factors - Uphill kρ+ - Downhill kρ-
Consumption formula C=...	$\boxed{CSC}(v)$	$\boxed{m \cdot k_{V\pm}} \cdot \frac{v_2^2 - v_1^2}{2}$		$\boxed{m \cdot k_{\rho\pm}} \cdot g \cdot \Delta h$
Eco attribute lowest level (13)	Calculated at runtime from link attributes	Calculated at runtime from link and intersection attributes	Calculated at compile time from ADAS curvature or shape	Calculated at compile time from ADAS slope or DTM data
Eco attribute aggregated levels	C_{speed} Grouped attribute: Speed classes by percentage of link	$C_{transit+}$ $C_{transit-}$	C_{curve+} C_{curve-}	C_{slope+} C_{slope-}
		C_{var+} C_{var-}		

To calculate the consumption estimate for a specific route (C_{route}), a sum over the four contribution factors along the route must be calculated. This applies to links and intersections on the most detailed routing level and on the upper (aggregated) levels.

$$C_{route} = \sum_{\substack{i \\ links}} \left(a_{speed} \cdot c_{speed\,,i} + a_{curve} \cdot c_{curve,i} + a_{slope} \cdot c_{slope\,,i} \right) \cdot L_i \quad + \quad \sum_{\substack{j \\ junctions}} a_{transit} \cdot C_{transit\,,\,j}$$

The routing criterion C_{route} sums the first three contributions over all links in the route, while the transition penalties are summed over the junctions along the route.

The application-specific adjustment factors, a_{xxx}, can be used to achieve a driver-specific weighting of the components. The adjustment factors may be either predefined and fixed, or adjusted to the driver‘s behavior at runtime. The a_{curve} value, for example, can be increased for sporty drivers who are more likely to brake sharply before entering a curve while normal drivers may drive the same curve with almost no loss of energy.

The consumption components $c_{xxx,i}$ represent the different influences on consumption for each link i or each junction j with regard to the total route consumption. The total consumption contribution of links varies according to the length of the link.

To avoid wasting storage space in the database, the by-link consumption contributions are stored as their equivalent specific c_{xxxi} values (lowercase c). These specific c-factors are derived from the absolute C_i values (uppercase C) by normalization to link length L_i:

$$c_{xxx,i} = \frac{C_{xxx,i}}{L_i}$$

The overall route consumption C_{route} can then be calculated from the specific c factors, the traveled link distances L_i, and the junction contributions, each weighted by an adjustment factor (a_{xxx}).

Note	As the intersection factors cannot be normalized to length, they are coded as absolute contributions $C_{transit,\, j}$.

The consumption factors c_{speed}, c_{slope}, c_{curve}, and $c_{transit}$ are calculated on the basis of the relationship shown in the formula row of Figure 6-1. The formulas explain how c_{xxx} are calculated from vehicle-specific parameters (CSC, m, k_{xxx}) and map-based input. The following sections describe in detail how these factors are modeled.

Two of the consumption factors, $c_{transit}$ and c_{curve}, are based on speed variation effects modeled by the same formula. They can, therefore, be represented jointly by the c_{var} parameter. The other consumption factors must be used according to the driving situation:

- c_{var+} for acceleration
- c_{var-} for deceleration
- c_{slope+} for uphill tracks
- c_{slope-} for downhill tracks

6.2 Speed Factor c_{speed} and Consumption Speed Curve (CSC)

The Consumption Speed Curve (CSC) is the most important component of the consumption calculation: The speed-dependant factor c_{speed} represents the vehicle-specific consumption caused by roll and air resistance and is expressed as a non-linear function of the average driving speed of a vehicle. The function is stored in the application, for example, in form of a lookup table. The application uses the speed data assigned to a link from the database as input for the lookup table and calculates the speed-related fuel or energy consumption factor c_{speed}.

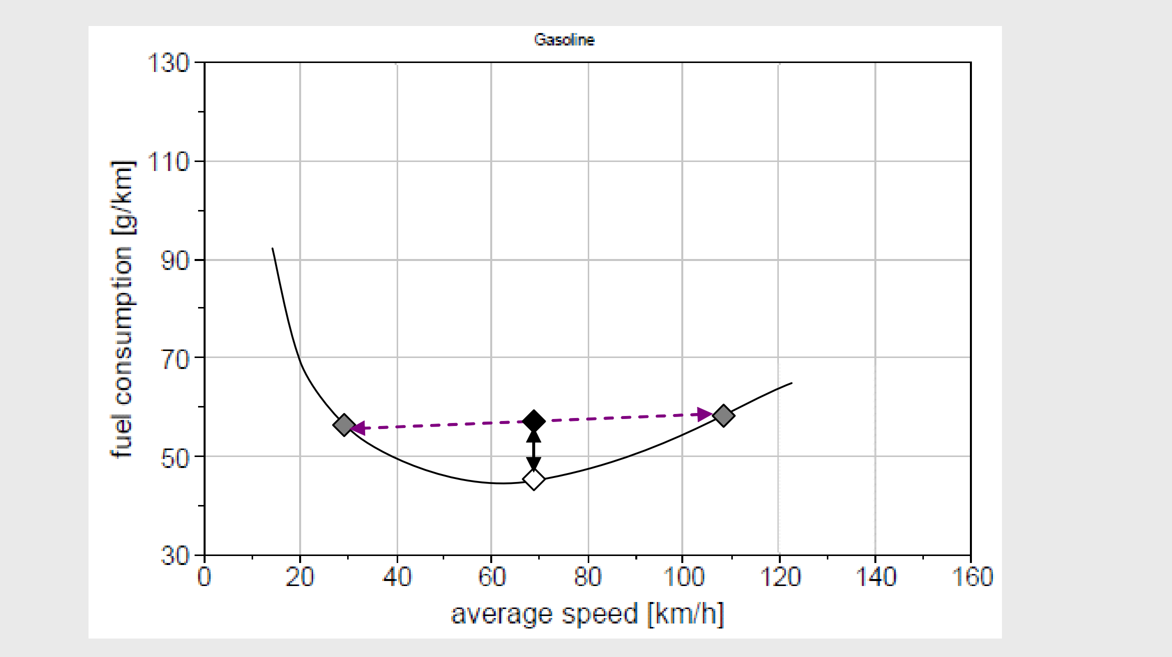
$$c_{speed} = \underbrace{CSC}_{vehicle}\overbrace{(v)}^{map}$$

The vehicle-specific Consumption Speed Curve (CSC) shall be available for calculation on both, the lowest (most detailed) and upper (aggregated) routing levels:

- On level 13, each link carries its own average speed attribute. The application can use the respective values directly for table lookup at runtime.
- On higher routing levels, links are aggregated. Due to the non-linearity of the CSC, taking the average speed for the aggregated link leads to wrong results.

Figure 6-2 shows a typical CSC for a combustion engine vehicle.

Figure 6-2 Non-linear CSC



Assume that an aggregated link consists of 50% city roads with an average speed of v=30km/h and 50% highway roads with an average speed of v=130km/h. This results in an aggregated average speed of 80 km/h. A lookup in the CSC table returns a consumption factor of ~43g/km (indicated by the white diamond mark in Figure 6-2). The actual consumption calculated from the two distinct values (left and right grey marks), however, results in a consumption factor of about 58 g/km (black mark on dashed line).

To correctly estimate consumption, NDS uses the aggregated attribute mechanism for links on upper levels. This mechanism allows storing different pairs of speed values with their respective weight factors. The compiler propagates the distribution of different speeds and weights through the aggregation level hierarchy and thereby allows vehicle-specific CSC calculation (at runtime) on all levels.

Table 6-1 shows a sequence of links that can be aggregated. The estimated average speed of a link in the second column is normally derived from the speed category for the link as retrieved from the map database.

Table 6-1 Example for collection of individual link data

Link	Average Speed ($\bar{v}_i$)	Length (L_i)	Relative weight (w_i)
1	30	1	10%
2	*50*	*0,5*	*5%*
3	80	1	10%
4	*50*	*1,5*	*15%*
5	*110*	*3*	*40%*
6	120	2	20%
7	*110*	*1*	*40%*
∑	**89**	**10**	**100%**

If all individual speed/weight pairs are propagated through the aggregation, they can be used for CSC calculation on a higher level. However, propagating the individual data pairs one by one would be inefficient, since speed values may appear several times (see the rows highlighted in italic, with v= 50 km/h and 110 km/h in Table 6-1). This leads to a high fragmentation, that is, a high number of attribute pairs. By putting all equal speed contributions together, a more compact representation is possible without losing information. This compact representation is shown in Table 6-2

Table 6-2 Example for collection of individual link data, compacted by the same speed values

Link	Average Speed ($\bar{v}_i$)	Length (L_i)	Relative weight (w_i)
1	30	1	10%
2+4	*50*	*2*	*20%*
3	80	1	10%
5+7	*110*	*4*	*40%*
7	120	2	20%
∑	**89**	**10**	**100%**

The results are equally accurate, since all the single original values are taken into account. The number of list entries is limited, because there are only a few distinct speed values and each of them is represented by a total (weight) percentage.

For map databases with a limited number of speed attribute classes (for example eight static speed classes), this approach is sufficient for the application to efficiently use CSC on aggregated levels.

For map databases with a high number of distinct speed values in the aggregation, however, this approach results in a high number of table entries and thus increases complexity of representing and handling the data. To avoid this, NDS recommends to group the aggregation data into a limited number of distinct velocity classes. The introduction of velocity classes reduces the number of speed/weight pairs required for aggregated links down to the chosen number of speed classes.

When the compiler merges a level 13 link with a single speed value v_j into an aggregated link on an upper level, it determines the speed class $[(v_i; v_{i+1}]$ to which the link belongs. All level 13 links of the same speed class are aggregated to a common average speed value $\bar{v}_i$ and a total class weight w_i. Both are calculated as a weighted average (by length) from the contributing level 13 links as follows:

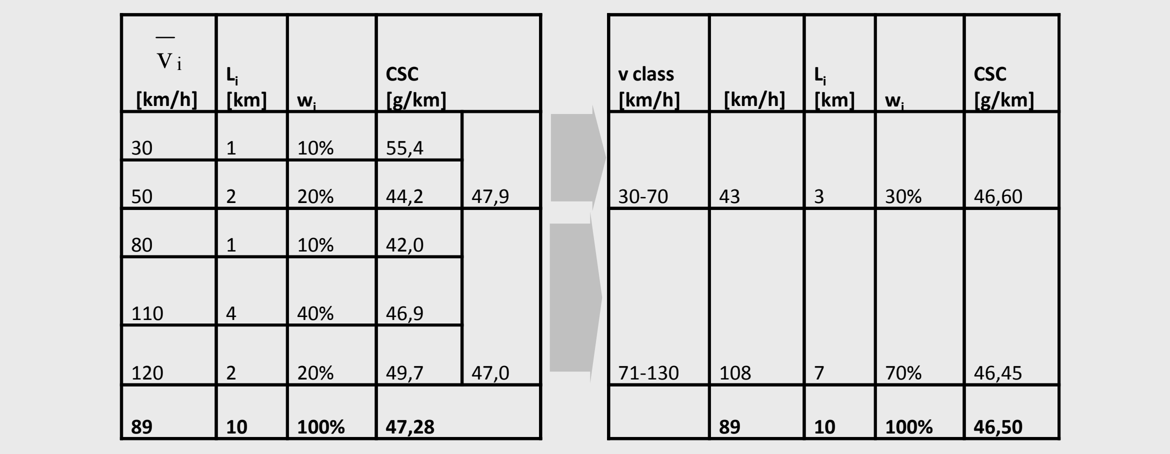
$$\bar{v}_i = \frac{\sum\limits_{\substack{\text{all links } j \\ \text{in speed class } i}} v_j \cdot L_j}{\sum L_j}$$

$$w_i = \frac{L_i}{L} = \frac{\sum L_j}{L}$$

When the compiler calculates $\bar{v}_i$, only those links L_i contribute to the speed class summation whose speed value v_i belongs to the respective speed class i. The value for w_i (= weight of speed class i) is calculated as the partial length L_i for that speed class, divided by the total length L of the whole aggregated link.

Figure 6-3 shows an exemplary aggregation by speed classes for the values in Table 6-2. The link references have been omitted, and the exemplary CSC values corresponding to the speeds are included. The original data on the left hand side has been grouped into two speed classes and processed to produce the compact list shown on the right hand side.

Figure 6-3 Further grouping by speed classes



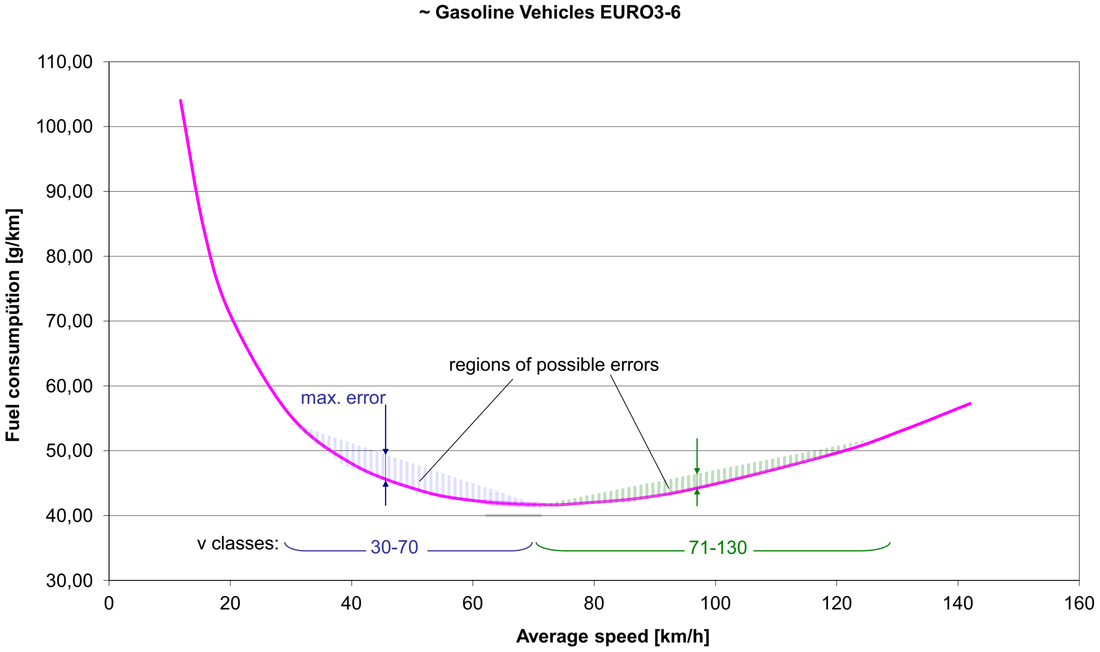
$\bar{v}_i$ [km/h]	L_i [km]	w_i	CSC [g/km]	
30	1	10%	55,4	47,9
50	2	20%	44,2	
80	1	10%	42,0	47,0
110	4	40%	46,9	
120	2	20%	49,7	
89	**10**	**100%**	**47,28**	

v class [km/h]	[km/h]	L_i [km]	w_i	CSC [g/km]
30-70	43	3	30%	46,60
71-130	108	7	70%	46,45
	89	**10**	**100%**	**46,50**

Each velocity class is represented by the class speed average value $\bar{v}_i$ and its partial weight w_i. The example shows, however, that the grouping by speed classes leads to information loss. The CSC consumption calculated on the basis of the grouped contributions on the right hand side gives different results from the original individual CSC lookup (see CSC columns on the right hand side). This loss is due to the nonlinear, curved appearance of the CSC and the deliberate choice of the speed classes.

Figure 6-4 shows the CSC used for the examples above and illustrates the approximation errors caused by speed class grouping. The maximum error margins for the two speed categories are indicated by the vertically hatched areas. These are maximum error bounds, and will not be reached under normal circumstances.

Figure 6-4 Speed classes and error margins from CSC calculation



The grouped speed attribution used with the CSC, is based on several $\{\bar{V}_i, w_i\}$ pairs for each speed class i. Links on upper (aggregated) levels carry a set of these pairs. You only need to register pairs with a weight $w_i > 0$.

Distribution and size of the speed classes may be freely chosen, as long as the nonlinearity of the CSC can be well approximated by using the chosen class distribution.

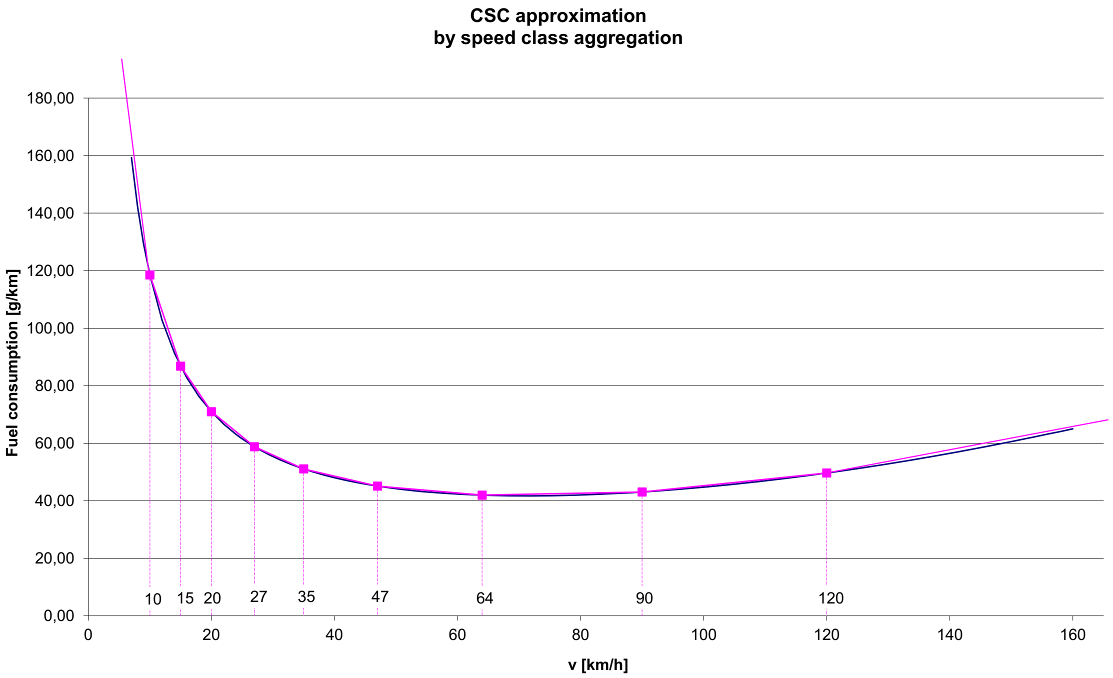
If the optional grouping into speed classes is used for the speed attribute aggregation, NDS recommends the class distribution shown in Table 6-3. This distribution uses a set of 10 speed classes and optimally represents the CSC taken from the *Handbook Emission Factors for Road Transport* (HBEFA).

Table 6-3 Speed class distribution recommend by NDS

Speed class [km/h]	0-10	10-15	15-20	20-27	27-35	35-47	47-64	64-90	90-120	>120

Figure 6-5 shows that the approximation deviation (dark curve) is rather small. The deviation should be roughly the same for any vehicle-specific CSC.

Figure 6-5 CSC approximation using aggregated speed classes



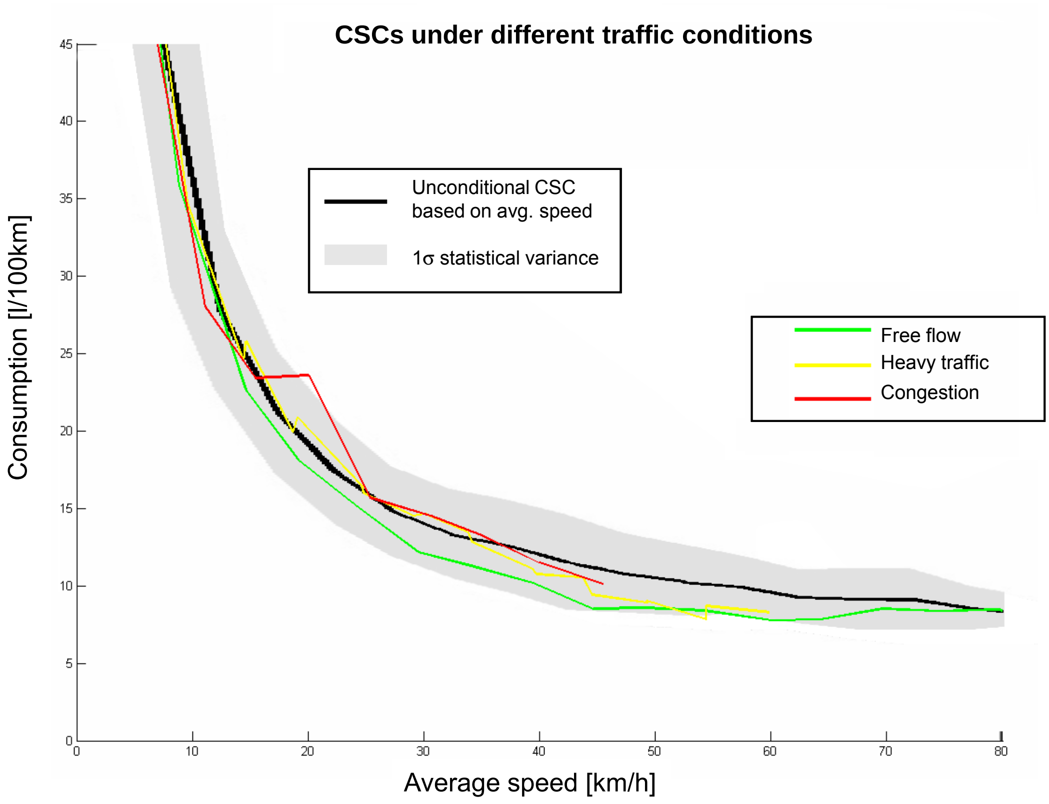
When using static map data, average speeds below 5 km/h or above 160 km/h are rarely encountered. The representation range is focused on speeds between 10 km/h and 160 km/h. However, it may be necessary to have more detailed classes at the low end, as real-time traffic data with very low average speeds may arise from traffic congestion.

Note NDS recommends this speed class grouping as it keeps a balance between representation accuracy and attribute group size. However, the number of speed classes used is up to the implementing company, which may also use less speed classes. Since the CSC is only a rough estimation of the fuel energy consumption with an uncertainty far above the shown approximation errors in Figure 6-5, a high approximation accuracy is not worth the effort.

6.2.1 Independence of CSC Effects from Other Influences

By comparing the results of driving tests under various traffic conditions, it was found that fuel consumption rates are almost independent of the traffic condition itself, whereas there was a high dependency on average speed. Figure 6-6 shows the results of these tests. The black line shows a measured CSC curve plus statistical variance, based on 600 kilometers of driving under general conditions. The other lines show the CSC under various traffic conditions as obtained from shorter test samples. This graph suggests that the average speed of a vehicle is the main parameter that determines fuel consumption under any driving condition.

Figure 6-6 Measured CSC under different traffic conditions



As a result of these tests, it seems appropriate to use the CSC as the single concept for modelling speed influences on fuel consumption, not depending on the source for the average speed data. The sources for average speed data may be:

- Static speed categories or speed limits
- Speed profiles (dependent on time of day/day of week)
- Speed data from real-time traffic feeds

6.2.2 Calculating c_{speed} with Speed Profiles and/or Real-time Flow Data

The most simple approach to calculate the speed-dependent consumption component uses the static speed information from the NDS map. This speed value can be used directly for a lookup in the vehicle-specific CSC function as described above. More complex handling is necessary, when dynamic speed information is also considered.

If additional data, such as real-time traffic data, is available for an eco route that has already been calculated based on the static average speed information, the corresponding route consumption estimates can be adapted according to the new data. This can be done as follows:

1. The application must identify the level 13 links belonging to the route and affected by the traffic event. It then needs to extract the corresponding static average speed and length data (v_i, L_i) from the database.
2. Using this input, the application calculates the original consumption estimates C_i for these links by using the CSC and then substitutes the calculated values by the corrected values C_i*. These values are calculated by using the up-to-date traffic speed data v* in the CSC lookup.
3. From the total route consumption (C_{Route}) calculated so far, the application can correct all erroneous components (marked by index i) by applying the corresponding consumption differences ΔC_i. The corrected total route consumption C^*_{Route} is the result of the operation:

$$\Delta C_i = C_i^* - C_i = \left(f_{\mathrm{csc}}(v_i^*) - f_{\mathrm{csc}}(v_i) \right) \cdot L_i$$

$$C^*_{Route} = C_{Route} + \sum_i \Delta C_i$$

There is another simple alternative way to make use of real-time traffic flow speed:

1. Identify the level 13 links and the aggregated (upper level) links affected by the traffic event.
2. Explode the aggregated links identified in step 1 into the corresponding level 13 links one-by-one and use the detailed speed data instead of the grouped speed data. Recalculate the route consumption using the level 13 real-time speed values v* where available.

The links with speeds that are changed due to traffic reports will directly influence the CSC consumption calculation on the basis of that reported speed. The total consumption C^*_{Route} of the route will then be correct as it is based on the corrected speeds from the dynamic traffic reports.

6.2.3 Consumption Calculation on Level 13

On level 13, a single 1 byte UINT average speed attribute $\bar{v}$ must be assigned to every link. This speed attribute is assumed to be valid for the entire length of the link. The routing application uses this value as input for fuel consumption lookup during runtime using the vehicle-dependent CSC.

6.2.4 Consumption Calculation on Higher Levels

On higher routing levels with aggregated links, the inside spread of different average speeds may be represented by a limited number of value pairs (average speed values and their corresponding weight factors). Both of these values are represented as 1 byte values.

The UINT8 value LINK_PERCENTAGE is used for the representation of the weight factor range $0<w\leq1$ by the following encoding:

LINK_PERCENTAGE = $w \cdot 256 - 1$

The decrement ensures that a factor w of 1.0 can be represented (resulting value of LINK_PERCENTAGE = 255), whereas the degenerate (unused) weight w=0.0 is out of the coding scope.

Coding of the average speed values $\bar{v}_n$ is according to the NDS-specific averageSpeed subtype.

Condensing the individual links into an aggregated (upper level) link is done during compilation. It is up to the compiler process to define the speed class representatives that reflect a meaningful spread.

6.3 Speed Variation

Speed or kinetic energy variations from acceleration and deceleration form the base for the two consumption factors $c_{transition}$ and c_{curve}.

$c_{transition}$ represents consumption effects caused by transitions between links. This can include:

- Speed changes from one link to the next
- Speed drops due to direction changes
- Speed drop due to right-of-way rules, either from traffic lights, crossing of main road with minor road, or other priorities derived from the road class

c_{curve} represents the consumption effects due to deliberate deceleration and acceleration caused by curves within a link.

6.3.1 Kinetic Energy Modeling

All changes in kinetic energy have an impact on fuel or energy consumption. The definition of kinetic energy is used as a common quantitative description metric for the different types of speed variation losses:

$$E_{kinetic} = \frac{m}{2} \cdot v^2$$

Transitions from one link to the next often involve speed changes. When the speed changes, there is a difference in kinetic energy. The change in energy can be calculated from the statistics of different speeds before and after the event. By removing the vehicle mass m from the kinetic energy equation, an energy factor can be derived as a purely map-specific figure that only depends on the speeds before and after the change:

$$\Delta E = \frac{m}{2} \cdot (v_2^2 - v_1^2) \quad \Rightarrow \quad \Delta e = \frac{v_2^2 - v_1^2}{2}$$

The Δe parameters for corresponding speed variation situations can thus calculated and stored in the map.

In the eco route calculation for a specific vehicle, the energy delta ΔE for the corresponding maneuver can be directly calculated by multiplying Δe with the vehicle mass m. ΔE directly corresponds to a consumed or recovered quantity of fuel or energy. The relationship between fuel consumption and kinetic energy change ΔE is modeled as a linear dependency, using an engine-dependent proportionality factor *k*. However, for deceleration and acceleration, two distinct vehicle-specific k-factors are introduced:

- k_{var+}, the acceleration consumption factor which measures fuel or electrical energy input per kinetic energy output in acceleration phases
- $k_{var–}$, the energy recovery factor, which quantifies the kinetic energy that can be fed back into energy storage during deceleration phases. The vehicle's kinetic energy existing at a point in time may be deliberately wasted by releasing it as thermal energy through braking (low k) or it can be partially recovered (high k).

Both, k_{var+} and $k_{var–}$ are positive numbers. An acceleration phase with $v_2 > v_1$ (positive Δe) will result in a positive consumption contribution, whereas a slowdown phase with $v_2 < v_1$ (negative Δe) will result in a negative consumption, indicating recovery. However, to be physically meaningful, the recovery factor is limited to the range $0 \leq k_{var–} \leq k_{var+}$.

A value of zero would indicate complete loss of kinetic energy at slowdowns, whereas equal factors would indicate that energy necessary for speedup is completely recovered in a corresponding slowdown phase.

By observing the mentioned influence factors, the corresponding overall consumption effects of transitions and curves ($c_{transition}$ and c_{curve}) can be reduced into a map-based component and a vehicle-specific component. For both effects, the same equation structure results:

$$C_{transition} = \underbrace{m \cdot k_{\mathrm{var}\pm}}_{vehicle} \cdot \underbrace{\Delta e_{transition}}_{map} \quad ; \quad C_{curve} = \underbrace{m \cdot k_{\mathrm{var}\pm}}_{vehicle} \cdot \underbrace{\Delta e_{curve}}_{map}$$

In principle it would be possible to use different k-factors ($k_{transition\pm}$, $k_{curve\pm}$) for the two cases to reflect statistical differences in the braking and acceleration behavior for the two situations. This would require, however, to store separate values for the two effects through all aggregation levels of the database. To avoid this complex approach, one common pair $k_{var\pm}$ consumption factor should be used for both cases, treating both effects equally.

The consumption factors $k_{var\pm}$ and the mass *m* are vehicle-specific parameters, whereas the kinetic energy terms Δe (with mass factored out) are determined by infrastructure and are provided by the map.

At transitions there may be slowdown or speedup phases, or both. According to the sign of speed variation, different consumption factors must be used: k_{var+} for acceleration, k_{var-} for deceleration. At curves, it is assumed that the slowdown and speedup phases are equal.

When the navigation function calculates the consumption values during routing, the corresponding energy difference terms Δe from the map are multiplied by the vehicle-specific mass and consumption coefficients to provide the consumption estimate C.

The variable C then stands for the quantity of fuel or electrical power needed to accelerate from v_1 to v_2, giving a positive Δe:

$$C_{+} = k_{\mathrm{var}+} \cdot m \cdot \frac{(v_2^2 - v_1^2)}{2} \quad = \quad k_{\mathrm{var}+} \cdot m \cdot \Delta\, e \qquad \text{(Acceleration)}$$

For deceleration (=negative Δe), the possible energy recovery is similarly calculated, using a different consumption factor k_{var-} for the recovery process:

$$C_{-} = k_{\mathrm{var}-} \cdot m \cdot \frac{(v_2^2 - v_1^2)}{2} \quad = \quad k_{\mathrm{var}-} \cdot m \cdot \Delta\, e \qquad \text{(Deceleration)}$$

Since the v^2 difference is negative in a deceleration scenario, the consumption C will also be negative, indicating energy recovery.

For combustion engines it is technically impossible to recover fuel, but mechanical energy may be recovered. This may appear as fuel gain. The following example illustrates this: A vehicle slows down considerably because it approaches an intersection. With modern engine controls, this situation typically causes a fuel supply cutoff, which leads to zero fuel consumption. However, the vehicle keeps moving, consuming its own kinetic energy instead of fuel. In eco routing calculation, consumption for normal travel is charged to the c_{speed} component. As the true consumption for the approach is zero, this looks as if the fuel has been saved or recovered by the slowdown phase.

The k parameters are formulated as consumption factors. k_+ is the reciprocal of drive train efficiency η and energy content per unit fuel H_i:

$$k_+ = \frac{1}{\eta \cdot H_i}$$

From this equation, the specific consumption values c = C/L (consumption per travelled length) are derived by using the link length L as the distance reference. The energy factors stored in the map are combined at runtime with vehicle-dependent factors received from the vehicle parameter set:

$$c_\pm = \frac{C_\pm}{L} = \underbrace{k_\pm \cdot m}_{vehicle} \cdot \underbrace{\frac{\Delta e}{L}}_{map}$$

As shown in the last row of Figure 6-1, the two contributions to kinetic energy variation ($c_{transition}$ for aggregated links and c_{curve}) may be put together into one common speed variation factor c_{var}. On level 13, this factor reflects the curve effect only. On aggregated levels, the factor should combine both – the curve and intersection transition factors as shown in Table 6-4.

Table 6-4 Combination of curve and transition effects into one single speed variation factor

Energy factor	Level 13 links	Aggregated links
$c_{transition}$	Not applicable Since $C_{transition}$ refers to transitions between links, routing must generate the factor from attributes encountered at/around the transition at runtime.	All $C_{transition}$ factors between every two consecutive links from inside the aggregated chain are accumulated and then normalized, dividing the sum by the total chain length: $c_{transition} = \frac{\sum_{i=1,n-1} C_{i,i+1}}{\sum_{i=1,n} L_i}$ The result is then put together with c_{curve}, which is already a link attribute.
		⇓

Energy factor	Level 13 links	Aggregated links
c_{curve}	Attributed to the level 13 link i by accumulating all m single curve effects encountered along that link and normalizing the result to the length L of link i: $c_{curve,\,i} = \frac{\sum_{j=1,m} C_{curve,\,j}}{L_i}$	Accumulated c_{curve} of all level 13 links contained in the aggregated link Aggregation is done by weighted addition of the individual c_{curve} of all level 13 links, normalized to the total length: $c_{curve} = \frac{\sum_{i=1,n} c_{curve,\,i} \cdot L_i}{\sum_{i=1,n} L_i}$
	⇓	⇓
c_{var}	$= c_{\text{curve}}$	$= c_{\text{transition}} + c_{\text{curve}}$

6.3.2 Using c_{var} as a Common Representation for $c_{\text{transition}}$ and c_{curve}

NDS combines the speed variation factors $c_{\text{transition}}$ and c_{curve} to a common speed variation factor for storage economy and simplicity.

$$c_{\text{var}} = \underbrace{k \cdot m}_{vehicle} \cdot \frac{\Delta e}{\underbrace{L}_{map}}$$

The map-resident component of this factor is stored as the `CONSUMPTION_SPEED_VARIATION` attribute with links, referring to the `speedVariation` data type. Due to the statistical nature of the underlying effects, it is not necessary to use high resolution. In NDS, the factor is confined to a `UINT8` representation using a scaling factor of 128:

$$\frac{\Delta e}{L} = (v_2^2 - v_1^2) / L = \texttt{CONSUMPTION\_SPEED\_VARIATION}\ 128\ [m/s^2]$$

By normalizing the Δe (derived from v_2 differences) to the link length L, this factor is equivalent to acceleration by physical units. The maximum scaled value of 255 signifies that the vehicle would accelerate or decelerate with a rate of $1 m/s^2$ along the entire length of the link. Test evaluations have shown that 97% of the acceleration or deceleration events stay within this value range. In the rare case that c_{var} is ≥ 1 m/s^2, corresponding to a `CONSUMPTION_SPEED_VARIATION` ≥ 255, it can simply be limited to the maximum value without a major risk of error.

For normal driving profiles, all acceleration steps along a route have complementary deceleration steps. This includes the speed dips at curves, assuming that the driver brakes before the curve and accelerates to the same level after it. This acceleration/deceleration balance can also be assumed for the speed behavior at transitions.

With the assumption of this balance property for all speed variation events, every speed change along the route is seen by the vehicle both as an acceleration and a deceleration event. During routing evaluation, all Δe steps can thus be summed and multiplied by the acceleration and deceleration factors k_+ and k_-.

Consequently, the total consumption due to speed changes C_{var} can be written as a sum of corresponding speed variation contributions $c_{var,i}$ from every link i:

$$C_{\mathrm{var}} = \sum_i c_{\mathrm{var},i} \cdot L_i$$

Each of them consists of an acceleration (⁺) and a deceleration (⁻) component:

$$c_{\mathrm{var}} \cdot L = k_{\mathrm{var+}} \cdot m \cdot \Delta\, e^{+} + k_{\mathrm{var-}} \cdot m \cdot \Delta\, e^{-} = (k_{\mathrm{var+}} - k_{\mathrm{var-}}) \cdot m \;\cdot\; \Delta e_{\mathrm{var}}$$

Since acceleration and deceleration energy deltas are complementary, this can finally be reduced to the following single term: $\Delta e^{+} = - \Delta e^{-} = \Delta e_{var}$.

It is sufficient to take only the positive energy differences $\Delta e^{+} = e_{var}$, and multiply them with the difference of the vehicle-specific acceleration and the deceleration factors k_{var+} and k_{var-}.

Note The difference between the two consumption factors for speed variation (k_{var+} and k_{var-}) is always positive. That means that a speed decrease always accounts for an energy loss. Otherwise, it would be possible to gain energy by permanently varying the speed, which would then result in a perpetuum mobile.

As discussed above, the total route consumption calculation combines the two speed variation components for curves and transitions in one factor C_{var}.

$$C_{Route} = C_{speed} + \underbrace{\left(C_{transit} + C_{curve}\right)} + C_{slope} = C_{speed} + C_{\mathrm{var}} + C_{slope}$$

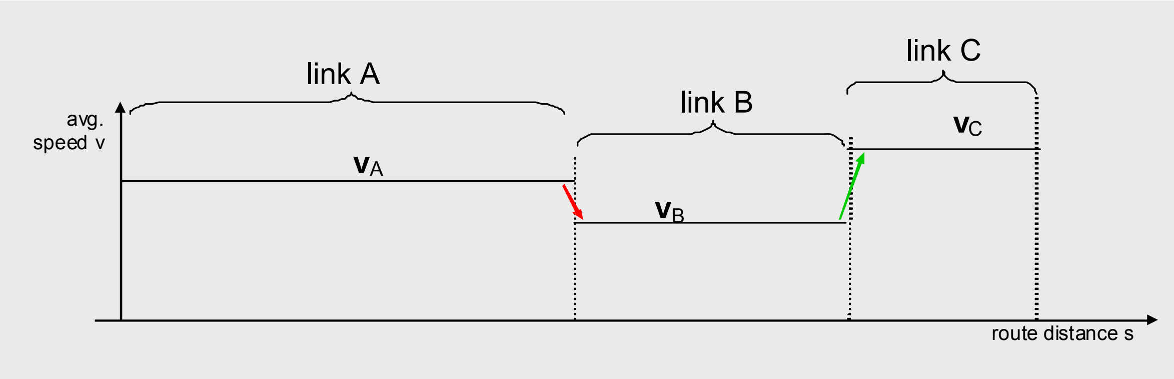
The following sections describe the basics and considerations for $c_{transition}$ and c_{curve}.

6.3.3 Speed Variation Factors for Transitions

The links of the road network are connected by nodes, which either correspond to intersections in the real world or simply indicate attribute changes (for example, the change of the average driving speed). Both transition types influence the energy consumption by property changes, such as the change of average speed or speed limit, which cause the need to decelerate or accelerate. Speed variation at intersections may additionally be influenced by right-of-way regulations and/or direction changes, forcing the driver to slow down.

In NDS, travelling speed information for each link can be either directly retrieved from the speed attribute or indirectly from speed limit, road class, and other attributes. The different speed values for two consecutive links can be used to calculate the influence on consumption. Figure 6-7 shows a sequence of links with two simple speed variations.

Figure 6-7 Sequence of links with simple speed variation



In the transition from link A to link B, there is a deceleration from V_A to V_B, and there may be an energy recovery with an electric vehicle, or at least an energy saving from coasting. In the transition from link B to link C, there is an acceleration from V_B to V_C involving energy consumption.

In addition to such simple speed variations, the following situations cause speed variation at transitions:

- Right-of-way rules and maneuvers
- Yielding right of way
- Turning maneuvers

Right-of-way Rules and Maneuvers

All situations with more than two streets connected to a junction may impact fuel or energy consumption. It is most likely that the driver has to decelerate in these situations, either to watch out for traffic or to give way.

The speed reduction can be estimated according to one or more of the following factors:

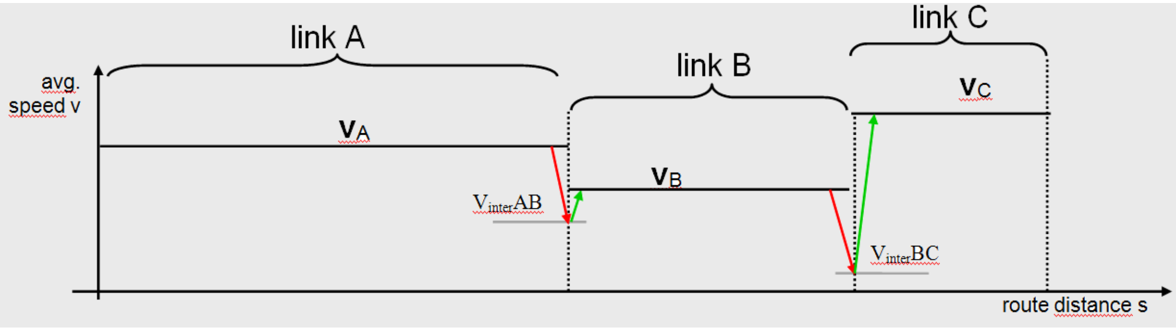
- Speed levels of incoming and outgoing links
- Explicit right-of-way regulations
- Implicit right-of-way effects, estimated by quantity and classification of competing links at the intersection

In these cases, it is assumed that the speed level drops to a minimum v_{inter}, which is modelled as a function of incoming and outgoing speed: $v_{inter} = f\ (v_{in}, V_{out})$. The resulting speed changes in a maneuver situation are decelerating from v_{in} to v_{inter} and accelerating from v_{inter} to v_{out}.

The speed minimum v_{inter} may be estimated from a value derived from right-of-way regulations and a minimum speed due to the angle change. Both parts separately would cause a speed reduction. For the calculation of the combined effect, the lower value for v_{inter} is used.

Figure 6-8 shows the speed reduction at an intersection that is below both speed values of the two adjacent links.

Figure 6-8 Sequence of links with speed reduction at intersections



Yielding right of way

It is assumed that the general obstruction caused at an intersection depends only on the situation when entering the intersection in relation to all concurrent links. In other words, the deceleration caused by giving way is independent of the following maneuver, which could be continuing to the left, to the right, or going straight.

If right-of-way regulations are available in the raw data, they can be used directly for the estimation of slowdown speed levels. If this is not the case, the influences of right-of-way regulations at an intersection must be estimated on the basis of existing NDS attributes, such as the following:

- Road class of links at intersection
- Total number of incoming links
- Single or multi-lane properties

In all cases, the encountered slowdowns due to these properties are subject to statistics, that is, sometimes the vehicle may pass without being affected, while other times it may be severely hindered by the situation. Which slowdown speed level v_{inter} is used at a given intersection should be derived from statistical estimates. These estimates should be based on extensive driving tests and measurements.

Turning maneuvers

If the vehicle must change driving direction at the maneuver point, the speed limitation for the turning maneuver is related to the angle change between the incoming and outgoing link. In other words, driving straight will cause less deceleration than turning right or left.

The influence of the turning angle on intersection speed v_{inter} is assumed to be a direct function of the angle change:

v_{angle} = f (turning angle)

where the turning angle is in the total range of –180° to +180° and function f is in numeric form or as a table lookup.

It is up to the compiler or navigation application to choose the most effective and realistic model for intersection slowdowns. NDS recommends that you make reasonable statistical evaluations for determining the corresponding functions and parameters to be used in the compiler and application.

6.3.4 NDS Database Storage Aspects for $c_{transition}$

This topic explains how the consumption effects of speed changes at link transitions are modeled in an NDS database on the different routing levels.

Routing Level 13

On level 13, transition effects are not attributable to links, since they are related to the intersection and involve the relationship between both the incoming and exiting link (bending angles, previous and consecutive speed levels). Corresponding costs should be calculated by the routing algorithm at runtime, using the specific transition conditions encountered at the intersections.

For this reason, the compiler will not evaluate transition contributions to links on level 13.

Aggregated Routing Levels

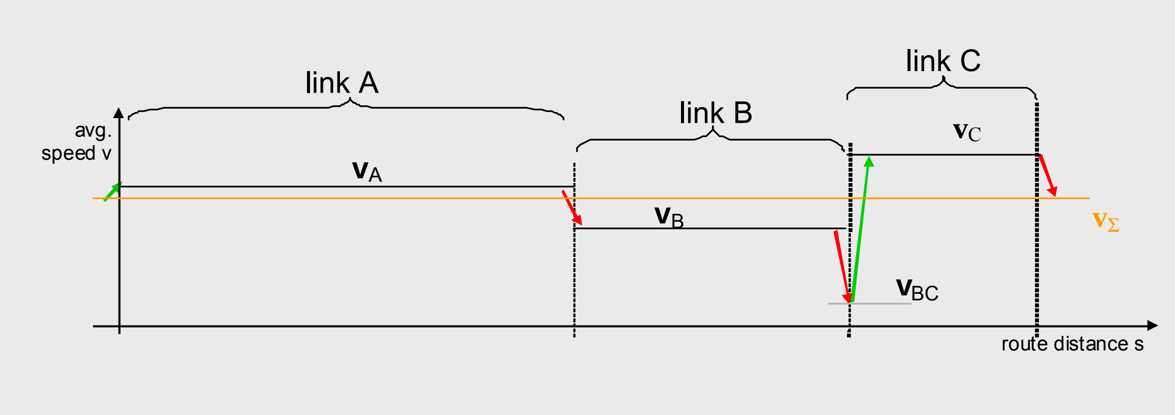
When aggregating several level 13 links to upper level links, the intersections become part of the upper level link. Consequently, energy effects due to acceleration and deceleration must be collected and summed up along the aggregated link. They are similar to the effects of a series of curves along a level 13 link, and they can be put together into the same speed variation energy factor as used for curves at level 13.

Symmetry

Considering the transition effects along a sequence of links, two symmetry properties can be used for storage efficiency:

- The sequence of accelerating and decelerating steps in one direction corresponds to an inverted sequence of decelerating and accelerating steps in the opposite direction with the same values and step sizes.
- Accelerating and decelerating while driving in one direction is balanced. This is the case when entry and exit speeds of the link sequence are assumed to be equal.

Figure 6-9 Link aggregation using average speed v_Σ (both for entry and exit speed)



Assuming that the sequence starts and ends on the identical speed level v_Σ, the sum of the acceleration steps is equal to the sum of the deceleration steps. This can be mathematically proven for any such sequence. For the example in Figure 6-9, this can be directly stated (acceleration steps on the left; deceleration on the right):

$$(v_A - v_\Sigma) + (v_C - v_{BC}) = (v_A - v_B) + (v_B - v_{BC}) + (v_C - v_\Sigma)$$

Even though there may be different conditions depending on both, driving direction and different entry and exit speeds, the symmetry concept is used to simplify calculations and save storage space. NDS recommends that the compiler assumes an identical entry and exit speed value, which equals the average speed value v_Σ between entry and exit speed of the aggregated link. This assumption ensures that the symmetry properties can be used and that only one speed variation parameter must be stored for both driving directions.

This property can be applied to:

- Curve effects (on level 13 and aggregated links)
- Transition effects (only on aggregated links)

This simplification can lead to inaccuracies in the consumption estimation. However, this error is probably very small compared to the overall fuel or energy consumption for an aggregated link.

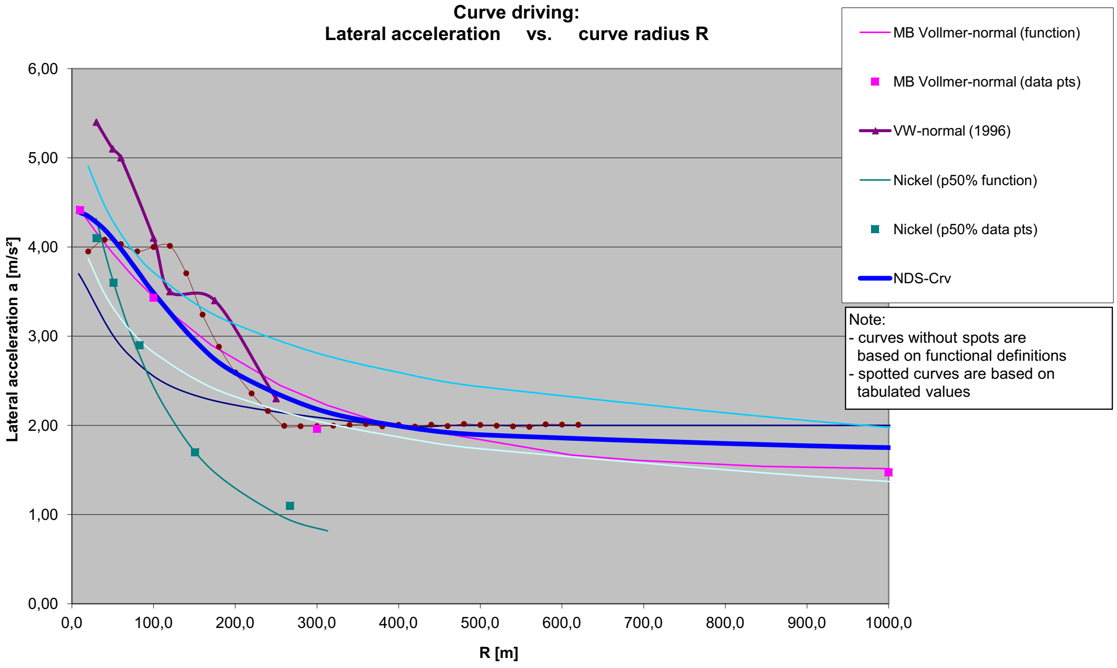
6.3.5 Speed Variation Factors for Curves

The influence of curves and the involved speed behavior is assumed to be similar to the $c_{transition}$ modelling. That means that – before entering a sharp curve – the driver may be forced to decelerate and then accelerate after passing the curve. Additional losses that occur by accounting for a drift angle >0 are not calculated, because they require driving close to the kinetic curve driving limits.

The curve influence modeling is based on the assumption that a normal road design should allow the vehicle to drive the link at its design speed v_{link} as derived from the link's speed category. It is also assumed that only few curves will force deceleration until below this design speed. To find the respective curves (at compile time), the compiler settings must define the curve speed threshold that should not be exceeded in order to keep the centrifugal forces within the driver's accepted comfort range.

Figure 6-10 shows the accepted lateral acceleration as a function of curve radius from different investigations.

Figure 6-10 Lateral acceleration over curve radius R (various sources)



Note:
- curves without spots are based on functional definitions
- spotted curves are based on tabulated values

NDS recommends to use the design function NDS-Crv which has been designed to reflect the plurality of the curves found in the different investigations (see the blue, bold curve in the diagram).

The formula representation of the NDS-Crv function is:

$$a_{lat} = \frac{(a_0 - a_\infty)}{1 + \left(\frac{R}{R_t}\right)^t} + a_\infty$$

Each parameter in the formula has a dedicated role regarding the design of the function. Table 6-5 provides the parameters used to produce the NDS-Crv curve shown in Figure 6-10.

Table 6-5 Lateral acceleration parameters

Parameter	Symbol	Unit	Value
Lateral acceleration at R → 0	a0	[m/s^2]	4.4
Lateral acceleration at R → ∞	a∞	[m/s^2]	1.7
Transition curve radius	R_t	[m]	140
Transition steepness	t	[ ]	2

Based on the function for the accepted lateral acceleration, a corresponding curve limit speed v_{lim} can be calculated. This is the speed that a normal driver would not exceed in a curve of radius R. v_{lim} is easily calculated from the centrifugal equation:

$$v_{\lim} = \sqrt{a_{lat} \cdot R}$$

By using these equations, the compiler can determine from the map data the maximum speed level v_{lim} for each curve and the normal speed v_{link} driven on the link. In a second step, the compiler estimates the energy loss by speed variation due to those curves and calculates the corresponding attribute values for the database.

6.3.6 NDS Database Compilation Aspects for c_{curve}

The compiler can use the following procedure to calculate the energy loss factor:

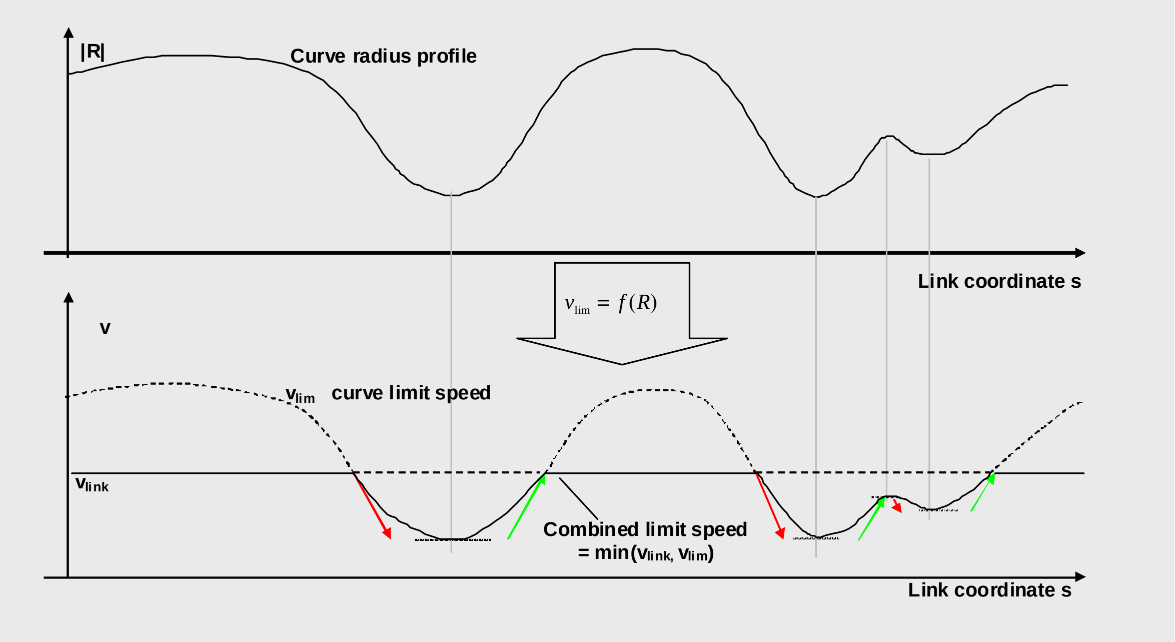
1. Take the design speed v_{link} from the map (speed category or speed limit setting of the link).
2. From the curve radius profile along the link, calculate curve limit speeds v_{lim} (local minima) for all curves along that link.
3. If $v_{lim} < v_{link}$, take $\Delta e_i = v^2_{link} – v^2_{lim}$ as curve loss measure of that curve 'i', otherwise, disregard the curve.
4. Sum up all curve-based energy losses along that link and normalize by length:

   $\Delta e_{curve} = \sum \Delta e_i / L_{link}$

This very simple approach suggests that every local curve speed minimum causes a slowdown from v_{link} to that local minimum v_{lim} and a corresponding speedup after the curve, from v_{lim} up to v_{link}, thereby always accounting for full sweeps between the two speed levels. This may not be realistic for many situations, for example, when curves follow in close distance to each other. As an alternative, more realistic approach, a more precise Δe_{curve} may be calculated by summing the differences between consecutive local maxima and minima, as illustrated in Figure 6-11.

Note The Δe parameter only represents the accumulated acceleration steps (green arrows pointing up). By the mentioned symmetry property, the accumulated deceleration steps must have the same magnitude, though with a negative sign.

Figure 6-11 Curvature profile along link and resulting curve limit speed profile



As in the simple approach, the radius profile (upper diagram) is evaluated for the link and transformed into a limit speed profile (lower diagram).

The curve loss parameters Δe_{curve} of the links are then calculated, scaled, normalized, and limited to the UINT8 representation of the CONSUMPTION_SPEED_VARIATION attribute, which is then stored for every link. The transition loss factors are added to the same attribute.

Driving tests have shown that losses for extreme curve driving are well described by the $v^2_{link} - v^2_{lim}$ model. However, curve loss figures depend on driver behavior and may even be very low. This is the case for cautious drivers, who slow down by idling instead of braking before sharp curves. This suggests that the application may reduce the curve consumption influence to realistic values by using a design factor, which may even be adapted to the driver behavior at runtime.

6.4 Slope

All changes of a vehicle's potential energy (=height) affect consumption. Just as kinetic energy, an increase of potential energy (ascending slope) or a reduction of potential energy (descending slope) has an impact on fuel consumption:

- Driving an ascending slope increases potential energy at the expense of propulsion energy. The energy consumption factor k_+ quantifies the consumed fuel or energy units per potential energy increase as a positive number.
- Driving a descending slope decreases potential energy. This energy is wasted through braking, recovered into electrical energy, or simply used to overcome driving distance with less motor energy. The energy recovery factor k_- quantifies the energy that is statistically recovered into electrical energy storage or is used for coasting.

As a quantitative description metric, the definition of potential energy is used:

$$E_{pot} = m \cdot g \cdot \Delta h$$

where E_{pot} quantifies the difference potential energy for a given height difference.

When stripping off the vehicle mass m and the gravity constant g from the potential energy term, as similarly proposed for the kinetic energy terms, a vehicle-independent loss factor Δe is calculated. This factor only depends on the height difference of the ascending/descending route:

$$\Delta E = m \cdot g \cdot (h_2 - h_1) \quad \Rightarrow \quad \Delta e = g \cdot \Delta h$$

This factor is the basis for the impact of slope in eco route calculations.

Similar to $C_{transition}$ and C_{curve}, the slope consumption factors extracted from the map must be scaled by the vehicle-specific mass and efficiency coefficients k_{slope} to produce an estimate for consumption due to slope.

$$C_{slope+} = m \cdot k_{slope+} \cdot g \cdot \Delta h \quad = \quad m \cdot k_{slope+} \cdot \Delta e \qquad \text{(Ascent)}$$

The consumption value C_{slope+} represents the quantity of fuel or energy needed to lift the vehicle mass m uphill by a height Δh. As with speed variation, the factor k contains all the parameters determining efficiency of energy transformation from fuel (chemical energy) to mechanical energy at the wheels.

For descending, the possible energy recovery is provided by an identical equation, using a different efficiency factor k_{slope-} for the recovery process:

$$C_{slope-} = m \cdot k_{slope-} \cdot g \cdot \Delta h \quad = \quad m \cdot k_{slope-} \cdot \Delta e \qquad \text{(Descent)}$$

However, as Δh is negative on a descending track, the consumption C may also be negative, indicating energy recovery.

Transforming C_{slope} into the specific consumption parameter c_{slope} (consumption per travelled distance) results in a relation where the slope ρ itself is the decisive variable instead of the height difference Δh:

$$C_{slope\pm} = m \cdot k_{slope\pm} \cdot g \cdot \Delta h$$

$$\Downarrow$$

$$c_{slope\pm} = m \cdot k_{slope\pm} \cdot g \cdot \frac{\Delta h}{L} = m \cdot k_{slope\pm} \cdot g \quad \cdot \rho$$

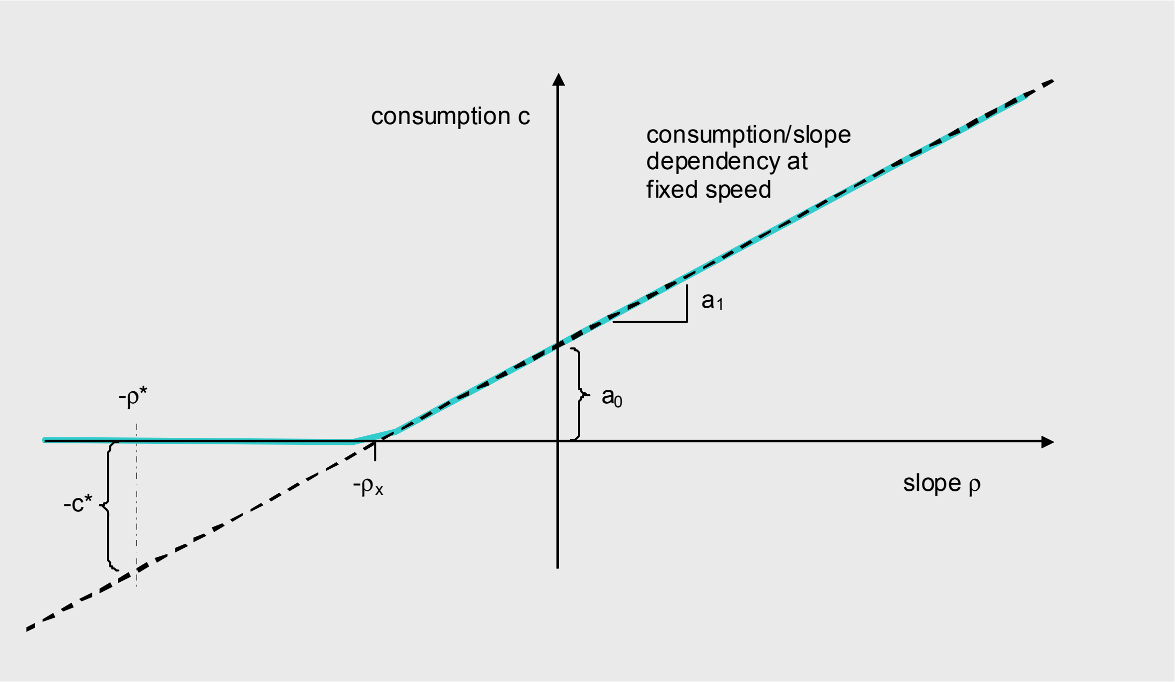
Where ρ is the average slope of a link of length L.

In the routing calculation, the slope value ρ coming from the map must be multiplied with a pre-calculated factor $m \cdot k_{slope\pm} \cdot g$ consisting of the vehicle-dependent parameters m, k_{slope} and the gravity constant g. If the vehicle mass is estimated at runtime, the parameters k, g (static) and m (online estimation) must be kept separate and only be multiplied at the moment of use.

6.4.1 Consumption Influence of Slope

Examination of the typical consumption behavior of combustion engines shows that there is an almost linear dependency between slope and consumption over a large operating range. Figure 6-12 illustrates a simplified slope consumption characteristic for a combustion engine vehicle.

Figure 6-12 Simplified slope consumption dependency for a combustion engine at constant speed



You can observe the following properties:

- At zero slope, the function crosses the consumption axis at a_0. This is the consumption necessary to overcome driving resistances at a certain, constant speed (as can be read out from the CSC function for that speed).
- For increasing slope >0, a corresponding increase of consumption per distance will result. The increase is normally a linear function of slope (see equations below for the background).
- This quasi-linear behavior is encountered for moderate slopes, ranging from ρ_x (typically around –4% … –3% for normal speed levels) to practically all positive slopes normally encountered on the road.
- The linear part of the consumption function crosses zero at a certain descent slope ρ_x. When driving down the slope at a fixed speed, downhill forces and motion resistances are exactly balanced, keeping the vehicle neutrally at constant speed with neither motor propulsion nor braking necessary.

- When descending slopes are steeper than ρ_x, some form of braking (engine or wheel brake) is necessary in order to keep the speed constant. Of course, fuel consumption will be zero in these situations, so there is a second horizontal segment of the consumption function for extreme negative slopes < ρ_x (see left hand branch at C = 0 in Figure 6-12).
- For a given negative slope $\rho^* < \rho_x$, an amount of energy is wasted by braking. The amount of energy waste can be derived from Figure 6-12: For a given slope ρ^*, simply take the vertical distance from the axis to the dashed line (continuation of the consumption line on the right hand side). Refer to the sketched example at slope ρ^* and observe the consumption equivalent c* of the wasted energy.
- The consumption equivalent (–c*) has a negative sign, indicating that this energy may be recovered in an ideal case. With e-drives, electrical braking in generator mode allows to feed back at least part of that energy into the battery. Under such conditions, c* sets the theoretical maximum for energy recovery.

The linear increase on the right branch of the function reflects the c_{slope+} equation with the interpretation that the a_1 parameter of that linear graph directly corresponds to vehicle properties and the gravity constant g in the following equation:

$$c_{slope+} = m \cdot k_{slope+} \cdot g \cdot \rho \quad = \quad a_1 \cdot \rho$$

This slope consumption function consisting of two linear segments is a simplification of the actual situation since for extreme positive slopes the function may again deviate from the single linear aspect. However, such extreme slopes rarely occur so that modelling that nonlinear effect separately is not worth the extra effort.

6.4.2 Slope Applications: Eco Routing vs. Range Estimation

There are considerations and options for database reduction that depend on the type of application using the slope. The technology used for data collection, and the quality of the raw data can have a significant impact on how well the application works.

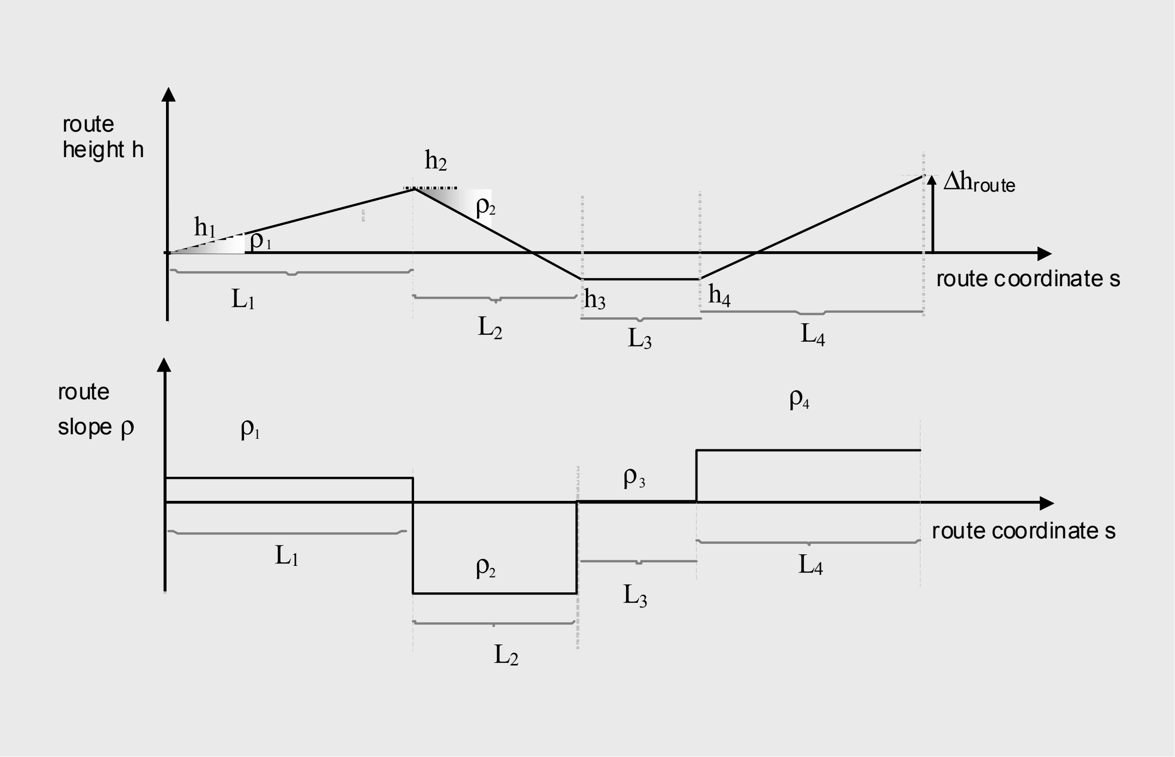
The application attempts to find the best eco route among a set of alternative routes. To find the optimal route, the absolute consumption estimates for the different routes are not important. The task is to compare routes, meaning that relative, not absolute consumption figures are important. Even though slope has a decisive influence on absolute consumption from route start to destination, its differentiating role in the choice between route alternatives is limited because all routes must overcome the same height difference from start to destination. Only the route parts with extreme slope will influence the routing decision.

From this, it may be concluded that only about 15% of the slope attributes – only the extreme slopes – are required for the routing task. As opposed to this requirement, range estimation requires consumption estimates at every point along a route.

Even if the source data would provide slope attributes on sub-link level (that is, a sequence of slopes within one link as is provided in the ADAS layer), eco routing needs the corresponding attributes as link attributes. This is required for performance reasons, as routing applications need to evaluate a high number of links in a short time.

In contrast to this, range estimation is only required for a limited number of possible routes, maybe only for a single route. This gives the application time to evaluate slopes on sub-link level. Irrespective of link or sub-link attribution, slopes along a route are represented by piecewise constant slope values, each valid for a limited *segment* of the route (see Figure 6-13). This segment may be a link or a sub-link.

Figure 6-13 Usual height and slope profile modeling with piecewise constant slopes



When calculating the consumption for driving along the route shown in Figure 6-13, the sum of all segment consumptions is calculated as follows:

$$C_{route} = \sum_{i=1}^{4} C_i = \sum_{i=1}^{4} c_i \cdot L_i$$

Take the specific consumption factors c_i as the linear function of slope according to Figure 6-12 with the slope ρ of segment (i) as the input. Assuming that the route only contains moderate slope values, the leftmost part (... < ρ_x) of the slope consumption characteristic can be disregarded. The linear slope consumption dependency is then described by the dashed line in Figure 6-12 on page 119.

$$c(\rho) = a_0 + a_1 \cdot \rho$$

This formula can be used to substitute the segment consumption factors c_i by a_0, a_1, and the respective slope values:

$$C_{route} = \sum_i c_i \cdot L_i = \sum_i (a_0 + a_1 \cdot \rho_i) \cdot L_i = a_0 \cdot \sum_i L_i + a_1 \cdot \sum_i \rho_i \cdot L_i = a_0 \cdot L + a_1 \cdot \Delta h_{route}$$

The ρ_i L_i products (slope multiplied by segment length) result in the individual height differences along the respective segments. The sum of these differences results in the total height difference Δh_{route} from start to destination.

The equation reduces to the two simple terms on the right hand side:

- In the first term, a_0 is the specific consumption for ρ=0 (i.e. on the horizontal plane). This parameter depends on driven speed and is derived from the well-known CSC function.
- L is the total length of the route.
- In the rightmost term, a_1 is the steepness of the linear consumption/slope characteristic (dashed line in Figure 6-12). This parameter is multiplied by the sum of the slopes, that is, the height difference from start to destination.

Since start and destination are identical for all route alternatives, they all have the same height difference term a_1 Δh. When comparing the consumptions of two alternative routes, this latter term will cancel out, so that differences in route consumption may only originate from the first term: route length and CSC influence. This reduces the problem to the classic eco routing calculation assuming horizontal driving (no slope influence).

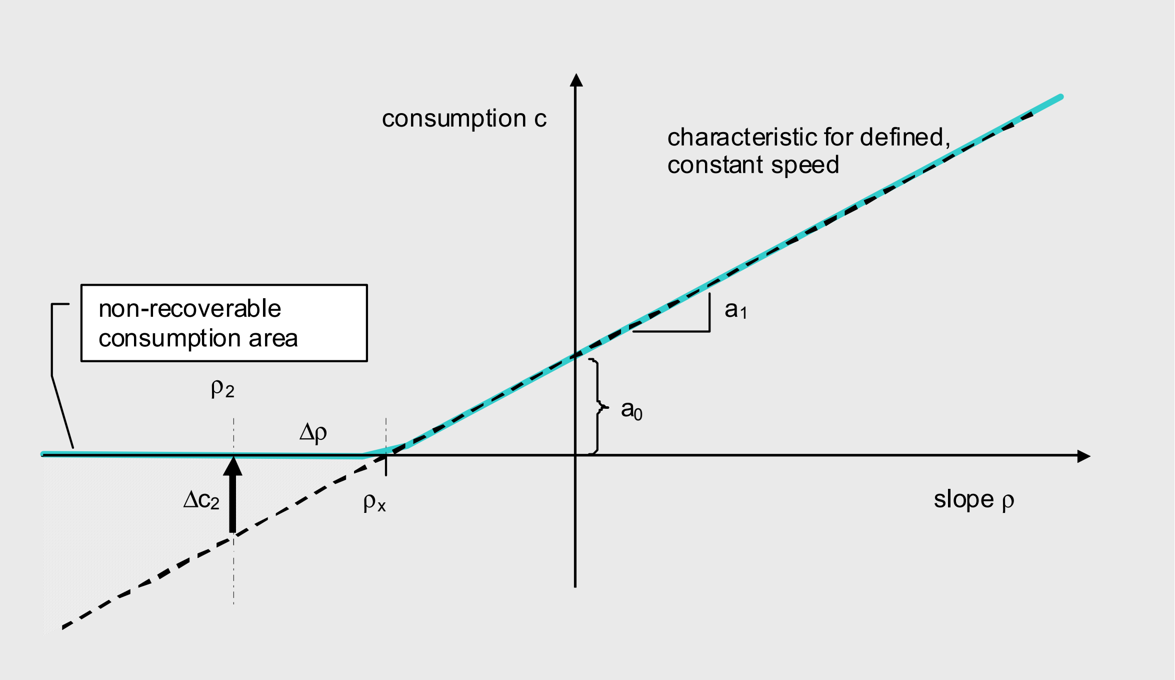
As a conclusion, slope does not have a decisive influence on eco routing if the route only contains moderate slopes. In this simplified case, physics show that the extra consumption that is invested for moving the vehicle uphill is not lost, but transformed into potential energy, which in turn allows lossless recovery in the subsequent downhill driving.

6.4.3 Excess Slope Attribute

As opposed to the conditions under moderate slopes, consequences are different for steep descent slopes <ρ_x (see Figure 6-14). For combustion engine vehicles, the consumption slope function remains zero in that range. The fact that the function does not follow the ideal (dashed) line below the axis is responsible for overall energy losses as outlined in the explanation of Figure 6-12. When descending such excessive slopes with a combustion vehicle, energy must be wasted by braking. This results in the triangular "non-recoverable consumption area" as shown in Figure 6-14.

Go back to the route consumption example in Figure 6-13 and assume that there is an excess descent slope in the second segment, meaning that ρ_2 is below the ρ_x threshold. The steep descent ρ_2 is such that recovery of Δ_{c2} is not possible as it would be with an ideal characteristic (dashed line).

Figure 6-14 Area of non-recovery for combustion engines (hatched triangle for slopes below x)



For this second segment, consumption would increase by Δc_2 with respect to the ideal characteristic. This leads to an extra penalty term $\underline{\Delta c_2 L_2}$, which accounts for the missed energy recovery that would occur in an ideal case (dashed line):

$$C_{route} = \ldots = a_0 \cdot \sum_i L_i + a_1 \cdot \sum_i \rho_i \cdot L_i + \underline{\Delta c_2 \cdot L_2} = \quad a_0 \cdot L \quad + \quad a_1 \cdot \Delta h_{route} \quad + \quad \underline{\Delta c_2 \cdot L_2}$$

By analyzing the slope consumption characteristic, you find that the excess consumption Δc_2 along segment 2 is easily calculated from the excess slope $\Delta\rho$ by which ρ_2 exceeds the threshold value of ρ_x:

$$\Delta c_2 = a_1 \cdot \Delta\rho = a_1 \cdot (\rho_x - \rho_2)$$

As this can be done for every occurrence of excess down slopes steeper than ρ_x, the eco routing application only needs to take negative slope into account in the measure by which it is exceeding the ρ_x threshold. As already stated, height differences between start and destination that are traveled at moderate slopes cancel out in the comparison of two route alternatives. The comparison of the route alternatives A and B shows this:

$$\begin{aligned} C_{route\,A} &= a_0 \cdot L_A + a_1 \cdot \Delta h_{route\,A} + \Delta c_A \cdot L_A \\ C_{route\,B} &= \underbrace{a_0 \cdot L_B}_{\substack{\text{Horizontal route}\\ \text{consumption} \cdot \text{Length}\\ A \neq B\\ \text{different}}} + \underbrace{a_1 \cdot \Delta h_{route\,B}}_{\substack{\text{moderate slope}\\ \text{influence}\\ A = B\\ \text{identical}}} + \underbrace{\Delta c_B \cdot L_B}_{\substack{\text{excess slope terms}\\ \\ A \neq B\\ \text{different}}} \end{aligned}$$

For eco routing as a decision between route alternatives to the same destination, it is not necessary to consider slope data as long as the slopes are moderate – only the excess slope parts (and of course length, CSC etc.) will make a difference in the fuel or energy consumption. Consequently, around 85% less database space is required for the slope attributes when only the links having excess slope portions are attributed. Only around 8% of the links in normal terrain and up to 30% in mountainous regions must carry such an excess slope attribute.

6.4.4 Average Slope Attribute for Range Estimation

The storage-efficient excess slope concept can only be used when the absolute consumption along a route is not of interest, as is the case with eco routing. For range estimation, the absolute consumption C is estimated as a function of distance along the intended route. In the cumulative consumption calculation, all three terms must be carried along, including the excess slope term.

$$C_{route} = \underbrace{a_0 \cdot \sum_i L_i}_{\substack{\text{Horizontal route}\\ \text{consumption -- length}\\ \text{CSC-based}}} + \underbrace{a_1 \cdot \sum_i \bar{\rho}_i \cdot L_i}_{\substack{\text{Average slope}\\ \text{influence}}} + \underbrace{\sum_i \Delta c_i \cdot L_i}_{\substack{\text{Excess slope}\\ \text{terms}}}$$

In a typical range estimation, the application sums up the terms incrementally while advancing along the route – link by link – until C_{route} exceeds the current fuel reserve. The incremental procedure requires that the corresponding average slope $\bar{\rho}$ for every single link is retrieved from the map. Even if this is a moderate slope, it must be taken into account, since it contributes to the absolute consumption.

6.4.5 NDS Database Representation for c_{slope}

Even though high quality slope data may already be present within the ADAS layer, it is too detailed (normally sub-link detail) and, therefore, not appropriate for rapid route calculations. Instead, the detailed slope data must be aggregated to attributes stored per link in the Routing building block.

The influence of slope on consumption is represented (per link) by:

- ρ_+/ρ_- pairs of excess slope factors (0 – 2 flexible attributes ... if $\rho_+ > 0$)
- One single average slope value $\bar{\rho}$ (mandatory attribute)

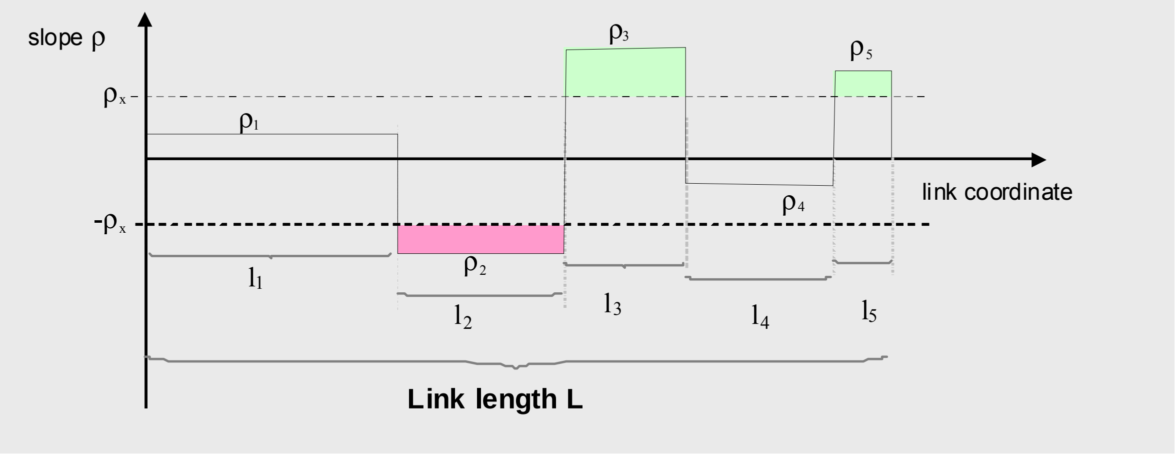
For eco routing, the attributes for excess slope factors are sufficient, whereas for range estimation, both the excess slope (if > 0) and the single average slope value (always) must be provided.

Excess Slope Attribute Definition

Excess slope ρ_{exc} is defined as the amount of slope, by which the neutral slope range $\pm\rho_X$ is exceeded in both directions. If the slope exceeds the upper threshold, you have $\rho_+ = \rho - \rho_X$. If the slope is below the negative threshold, you have the complement $\rho_- = \rho_X - \rho$, also registered as a positive number (see green or red areas in Figure 6-15).

Since basically only the excess down slope ρ_- affects consumption calculation, only the ρ_- component is required for the consumption estimation. However, when driving the same link in opposite direction, the roles of ρ_- and ρ_+ are inverted: The former ρ_+ is then a descending excess slope, and vice versa. This is the reason, why two excess slope parameters are attributed to the driving direction.

Figure 6-15 Link with inner detail slope profile



While the link shown in Figure 6-15 has an approximate average slope of around zero, its inner structure reveals positive and negative excess slope parts that should appear in the ρ_- and ρ_+ evaluations.

The compiler calculates the excess slope parameter $\Delta\rho_+$ for the link by aggregation of all green areas above $+\rho_x$, normalized to the whole link length L:

$$\Delta\rho_+ = (\rho_3 - \rho_x)\cdot\frac{l_3}{L} \quad + \quad (\rho_5 - \rho_x)\cdot\frac{l_5}{L}$$

This results in a positive number, stored as `UINT8`. In the same way, all the negative excess slope parts $\Delta\rho_-$ from the red areas below $-\rho_x$ have to be aggregated separately:

$$\Delta\rho_- = (\rho_2 + \rho_x)\cdot\frac{l_2}{L}$$

Note that $\Delta\rho\_$ also results in a positive number in this equation. This is intended, because ρ_+ and $\rho\_$ shall be stored as `UINT8` numbers in separate attributes. The sign to be applied to the excess slope attributes $\rho\_$ and ρ_+ is implicitly defined by their orientation (in-line vs. opposite to link orientation).

The accumulated excess slope attributes $\Delta\rho_\pm$ are the map-based part of the slope consumption estimation for eco routing. To calculate the corresponding excess consumption along a link for a specific vehicle, the excess consumption factor Δc must be evaluated (see Figure 6-14). This is simply calculated by

$$\Delta c = \Delta\rho_\pm \cdot a_1$$

since $\Delta\rho_+$ or $\Delta\rho\_$ already represents the direction-dependent sum of all excess slope parts, weighted by their respective lengths. This map-dependent Δρ figure is then mounted together with the parameter a_1 representing the vehicle-specific consumption slope dependency factor.

For every link, the excess slope factors $\Delta\rho_+$ and $\Delta\rho\_$ are calculated by aggregating the values of all sub-segment parts (i):

$$\rho_+ = \frac{\sum_i \max\{\,(\rho_i - \rho_x)\cdot L_i\ ;0\,\}}{\sum_i L_i}$$

$$\rho_- = \frac{\sum_i \max\{\,(\rho_x - \rho_i)\cdot L_i\ ;0\,\}}{\sum_i L_i}$$

These values are the weighted sums of the excess slopes (differences) of the parts exceeding the positive threshold ρ_x (case of ρ_+) and the parts less than the negative threshold $-\rho_x$, (case of $\rho\_$) taken as their absolute value.

These sums are then normalized by the total link length (that is, the sum of sub-link lengths).

Average Slope Attribute Definition

The average slope of a link is, according to its name, calculated either by averaging over all slopes, or simply by dividing the height difference Δh of the link in the positive link direction by the link length:

$$\bar{\rho} = \Delta \mathrm{h} / \mathrm{L}$$

Slope is usually coded as a dimension-free percentage value. The binary coding of the average slope attribute shall be equal to normal slope coding as `INT8`.

This is different to the excess slope `UINT` definition, as the average slope must be a signed number equally reflecting up and down slopes.

6.4.6 Compiler Aspects for c_{slope}

For slope data there is a major issue regarding the availability of map data. The very detailed ADAS slope data may be used for generating the per-link attributes and coefficients (average slope, excess slopes), but the ADAS quality data is often only available for high road classes.

Eco routing results based on a database that is only partly attributed (mainly on the higher road classes) could shift the route to the non-attributed lower classes, since the application could wrongly assume that missing slope attributes mean no slope. Unfortunately, the lower road classes usually are subject to higher slope values so that there is a risk of wrong routing results.

An alternative or complement to using ADAS data may be to derive slope data from coarse DTM sources. However, use of DTM data requires careful filter design in order to avoid artifacts caused by the DTM survey process or by the use of mixed slope sources (ADAS and DTM).

7 Name Building Block

The Name building block contains the name data for the features of Routing and Basis Map Display building blocks. Name data is used, for example, for location input, labeling maps, route guidance, and address retrieval.

7.1 Restrictions

There are some restrictions for the Name building block, for example, regarding maximum size. The compiler must be configured in such a way that these restrictions are adhered to. Note that the declaration of sizes always refers to uncompressed sizes.

7.2 Processing Sequence

The following processing sequence is recommended for name data:

1. Processing all named objects with their hierarchical relations to each other
2. Generating next-valid-character trees based on these hierarchical relations

7.3 Building Block-specific Metadata

The Name building block's metadata describes classifications of available named objects and how they are related to each other for location input.

For a detailed overview of the building block's metadata, refer to *NDS – Format Specification, Name Metadata* on page 173. The following topics are relevant for interoperability:

- Definition of name feature classes
- Definition of selection graphs
- Definition of selection criteria
- Collation rules
- Rotation metadata

7.3.1 Definition of Named Object Feature Classes

To ensure the referential integrity of a navigation database, the metadata must contain a list of all named object feature classes used in the Name building block including class definition. Also, the attribute types used must be defined for the named object feature classes.

7.3.2 Definition of Selection Graphs

The definition of a selection graph is mandatory. The selection graph definition must be complete and correct according to the conditions described below. It is recommended to fill the selection graphs during compilation.

For more information, refer to *NDS – Format Specification*, 10.4 *Selection Graphs* on page 190.

Reason for Introduction of Selection Graphs

The application checks whether the precompiled location input structures fit to its assumptions. Generic applications can adapt themselves to the precompiled location input structures (this is optional). The selection graph can be used for release notes and hints for the test of database content. The selection graph also gives information on NVC trees that use latinization for ideograms.

A Selection Graph Definition Must be Complete

A selection graph definition is complete if the following conditions are met:

- All named objects or name attributes (for example, house numbers) required for location input exist.
- All named objects can be accessed via a next-valid-character tree or by using a spatial selection of named objects and the relations between them. For more information, refer to *NDS – Format Specification*, 10.2.1 *Attributes of Named Object Features* on page 179.
- A reference to the region named object in the columns of the selection graph table exists. This information is used as a starting point for location input and is mandatory.

A Selection Graph Definition Must be Correct

Named objects and name attributes (for example, house numbers) that are defined together with their access paths must be represented by a corresponding selection criterion. Access paths can be next-valid-character trees, corresponding relations, disambiguation, and name attributes being assigned to features.

However, it is not necessary to have homogenous access structures. Alternate structures may be used, such as:

- Place > street > house number (for example, for cities)
- Place > house number (for example, for hamlets or city areas)

If a name format other than the default name format is used in an access path, this information must be noted in the corresponding edge of the selection graph.

Minimum Filling

At least the path country > city/city district > road > house number shall be available in the selection graph of each database and hence it will also be available in the respective location input data. City and city district may be merged into one selection criterion ("place") or provided as separate selection criteria.

Storage of Selection Graphs

Selection Graphs are stored per update region in the `SelectionGraphTable` (DataScript location: `nds.name.main`). The table contains one selection graph per region named object. Each selection graph is identified by an arbitrary integer number.

7.3.3 Definition of Selection Criteria

Selection criteria must fulfill the following requirements:

- A name for that criterion must be defined for all languages that are supported in the update region.
- At least one named object feature class must be assigned.

For more information, refer to *NDS – Format Specification*, 10.5 *Selection Criteria* on page 192.

Storage of Selection Criteria

Selection criteria definitions are stored per update region in the `SelectionMetadataTable` (DataScript location: `nds.name.main` > `SelectionMetadataTable` > `ndsData`). The table contains all selection criteria definitions used in all selection graphs of an update region. The `SelectionMetadataTable` has exactly one row, which is identified by an arbitrary artificial ID.

7.3.4 Collation Rules

Navigation Data Standard defines that all names are stored in accordance to the Unicode code point order.

Applications may store language-specific collation rules. Some update regions can contain characters which are not available in the languages supported by the application. The administrative area-specific metadata for an update region contains Unicode character code charts used in the respective administrative area, for example, Latin-1. Applications can use this metadata to identify the respective collation rule for an update region.

If strings with different languages/alphabets appear, the application shall sort by binary Unicode code point order.

7.3.5 Rotation Metadata

The rotation marker as well as the rotation separation string are defined as constants. They must not be stored as separate name strings and their use is restricted to next-valid-character trees.

For more information on rotation metadata, refer to *NDS – Format Specification*, 20.2.1 *Rotation Metadata* on page 353. For rotation in NVC trees, see Section *Rotation of Names* on page 139.

7.4 Allocation of Named Objects to Basic Map Display and Routing Levels

For route link features, references to road named objects on higher levels are mandatory to enable route lists and map display of road names. Also, references to house named objects on level 13 are mandatory for road geometry lines and base link features. It is not recommended to store them on higher levels.

Note Only a single mapping is valid for basic map display and routing features.

For basic map display features, references to map named objects are recommended on all levels to enable a map display including names.

Table 7-1 Allocation of named objects to levels, overview

Level of named object references	Map display (for basic map display)	Road (for road geometry lines and base links)	Road (for route links)	House (for road geometry lines and base links)
Higher levels	Recommended	Not available	Mandatory	Not recommended
Level 13	Recommended	Mandatory	Mandatory	Mandatory

Use Cases for Displaying Names

There are several use cases for displaying names:

- Always display official names and additional names in the user-selected language.
- Always display available official and alternate names.

In order to reduce ambiguity, the application has to filter out identical names.

7.5 Location Input

Location input is supported by the following data structures:

- Global entry
- Next-valid-character trees
- Geographical extent attribute
- Follow-up relations
- Full-text search (not yet relevant)

7.5.1 Global Entry

The location input starts with the regions defined in the selection graph table rows. This list of region named object references is the only entry point to the database for hierarchical location input. This entry point may have the named object feature class “root”. This class is used if the name of the initial region named object is a technical name and is not to be displayed to the user.

Example The initial object represents the update region BENELUX, however, this name is not to be shown to the users.

To keep the list of regions consistent, it shall only contain similar named objects. For example, it is not allowed to mix district and state named objects, but it is allowed to mix countries and supra-national areas.

Note NDS does not provide data structures for global NVC trees. Global NVC trees refer to NVC trees covering more than one update region, for example, a city NVC for Europe assuming that Europe is separated in several update regions (one per country, for example). The existing NVC trees can be merged at runtime, see *Merge NVC Trees* in the NDS Wiki.

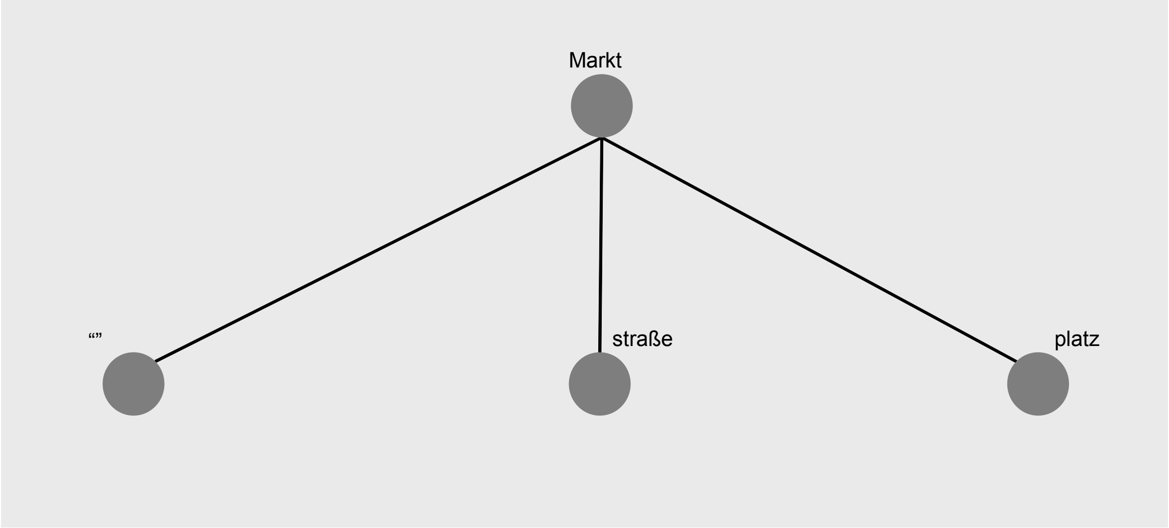
In the example, the supra-national area named object (Europe) forms the global entry to the selection graph. As other update regions may contain the same named object, the identity can be identified by name and classification. This could also be done by just using a named object "root", but then a unique identification would not be possible and the name "Europe" could not be displayed.

7.5.2 Next-valid-character Trees

The following rules must be observed for next-valid-character (NVC) trees:

- Sibling nodes shall differ in their continuation string. They may, however, have a common prefix. Sibling nodes are sorted in ascending order according to their continuation string. A common prefix is only allowed for leaf nodes.
- A continuation string may be empty (see Figure 7-1).
- Leaf nodes always refer to named objects or POIs. The named object references shall be identical with the named objects that are referenced by the link or road geometry line features on level 13.
- NVC trees must have a unique entry point.

Figure 7-1 Empty continuation string, example



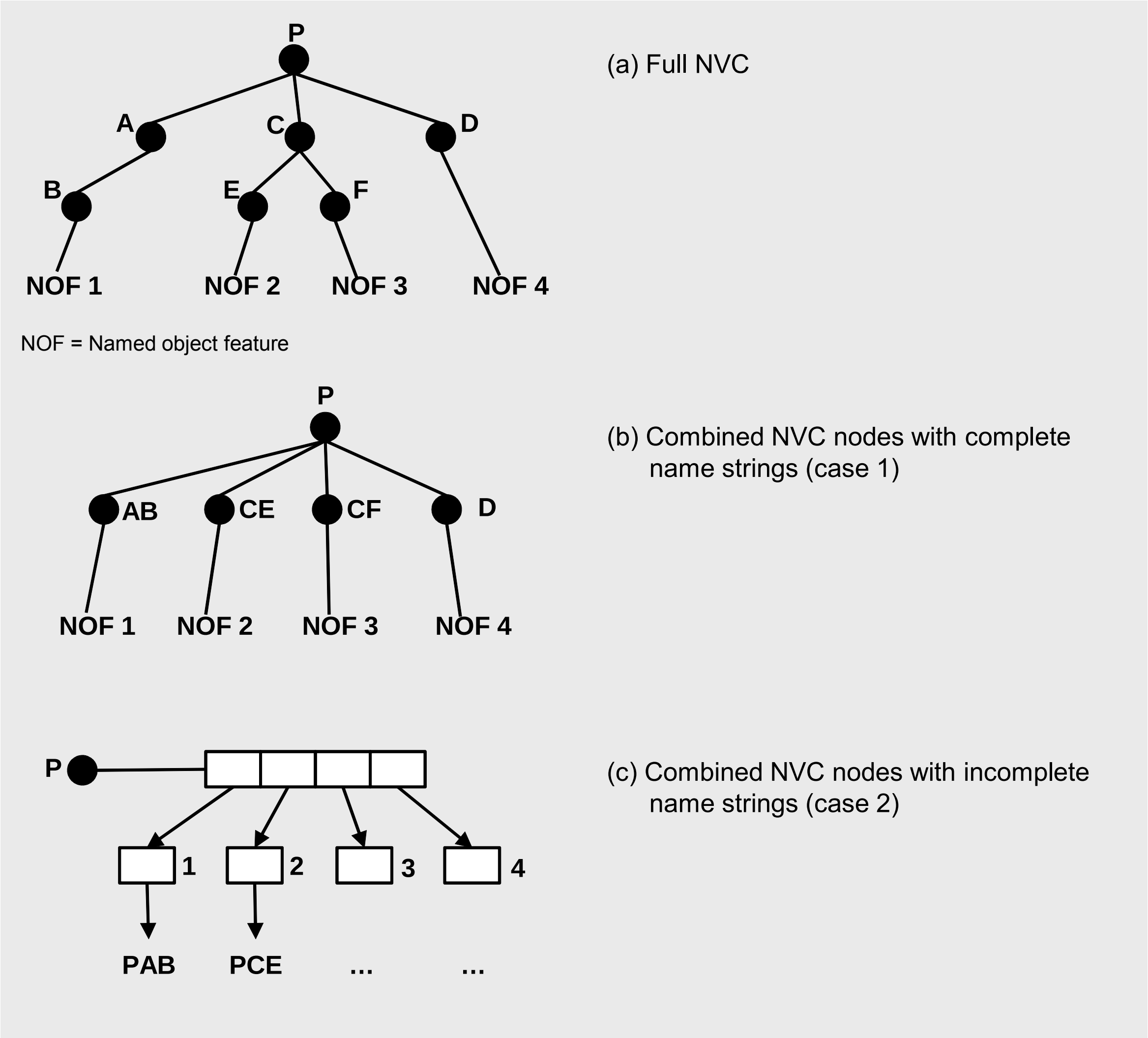
For more information, refer to *NDS – Format Specification*, 10.6 *Next-valid-character Trees* on page 194.

Optimization Options for NVC Trees

The following optimization options are possible for NVC trees:

- Full NVC tree: For size optimization, full NVC trees may have a single character per node and contain all strings completely (see Figure 7-2).
- Combined NVC nodes (with all and complete name strings): For size optimization, single nodes may be merged (see Figure 7-2 (b)) while keeping the complete strings. This includes also the option to combine intermediate strings, if a sequence of single nodes is available.
- Combined NVC node (with incomplete name strings): For performance optimization, NVC trees do not need to contain all strings completely. A subtree may be replaced by a leaf node pointing to a list of references to named objects (see Figure 7-2(c)). If such a list is available, then an application does not need to generate the list form which may save computation effort. The size of these lists shall not exceed the number defined in Table 7-2 for case 2.

Figure 7-2 Full NVC and combined NVCs, example


P
(a) Full NVC
A
C
D
B
E
F
NOF 1
NOF 2
NOF 3
NOF 4
NOF = Named object feature
P
(b) Combined NVC nodes with complete name strings (case 1)
AB
CE
CF
D
NOF 1
NOF 2
NOF 3
NOF 4
P
(c) Combined NVC nodes with incomplete name strings (case 2)
1
2
3
4
PAB
PCE
...
...

The restrictions shown in Table 7-2 need to be considered for NVC trees. There are two main factors: Performance and RAM of the system (low, mid, high) and the storage medium used in the system. If the restriction values overlap, the lowest restriction value must be used.

Table 7-2 Restriction values for NVC trees

Restriction	Low	Mid	High
Maximum number of names for creation of the NVC node list for combined NVC nodes with complete name strings (case 1)	1000	5000	10000
Restriction	**DVD**	**HD**	**SD**
Maximum number of different named object BLOBs per leaf node for combined NVC nodes with incomplete name strings (case 2)	5	10	20

If an NVC tree node contains only a one-byte prefix character, which should be the default case, then the `isShort` flag in the tree node is set to `TRUE`. This makes it possible to use a single byte instead of an UTF-8 string for storing this prefix, in order to save space. This optimization will only work for languages with 1-byte UTF-8 characters.

Size and Partitioning

The partitioning of NVC trees enables applications to load several NVC subtrees (see restriction definition in Table 7-3). Each subtree is stored as a BLOB in one row of the NVC table. A subtree can be a root partition of a given NVC tree. In this case its ID must be set to `0`. (DataScript location: `nds.names.main` > `NvcTable` > `nvcSubTreeId`)

Note Partitioning of NVC trees is left to the compiling party. Linearization does not describe a required partitioning, but only the streaming of data for one partition. See also Section *Linearization of NVC Trees* on page 138.

Partitioning can be optimized, for example, for the following use cases:

- For fast access to first leaves and, thus, creating L shaped partitions
- For fast access, creating even-sized partitions

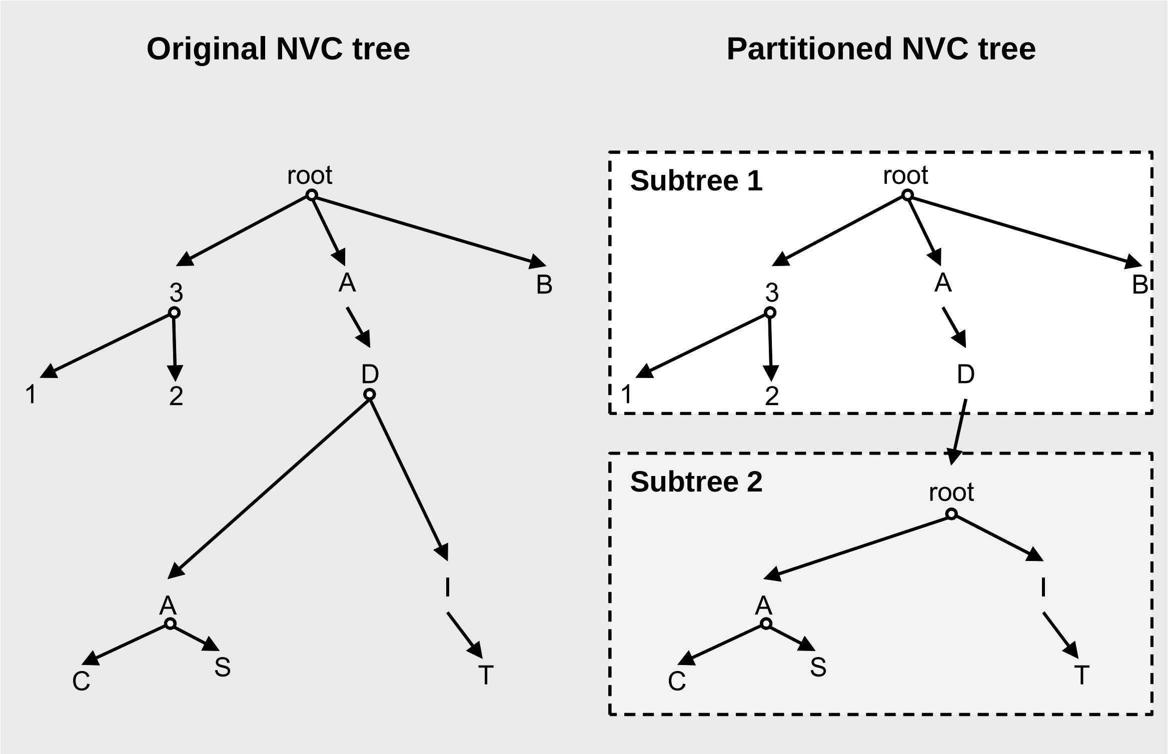
Sibling nodes must always belong to the same subtree. During partitioning of an NVC tree, a reference node is created pointing to the subtree.

When an NVC tree is divided into several subtrees, the root nodes of the subtrees (excluding the root tree) shall not contain name strings because they would overlap with the name strings of the leaf nodes.

The root partition of the NVC tree is referenced by the named object via named object ID, selection criterion, and subtree ID `0`. A subtree is referenced by the parent successor via subtree ID (DataScript location: `nds.names.nvctree` > `TreeRefNode` > `nvcSubTreeId`). Note that the other two components of the primary key (named object ID and selection criterion) are the same as for the root partition of the enclosing NVC tree.

Figure 7-3 shows an original NVC tree and its corresponding partitioned NVC tree. Subtree 2 is the successor tree of subtree 1. D is the reference node pointing from subtree 1 to subtree 2. The root node of subtree 2 has an empty string. Nodes A and I have the same parent node ID and will therefore not be assigned to different subtrees.

Figure 7-3 Partitioning of NVC trees, example



Loading NVC tree BLOBs can be critical for systems with low RAM resources if the BLOBS exceed a certain size. A maximum size restriction for NVC tree BLOBs must therefore be defined. Additionally, different data carriers result in different access times with a reduced number of NVC tree BLOBs that can be accessed in a subsequent selection path. However, as the maximum size of NVC trees is already defined and a maximum coverage definition of NVC trees shall be avoided, only the average size of NVC tree BLOBs needs to be defined here. If the restriction values overlap, the lowest maximum and the highest average restriction value have to be used.

Table 7-3 Restriction values for partitioning and size of NVC trees

Restriction	**DVD**	**HD**	**SD**
Average size of NVC subtree BLOBs	128 KB	64 KB	16 KB
Restriction	**Low**	**Mid**	**High**
Maximum size of NVC subtree BLOBs	0.5 MB	1 MB	2 MB

Sorting Rules Relating to NVC Trees

Table 7-4 shows the sorting rules for NVC trees.

Table 7-4 Sorting rules for NVC trees

Items that must be sorted	Purpose
Strings in an NVC tree node	To enable a fast binary search, strings in NVC tree nodes are sorted according to Unicode code point order.
Name feature references for an NVC tree node	To stop the loading process as soon as all name features have been loaded. For example, the first 20 names are loaded and displayed in a list for this node (for case 2 in Figure 7-2).
Name string references for name features	To stop the loading process for name strings of a name feature as soon as all name strings have been found (for case 2 in Figure 7-2 and default for other names for BMD, route list). **Example** Setting: NVC tree with incomplete names. The prefix is "Karl". The NVC tree points to one named object with this prefix. Solution: The system searches the name strings "Karlsplatz", "Stachus", and "Z-Platz". Once the system finds "Karlsplatz" the name comparison stops because "Stachus" does not meet the expected result ("Karl…").

Disambiguation in NVC Trees

A leaf node may reference more than one name feature (`areNameStringsComplete=FALSE`/ case 2 in Figure 7-2). It is possible to have several named objects with the same name string values. If named objects are ambiguous, they may be replaced by an aggregation named object.

If an aggregation named object is available, at least one NVC tree for disambiguation of the aggregated named objects is required.

For more information, refer to *NDS – Format Specification*, *Disambiguation in Selection Graphs* on page 191.

Linearization of NVC Trees

The NVC tree is linearized by a breadth-first search tree traversal, also known as *level-order tree traversal*. For example, the nodes of the NVC tree in Figure 7-2 (a) are physically stored in the following: order P, A, C, D, B, E, F.

Ideograms in NVC Trees

For ideograms that are used in NVC trees (for example, for China and Korea), the latinization will only be available in the strings of the NVC tree. For named objects, the ideogram is available as a name string. If the latinizations are identical for several named objects, they can be distinguished via the ideogram string being stored as a name string with the named objects. For the differentiation between latinization and ideogram names, the language codes are used. The selection graph definition metadata contains the information whether latinizations are used in NVC trees.

To avoid performance concerns for low-end systems, NVC strings may be optimized with regard to ideograms. The ideogram name strings are stored as an appendix to the NVC string with the latinization.

Note “Beijing” is the intended latinization in the NVC tree and “??” is the ideogram representing it. The NVC string for this example is modified to “Beijing#??” (latinization + separation character + ideogram string).

The separation character is stored per database in the update region table (DataScript location: `UpdateRegionTable` > `nvcStringSeparationCharacter`). This approach enables the application to show the ideogram by directly displaying the stored appendix of the NVC string instead of loading the named object and name string.

Filtering of Names in Different Languages in NVC Trees

High-performance filtering for different languages to be displayed to the user might be required for NVC trees. High-end systems can load all related named objects and their name strings to identify the required name strings based on language codes. Low-end systems, however, filter the information about available languages. For low-end systems it is, therefore, mandatory to store the respective number of matching names that are referenced in subtrees and leaf nodes of the NVC tree.

Rotation of Names

Whenever rotated name strings are used in NVC trees, the rotated and non-rotated part is separated by the rotation separation string, see *NDS – Format Specification*, 20.2.1 *Rotation Metadata* on page 353 and Section 7.3.5 *Rotation Metadata* on page 131. As the rotation separation string is a predefined constant, the application has to replace it with an own rotation separator, for example ", ". This enables the application to

- Provide a consistent view on precompiled and application-generated NVC trees
- Provide rotation separators adapted to specific needs, such as OEM requirements, country specifics, or user needs

Extended Postal Codes

The suffix for extended postal codes is stored in the NVC path. Thus, a complete NVC tree is required for extended postal codes.

The reference at the leaf node contains an attribute list. Therein the attribute `EXTENDED_POSTAL_CODE_POSITION` contains the relative geographic position for an extended postal code.

The presence of this attribute identifies an entry as an extended postal code entry. These entries have named object references to the postal code named object for the base postal code. The administrative area to which an extended postal code belongs can be identified by means of `CONTAINED_IN` relations for the base postal code area.

Big Cities in NVC Trees

To facilitate location input for cities, NVC tree nodes may contain a list of large cities starting with the string composed by the path leading from the root node to the respective tree node.

Example A user enters "B". The application immediately displays the following cities: Berlin, Bremen, Bochum. The user enters a second character: "i". The list of cities displayed in the application now shows Bielefeld, Bitterfeld-Wolfen, Bietigheim-Bissingen (see Figure 7-4 and Figure 7-5.

To optimize the size, the information is stored only in the reference nodes and leaf nodes. Information for inner nodes is calculated online based on the following information:

- The reference node contains a list of flexible attributes with a reference to the named object and a size indicator for the respective cities (`TOP_SELECTION_ENTRY`).
- The leaf node contains a size indicator, which is stored as a flexible attribute for the city represented by the leaf node (`TOP_SELECTION_INDICATOR`).

To reduce space, the database does not store the actual population of the city, but the size indicator. The size indicator provides a logarithmic mapping of the population.

Figure 7-4 Size indicator for big cities, example

	Population	Log10	Size Indicator
Berlin	3431473	6,54	208
Bielefeld	323615	5,51	176
Bochum	378596	5,58	178
Bremen	547360	5,74	183
Bitterfeld-Wolfen	46971	4,67	149

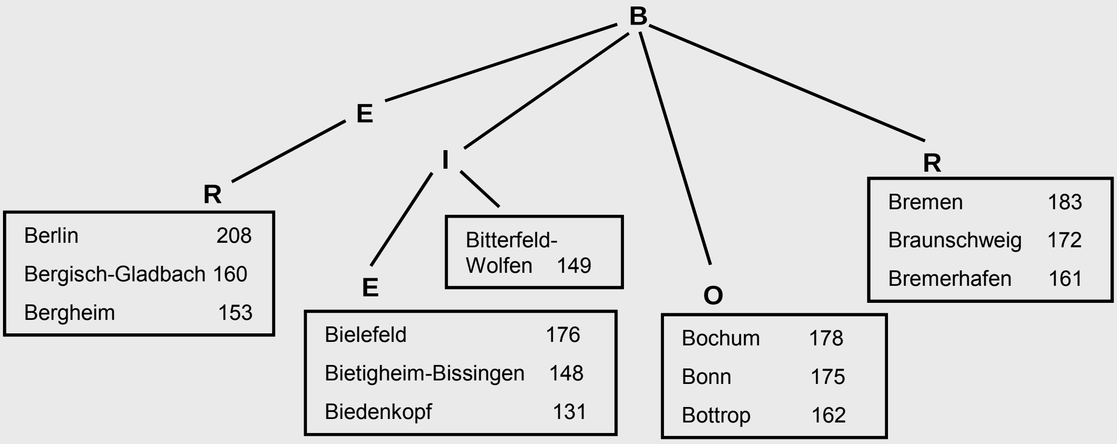
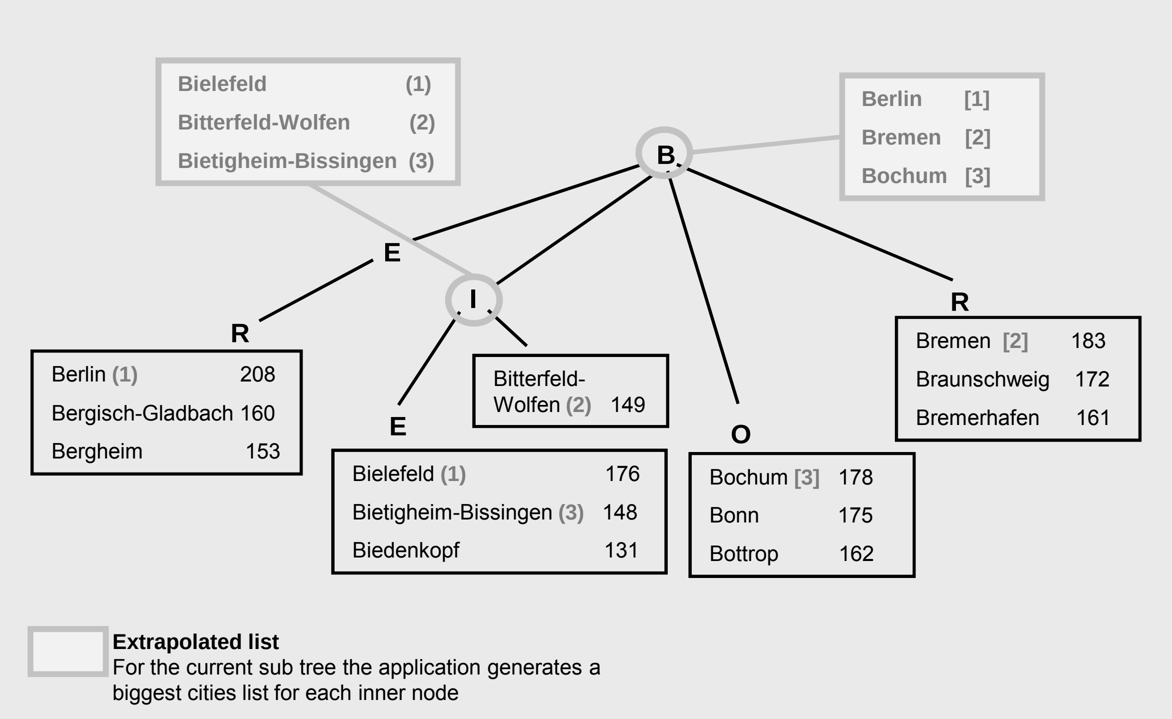


Figure 7-5 Extrapolated lists for big cities, example



7.5.3 Geographical Extent

A geographical extent can be defined for a named object feature. The geographical extent of a feature can be either specified by defining a bounding box or by listing the intersected routing or basic map display tiles.

The number of tiles in the list of intersected tiles shall not exceed 255. If the feature extends over more tiles, a bounding box shall be used for automatic zoom and NVC trees for refined location input. To automatically zoom in a very long road or a ferry line, for example, a bounding box may be assigned to this feature. NVC trees may be created for the crossroads and house numbers of long streets. For this purpose, house number named objects must be available. Alternatively, the road may be refined by other information, such as city district, postal code, etc.

For location input using the geographical extent attribute, no precompiled NVC trees are needed. The geographical extent may be used if the relations of a named object to other named objects can be evaluated at runtime, using the data derived from transitive relation. From these relations, NVC trees can be calculated at runtime.

For more information, refer to *NDS – Format Specification*, *A Named Object Feature Has a Geographical Extent* on page 180.

The following list gives examples for location input via geographical extent:

- After the initial selection of a road named object, the corresponding routing features are retrieved via the geographical extent attribute. The intersecting links and their road named objects are also retrieved. Based on this information, the user gets a list of all roads crossing the initially selected road named object.

  Note The determination of intersecting roads for complex crossing situations, such as highway interchanges, is not possible in NDS, because routing data does not provide the required information. Hence, in these cases, the creation at runtime is not possible and a crossroad NVC tree is required.

- After the initial selection of a region named object (for example a small city), the corresponding routing features are retrieved via the geographical extent attribute. The road named objects for the respective links are also retrieved. They are then filtered by their contained-in relations, which refer (directly or indirectly) to the initially selected region named object. Based on this information, the user gets a list of roads assigned to the selected region named object.

Table 7-5 Restriction values for geographical extent attribute for location input

Restriction	DVD	HD	SD
Maximum number of level 13 BLOBs covered by a named object for using the geographical extent instead of NVC tree	2	3	4

7.5.4 Using Relations for Filtering or Address Retrieval

For filtering named objects or for address retrieval, contained-in relations are used. For more information refer to Section 7.6.3 *Relations* on page 145.

7.6 Handling of Named Objects

For the handling of named objects, some issues need to be considered:

- Handling objects and object IDs (see Section 7.6.1 *Handling Objects and Object IDs* on page 144)
- Mandatory and optional use of attributes and relations (see Section 7.6.2 *Attributes* on page 144)
- Assigning named objects to routing and basic map display features (see Section 7.6.4 *Assigning Named Objects to Routing and Basic Map Display Tiles and Features* on page 146)
- Generation of unique named object IDs (see Section 7.6.1 *Handling Objects and Object IDs* on page 144)

7.6.1 Handling Objects and Object IDs

Semantically identical named objects shall not be duplicated or split within one update region to ensure that the application can, for example:

- Always find all links of a road that is selected in an NVC tree
- Draw names for polylines that are composed out of several lines
- Minimize disambiguation for NVC trees calculated on the fly for route lists

Note House named objects will not be included in NVC trees, because there already is a mandatory road-to-house named object relation.

For information on spatial ordering of named objects, see *NDS – Compiler Interoperability Specification*, 3.1.1 *Generating IDs According to Morton Order* on page 47.

7.6.2 Attributes

The following table shows the attributes for named objects, their use (mandatory or optional), and the attribute type (fixed or flexible).

For more information, refer to *NDS – Format Specification*, 10.2.1 *Attributes of Named Object Features* on page 179.

Table 7-6 Attributes for named object features, overview

Attribute	DataScript	Use	Attribute type
`isDestination`	`nds.name.object > NamedObjectAttributes`	Mandatory	Fixed
`isAggregationFeature`	`nds.name.object > NamedObjectAttributes`	Mandatory	Fixed
`useBoundingRectangle`	`nds.name.object > NamedObjectAttributes`	Mandatory for all non-aggregation named objects	Fixed
`position`	`nds.name.object > noAggregationFeatureAttributes`	Optional	Fixed
`boundingRectangle`	`nds.name.object > noAggregationFeatureAttributes`	Optional	Fixed
`extTileIdList`	`nds.name.object > noAggregationFeatureAttributes`	Optional	Fixed
`accessLocationList`	`nds.name.object > noAggregationFeatureAttributes`	Optional	Fixed

Attribute	DataScript	Use	Attribute type
NAMED_OBJECT_POSITION	nds.common.flexibleattributes>AttributeValue	Optional (if more than one position for a named object available)	Flexible
FLOOR_NUMBER	nds.common.flexibleattributes>AttributeValue	Optional	Flexible
POPULATION	nds.common.flexibleattributes>AttributeValue	Optional	Flexible

7.6.3 Relations

The following relations may be used for referencing named object features to other named objects. The relations are filled and evaluated during compilation.

Note House number named objects are optional and only needed for long roads. They may also be saved as flexible attributes for routing base links or road geometry lines.

For more information, refer to *NDS – Format Specification*, 10.2.2 *Relations between Named Object Features* on page 181.

Contained-in Relations for Address Retrieval

All named objects forming address elements, for example administrative area, road, and house number, shall refer directly or indirectly (via subsequent contained-in relations) to a country named object. Named objects forming address elements below an order 8 admin area, for example, order 9 admin areas or postal areas, shall refer directly or indirectly to the corresponding order 8 admin area.

All other named objects (for example, map display or ferry lines which are in an international area) may not have a direct or indirect relation to any country named object.

House-to-road Relations (ACCESS_AT_ROAD) for Address Retrieval

House numbers (or house number ranges) are usually tied to road named objects. In this case, the relation of house named objects to road named objects is mandatory. In some countries, however, house numbers are tied to region named objects. In this case, a relation between house and region named objects is not necessary. These relations are also required as contained-in relations for address retrieval.

Road-to-house Relations for Location Input

Road-to-house relations shall be realized either by means of a relation of the (optional) house number named object to the corresponding road named object or by means of a flexible attribute attached to a base link or a road geometry line:

- If house number named objects are available, each road named object used for location input and having house numbers or house number ranges, must be related to its house named objects. This is the only access that a location input application can use to retrieve the house numbers for a selected road named object.
- If no house number named objects are available, the flexible attributes for base links and road geometry lines need to be evaluated.

Capital Relations

This relation connects a named city feature with a named region feature for which a capital may be defined, for example a country or a state. This relation is mandatory for administrative area named objects being a capital. The feature class of the referenced named object defines the administrative level of the capital. For example, Berlin references Germany. Germany is a country. Hence, Berlin is a country capital.

7.6.4 Assigning Named Objects to Routing and Basic Map Display Tiles and Features

Named objects are assigned to **features** of the Routing and Basic Map Display building blocks by means of a unique `namedObjectId` (DataScript location: `nds.common.flexibleattributes`).

In Routing or Basic Map Display **tiles**, named objects can be referenced by a named object directly, or by a combination of the base value and relative value for the named object ID. The latter needs less storage space, but requires an additional step in reference resolving.

The base value for the named object reference is stored as a flexible attribute (DataScript location: `nds.common.flexibleattributes` > `AttributeValue` > `NAMED_OBJECT_BASE_REF`). This attribute can only be assigned to tiles, not to individual features.

For tiles containing the attribute, the references to named objects in the tile are relative. They are calculated as follows:

`namedObjectId` = `NAMED_OBJECT_BASE_REF` + `NAMED_OBJECT_RELATIVE_REF`

The flexible attribute `NAMED_OBJECT_RELATIVE_REF` stores a value which is relative to `NAMED_OBJECT_BASE_REF`. `NAMED_OBJECT_RELATIVE_REF` must be used in combination with `NAMED_OBJECT_BASE_REF`.

The relative reference is a variable integer, meaning that the range of named object IDs in a tile with `NAMED_OBJECT_BASE_REF` can be `[namedObjectBaseRef+MIN(VarInt16), namedObjectBaseRef+MAX(VarInt16)]`.

Tiles without the `NAMED_OBJECT_BASE_REF` attribute have absolute references to named objects in the tile, that is, the references are modeled by means of named object IDs as indicated in the `AttributeValue` > `featureToNamedObjectReference` attribute.

7.6.5 Region Named Object Reference

The region named object reference (`REGION_NAMED_OBJECT_REFERENCE`) is mandatory for routing and basic map display tiles. The reference is used to find out if a set of features or all features of a tile are contained in a certain region, for example, an administrative area.

This information is necessary, if the configuration in global metadata requires specific treatment or special attributes for an area, for example:

- Road numbers require icons
- Vignettes are required

7.7 Name Strings

A named object shall always have a default name. If several alternative names are available for a named object, at least one preferred name must be related to the named object.

Note Named objects without an assigned name string are allowed. They are placeholders for the address reconstruction. For example, a road without a name is needed for address retrieval.

Name strings shall be self-contained. This means that complete UTF-8 characters shall be stored instead of some bytes in one string and the continuation bytes in another string.

Note Name strings may have no defined language due to insufficient or erroneous data. The application may decide whether to display or ignore these name strings.

Unexpected characters can occur in some countries, for example Turkish characters in German names. It is recommended to replace these characters by their representation in the local language. There can be cases, for example English names for POIs in Japan, in which replacing characters is not possible.

For more information, refer to *NDS – Format Specification*, 10.3 *Name String Features* on page 185.

Identical Names for One Named Object

There are several use cases for displaying names:

- Always display the official names and additionally the names in the user-selected language
- Always display the available official and alternate names

Note	In order to reduce ambiguity, the application has to filter out identical names.

Corner Cases for Name Strings

Names are usually displayed on the map or in the NVC in the user-selected language. To save space, exonyms, which are available in the source data, but are identical to the official name of the named object are not stored as named objects strings. Neither are they counted as remaining matches of NVC nodes. For example, the exonym "Berlin" is not stored in English, because the official name is identical in German: "Berlin".

Exonym names are not removed, if they are identical in different languages, but not identical to the official name of the named object . If the application displays all names in all languages, the exonym would be available in the list several times. To avoid confusion, the application needs to eliminate these duplicates. This elimination applies to the list of name strings presented to the user and to the number of remaining matches. The official name for Regensburg in German is, for example, „Regensburg“. The exonyms for „Regensburg“ in Italian and Spanish have the name string "Ratisbona". In this case, all name strings are stored: Regensburg(DEU_deu), Ratisbona(ESP_spa), Ratisbona(ITA_ita).

Assigning Language Codes

To reduce the database size, the compiler should assign language codes according to the frequency of use. That means the compiler should assign a low code number (smaller than 128) to frequently used languages such as British English (`UK_eng_Latn`).

8 Basic Map Display Building Block

The Basic Map Display building block contains map features which are necessary for rendering two-dimensional maps together with their geometrical data.

8.1 Validity of Feature Geometry

In order to ensure that the geometry data of basic map display features is displayed correctly in a navigation system, the topics described in the following sections shall be considered during compilation.

8.1.1 Point Features

During encoding, point features are assigned to a tile according to the *south-west rule*.

Navigation Data Standard defines a tile as follows: If (x1, y1) is the south-west corner of a tile and (x2, y2) is its north-east corner, then all points (x, y) with x1 ≤ x < x2 and y1 ≤ y < y2 are uniquely assigned to the tile. The western and southern borders belong to the tile whereas the eastern and northern borders belong to the neighboring tiles. Therefore, point features located on the southern or western border are assigned to the tile, and point features located on the northern or eastern border are assigned to the neighboring tile.

8.1.2 Line Features

A line must have at least two distinct points.

Multi-line geometries are not supported by Navigation Data Standard. If features with a multi-line geometry (so-called *multi-line features*) occur in the source data, they are represented as separate line features in the NDS database. To indicate that the parts of a multi-line geometry belong together, they are referenced by an object feature with the same name.

The `DRAWING_ORDER` attribute (see Section 8.3 *Drawing Order* on page 154) defines the relative height of features to each other. For self-intersecting lines, the following dependencies with the drawing order need to be considered:

- If the height at the intersection is relevant for a line feature with self-intersection (for example a serpentine road with a bridge), the feature has to be divided into two parts at an existing or newly introduced shape point. A different drawing order attribute must be assigned to the two parts.
- If no height information exists or if it is not relevant (for example, for a sidewalk in shape of an eight), the line feature may be modeled as a single-line geometry with self-intersection.

Note This applies to line features only. Self-intersection is not allowed for polygons.

8.1.3 Area Features

The physical data format of the Navigation Data Standard supports the following three types of polygon types:

- Simple polygons
- Triangle strips
- Triangle fans

For more information on the three polygon types, refer to *NDS – Format Specification*, 11.5 *Area Features* on page 219.

Which polygon type is used depends on the navigation system:

- For **high-end systems** (systems with 3D graphic boards), the use of triangle strips and triangle fans instead of simple polygons is recommended.

  As triangulation for these types of simple polygons is carried out during compilation, the 3D graphic boards used in high-end systems can already receive the pre-processed area data. Triangulation for simple polygons, however, is carried out by the software in the navigation system and takes more time.
- For **low-end systems** (systems without 3D graphics boards), the use of simple polygons is recommended because these systems can process simple polygons faster than they can process triangle strips or fans.

Requirements for Polygon Types

The Open GL (Open Graphics Library) standard defines a cross-language cross-platform interface for writing applications that produce 2D and 3D computer graphics. The NDS standard references the logical geometry model used in the Open GL standard.

As OpenGL does not support self-intersection, the polygon types must comply with the following requirements:

- **Simple polygon**: Defined by a sequence of vertices (v0, v1, v2, v3, v4, v5, ..., vn) and edges e0 = (v0, v1), e1 = (v1, v2), …, en = (vn, v0). Two edges of a simple polygon do not intersect, except for the common vertex of two successive edges.
- **General triangle strip**: Defined by a sequence of triangles. Any two successive triangles of the sequence share a common edge.

- **OpenGL strip**: Defined by a sequence of vertices (v0, v1, v2, v3, v4, v5, ...), where any three successive vertices define a triangle with positive orientation. To maintain positive orientation, the first two vertices are reversed in every other triangle: (v0, v1, v2), (v2, v1, v3), (v2, v3, v4), (v4, v3, v5), …
- **OpenGL fan**: Defined by a sequence of vertices (v0, v1, v2, v3, v4, v5, ...). The first vertex together with any two successive vertices define a triangle with positive orientation, that is, (v0, v1, v2), (v0, v2, v3), (v0, v3, v4), (v0, v4, v5), ...

From a general triangle strip, a vertex sequence can be constructed by taking the three vertices of the first triangle and then adding the unique vertex of each following triangle which is not contained in the preceding triangle.

This sequence, however, will not always be an OpenGL strip. The vertex sequence of a general strip is an OpenGL strip if any t[i+1] is obtained from t[i] by replacing the oldest vertex (meaning the one with minimum index) in t[i].

Any strip can be turned into an OpenGL strip by inserting duplicate vertices, so-called *swap vertices*. Figure 8-1 shows an example of a triangle strip with swap vertices.

Example The triangle strip is defined as follows:

t0 = (v0, v1, v2)

t1 = (v2, v1, v3)

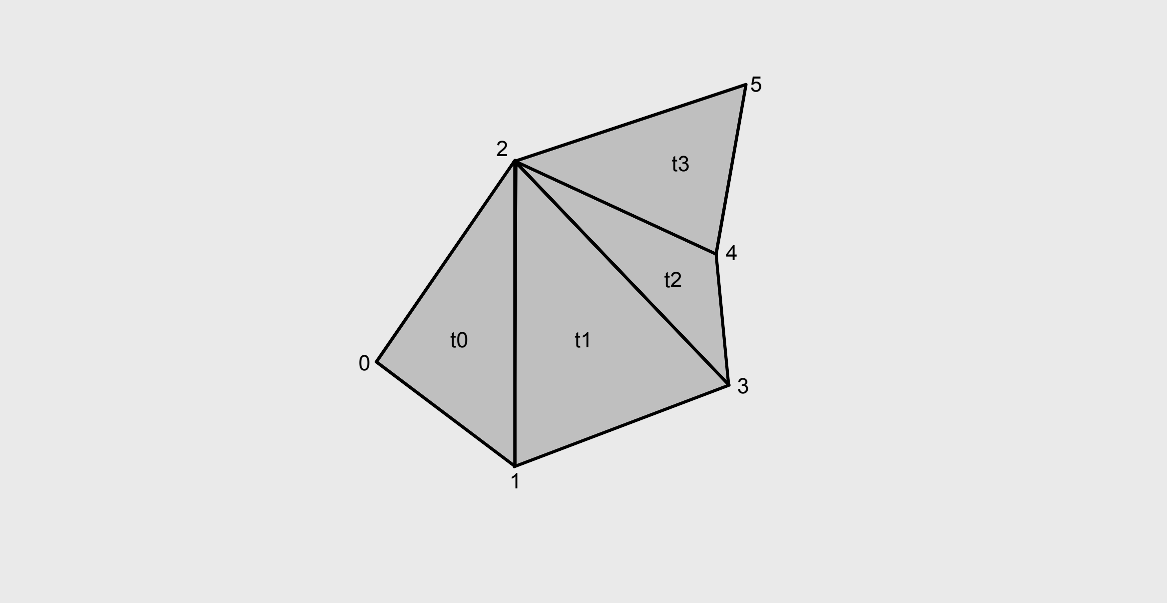
t2 = (v2, v3, v4)

t3 = (v2, v4, v5)

The vertex sequence resulting from this strip is v0, v1, v2, v3, v4, v5. If this sequence is interpreted as an OpenGL strip, the result would be t3 = (v4, v3, v5), which is not correct.

Therefore, this strip must be converted to an OpenGL strip.

Figure 8-1 Triangle strip with swap vertices, example



The new vertex (v5) of the last triangle is appended to the sequence, and, in addition, a copy of the oldest vertex of the previous triangle (v2) is inserted between the second and third vertices of the previous triangle. In our example, the resulting sequence is as follows: 0 1 2 3 2 4 5.

Any swap vertex creates a degenerate triangle (in our example: (v2, v3, v2)). The OpenGL engine silently suppresses this degenerate triangle. The number of vertices in the resulting strip is n+s where n is the original number of vertices and s is the number of swaps. Thus, the worst case is a total strip length of 2n, which is still better than the 3n vertices required for n individual triangles.

8.2 Use of Levels

The Basic Map Display Building Block may use up to 16 levels. Data is stored partly redundant on different levels. To reduce the density, the compiler generalizes the data for upper levels.

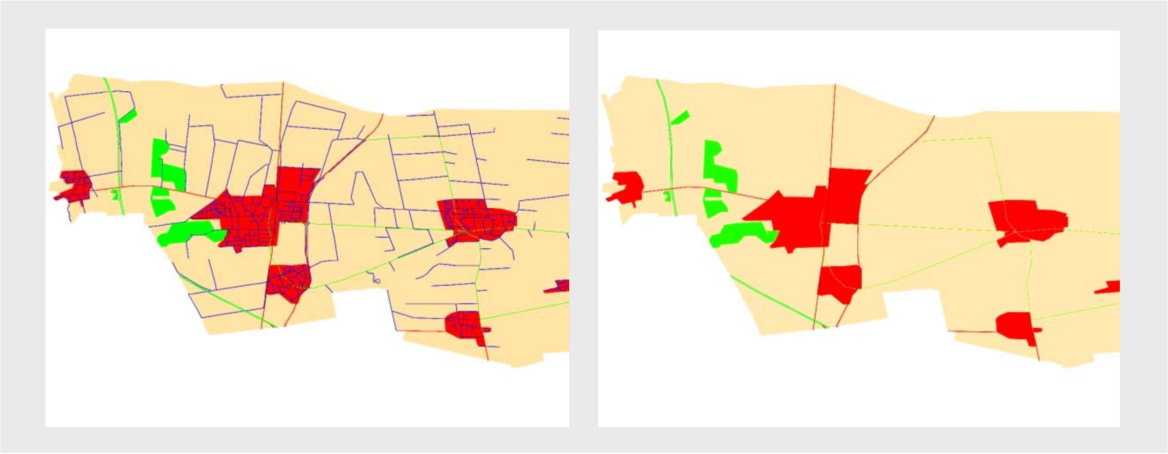
There are four generalization methods: Filtering, simplification, adjustment, and aggregation. Generalization recommendations are subject to change according to prototyping experiences. Therefore, recommendations for the compiler cannot be given at this point of time.

Filtering

Filtering is a method of removing features that are unimportant for a given level of detail. Areas of the feature class `AREA_BUS_STATION`, for example, can be removed from level 10 tiles while features of the feature class `AREA_AIRPORT` should not be removed. Another example for filtered features are roads with the lowest road class or area features with a width or length smaller than a defined threshold value.

Figure 8-2 illustrates the filtering of features.

Figure 8-2 Filtering of less important features

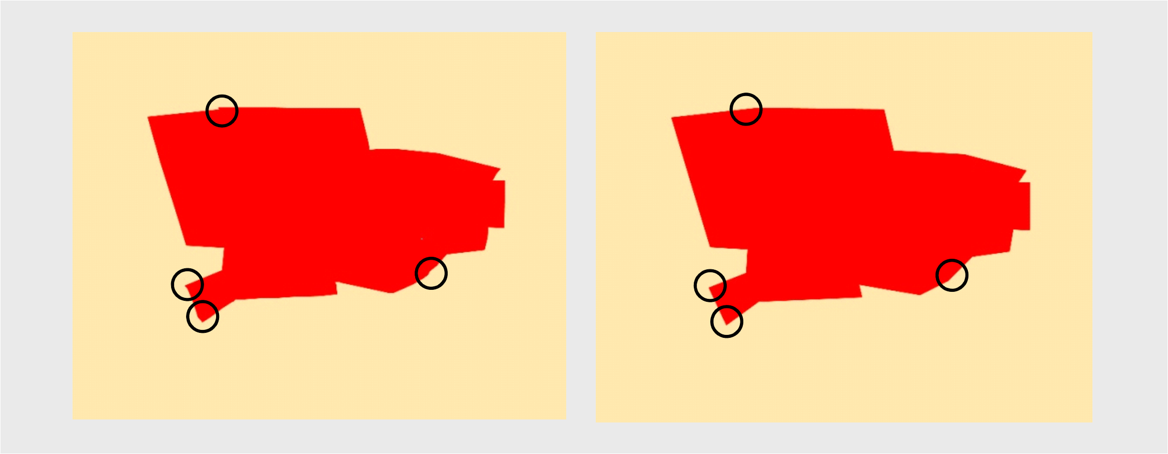


Simplification

Simplification is a method of reducing the number of shape points for a given level of detail. If two shape points are too close to each other on level 10, for example, then one of them can be removed.

Figure 8-3 illustrates the simplification of features.

Figure 8-3 Simplification of features



Adjustment

Adjustment is a method of coordinate modification according to the precision specified for a given level of detail. In this way, elements can be displaced. Example: A road, a railroad, and a river are located close to each other. If the map is zoomed out, the displayed width of each of these elements is larger than the corresponding scaled size. Therefore, river and railroad can be moved to prevent them from overlapping.

For general information on coordinate precision, refer to 3.3 *Reducing Coordinate Precision* on page 49.

Aggregation

Aggregation is a method of replacing a group of features of the same class that logically belong together with a single feature of that class. Example: If level 13 contains three areas of the feature class `AREA_AIRPORT` that are all adjacent to each other, they can be replaced with a single area of the feature class `AREA_AIRPORT` on level 10.

8.3 Drawing Order

The general rules for determining and storing information on the drawing order are described in *NDS – Format Specification*, 11.9 *Drawing Order* on page 223.

Handling the `DRAWING_ORDER` attribute

The drawing order may be defined by the value of the flexible `DRAWING_ORDER` attribute, which may be assigned to each basic map display feature and to road geometry lines which belong to the Routing building block. The value may be defined by the NDS database suppliers.

The compiler shall adhere to the following drawing order when saving the data to the data carrier:

- Features with lower drawing order are rendered first.
- Features with higher drawing order are rendered on top of features with lower drawing order.

Example A park and a building which is located in the park are both represented by rectangles. To prevent that the building is covered by the park, the park feature shall have a lower drawing order than the building. Thus, the park is rendered first and the building is rendered on top of the park.

- Features with the same drawing order can be rearranged by the application and, for example, be grouped by the drawing style which is a useful optimization for 3D graphics engines.
- To avoid visual artifacts on tile borders, it is recommended to first render all features with drawing order `0` from all visible tiles and then all features with drawing order `1` from all visible tiles, and so on.

The attribute is only required explicitly in case of changes. Otherwise it can be determined implicitly, and writing it would cost memory space. To save space, the features in a tile are ordered with a monotonically increasing drawing order attribute. A feature without drawing order attribute is either assigned the same drawing order attribute as its predecessor, or it has drawing order attribute set to `0` if its local index is `0` and it has no explicit attribute.

Table 8-1 Drawing order and local index, example

Local index	Drawing order
0	0

Local index	Drawing order
1	0
2	0
3	4
4	4
5	7
6	7

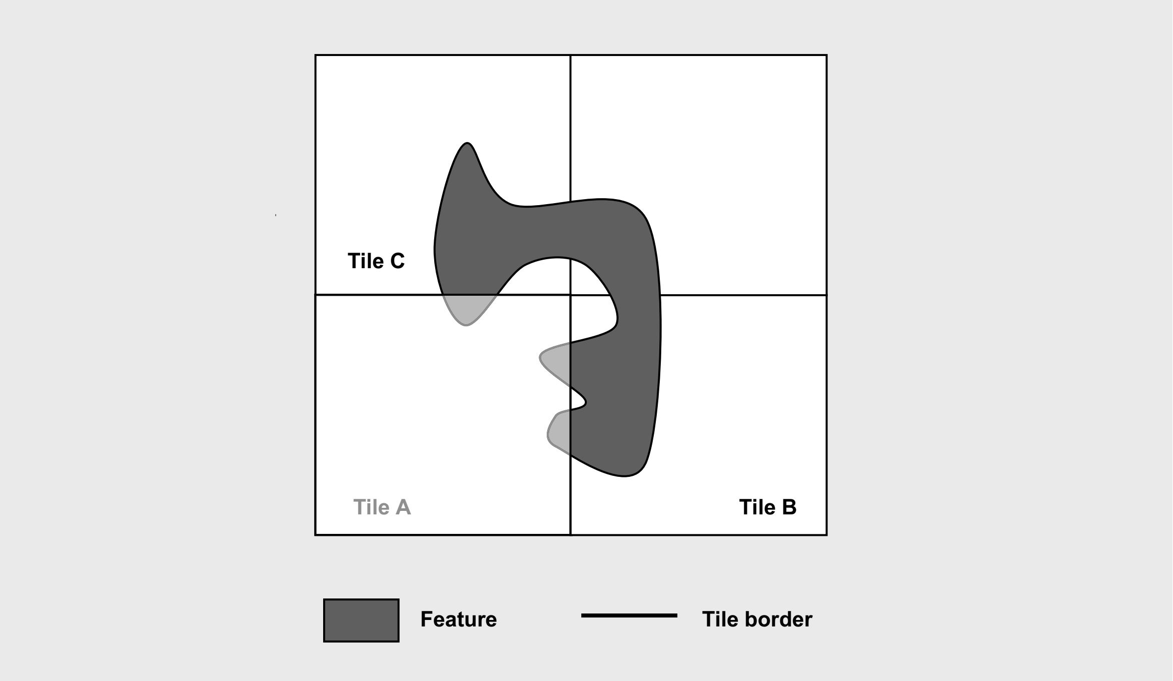
8.4 Relation between Source Features and Compiled Features

Due to clipping at tile borders, it is possible that a source feature is split into several features in the same tile. The parts belonging to the same source feature can be identified implicitly by referencing the same named object.

Area features that belong to the same source feature, which has been split due to clipping at tile borders, can also be identified by their pseudo edges (`AreaCoordXYDiff.isPseudo` flag for polygon edges). The application can execute a spatial search for neighboring area features with pseudo edges and thus identify area features that belong together. This can be useful, for example, for drawing building footprints across tile borders.

Figure 8-4 illustrates an area feature split into several features in the same tile.

Figure 8-4 Area feature split into several features in the same tile



For more information about clipping, refer to the *NDS – Format Specification*.

8.5 Using Z Level Attributes for Map Rendering

The Z level concept of Navigation Data Standard provides more flexibility in rendering ramps, tunnels (underpasses), and bridges. Applications using databases without Z level information can only use the drawing order information to ensure that more important roads are rendered above less important roads, and to avoid that important roads are interrupted.

This section explains the implications of the Z level concept for database compilation. For more information on the data structures used for Z levels, refer to *NDS – Format Specification*, 11.10 *Using Z Levels for Map Rendering* on page 224.

Side Note: Avoiding Artifacts in 2D Map Rendering

A rectangular bar is used to render lines with a width of more than 1 pixel. This will, however, result in artifacts for two bars with one common point as shown in Figure 8-5. To avoid these artifacts, so-called end caps are used for 2D graphics. In Figure 8-6, rounded end caps have been applied to the left and bottom side of the lines. In the corner, the rounded end caps overlap in a circular sector.

Figure 8-5 Artifacts in 2D map rendering

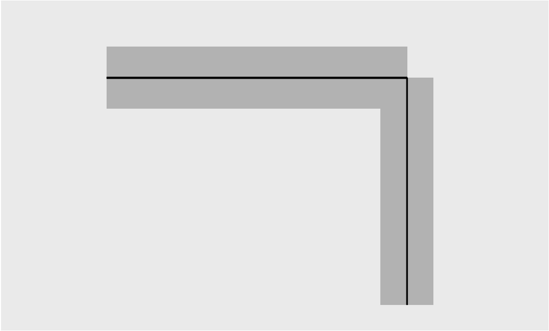
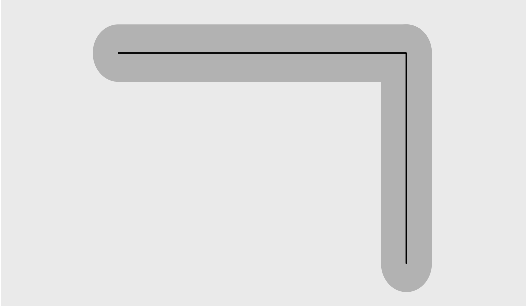


Figure 8-6 Using end caps in 2D map rendering to avoid artifacts



8.5.1 Defining the Overlap Section

For current map displays, roads are rendered with an inline and an outline section in two different colors. Figure 8-7 shows the outline section in black. The outline sections are rendered first. The inline section is illustrated in gray. It is rendered on top of the outline section.

Figure 8-7 Outline section for rendering a road

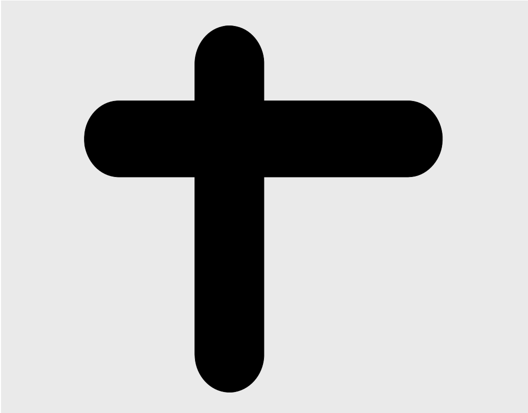


Figure 8-8 Inline section for rendering a road

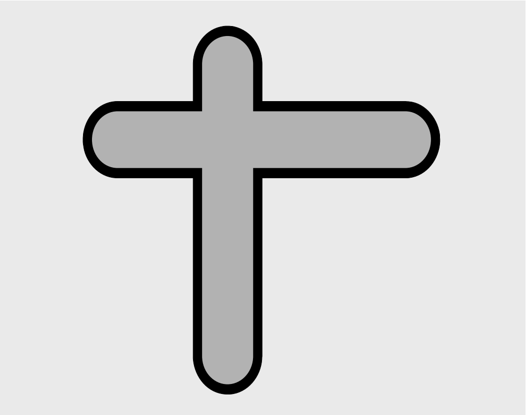


Figure 8-9 and Figure 8-10 show how this is realized for a road with a ramp that branches off the main road and then leads across the main road. The outline section of the ramp is rendered first, then the inline section.

Figure 8-9 Outline section for rendering a road with a ramp (with artifact)

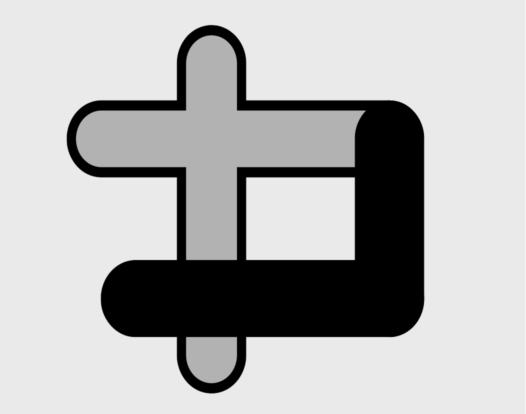
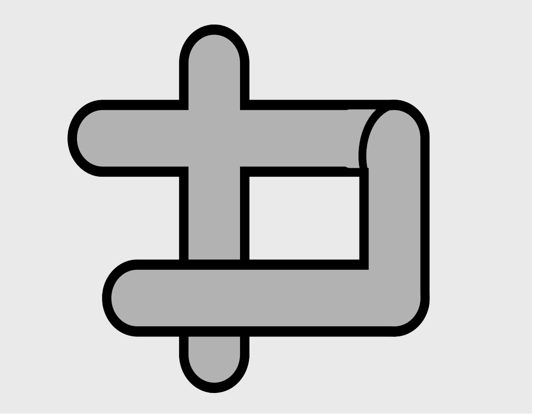


Figure 8-10 Inline section for rendering a road with a ramp (with artifact)

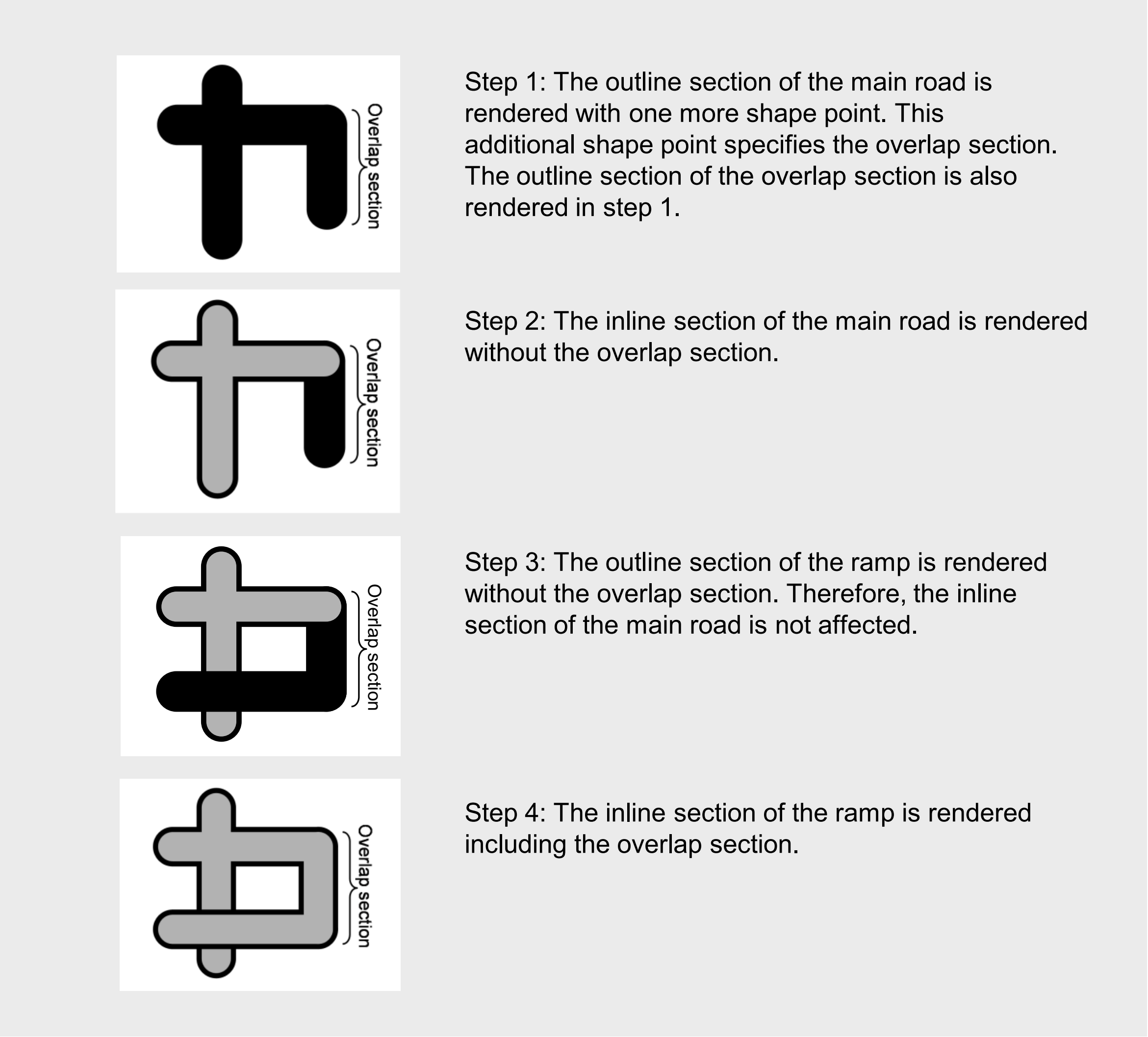


The figure also shows that the result is okay for the part of the ramp that leads across the main road. For the merging section, however, an unwanted artifact occurs.

To avoid the artifact, an overlap section needs to be introduced. The overlap section is defined by an additional shape point in the merging section of the main road and the ramp. For more information on additional shape points, see 8.5 *Using Z Level Attributes for Map Rendering* on page 156.

Figure 8-11 shows the steps for map rendering.

Figure 8-11 Rendering a road without artifacts



Step 1: The outline section of the main road is rendered with one more shape point. This additional shape point specifies the overlap section. The outline section of the overlap section is also rendered in step 1.

Step 2: The inline section of the main road is rendered without the overlap section.

Step 3: The outline section of the ramp is rendered without the overlap section. Therefore, the inline section of the main road is not affected.

Step 4: The inline section of the ramp is rendered including the overlap section.

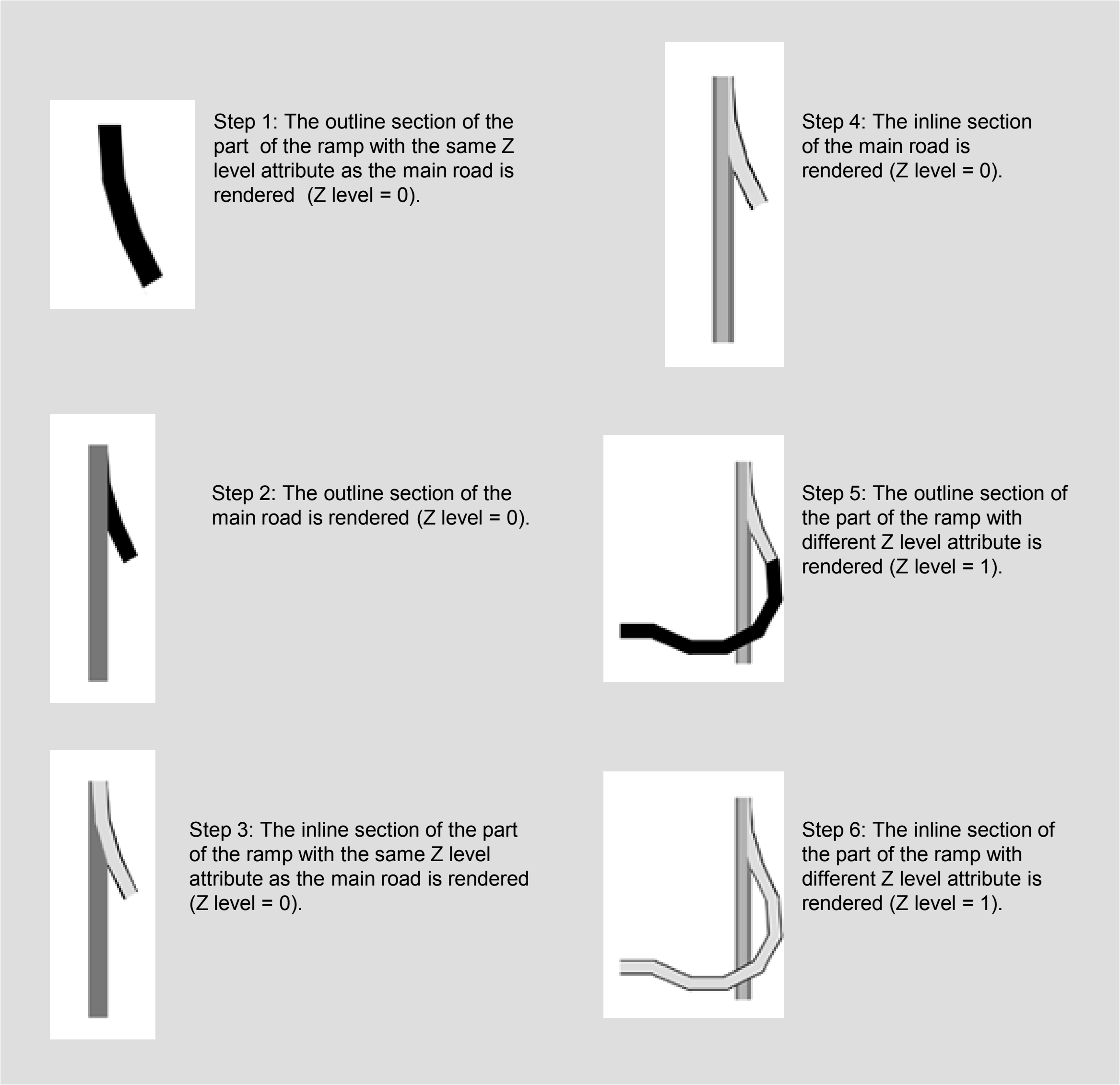
8.5.2 Using a Single Link for Z Level change

Z level changes shall only occur within links. Z level changes at the end of links are not allowed. Links that meet in an intersection shall have the same Z level at the intersection (see Section 8.5.3 *Using Shape Points for Map Rendering with Z Levels* on page 161). The link starts with Z level A and ends with Z level B. It is also possible to have a link starting with Z level A, then Z level B, and ending with Z level A again. This is used, for example, for roads crossing a river.

To render a main road with a ramp correctly, it is necessary to have a Z level change either for the main road or for the ramp. The position of the Z level change on the road link is important for good map rendering results.

Figure 8-12 illustrates the steps for map rendering, and also shows a reasonable position for the Z level change.

Figure 8-12 Splitting road for map rendering of roads with Z level attribute


Step 1: The outline section of the part of the ramp with the same Z level attribute as the main road is rendered (Z level = 0).
Step 4: The inline section of the main road is rendered (Z level = 0).
Step 2: The outline section of the main road is rendered (Z level = 0).
Step 5: The outline section of the part of the ramp with different Z level attribute is rendered (Z level = 1).
Step 3: The inline section of the part of the ramp with the same Z level attribute as the main road is rendered (Z level = 0).
Step 6: The inline section of the part of the ramp with different Z level attribute is rendered (Z level = 1).

8.5.3 Using Shape Points for Map Rendering with Z Levels

To capture the real topology for ramps, tunnels (underpasses), and bridges as good as possible, additional shape points between the start shape point and the end shape point need to be inserted.

Some of these shape points must be inserted during compilation. The following figures show examples of how shape points can be used for map rendering with Z levels.

In Figure 8-13, shape point C is an additional shape point. It is needed to create the overlap section between the shape points A and D.

Figure 8-13 Adding a shape point to create an overlap section

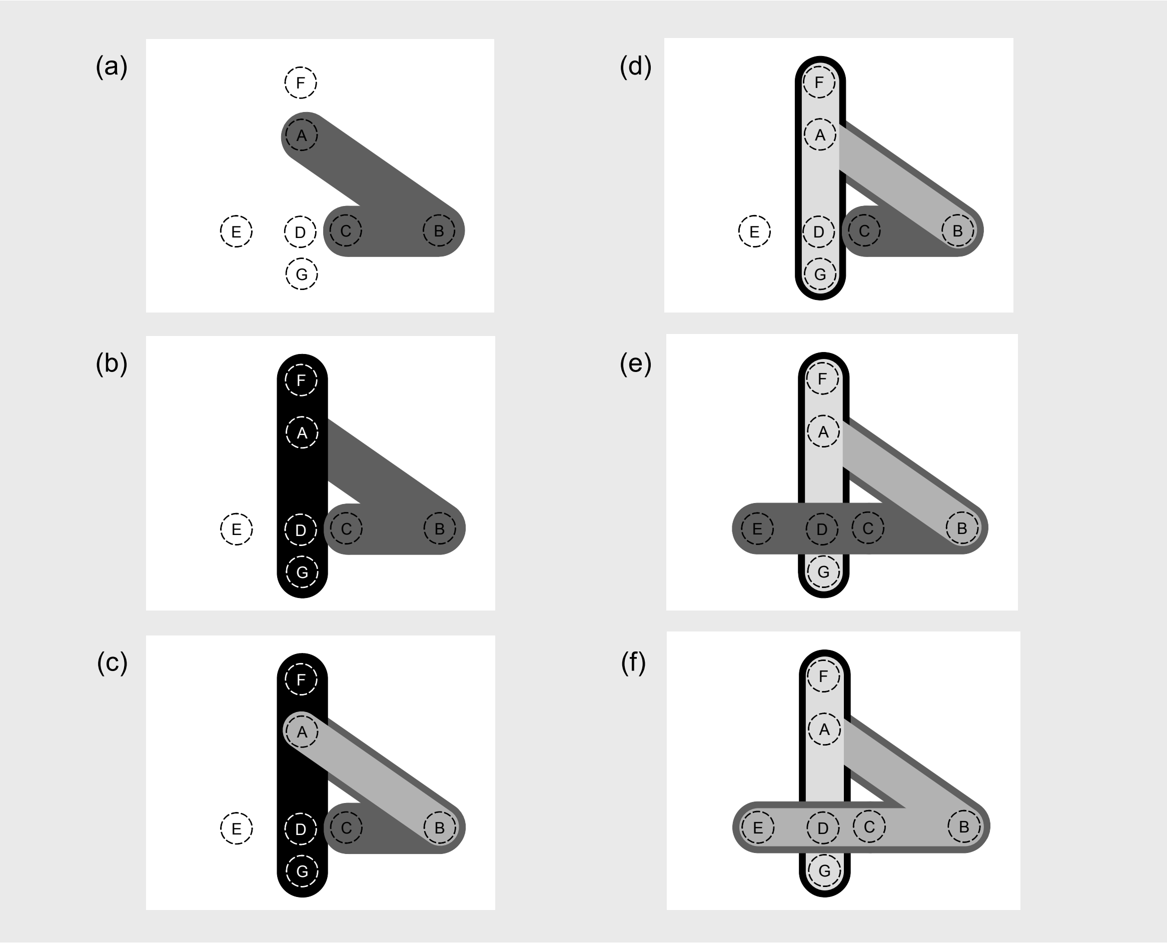
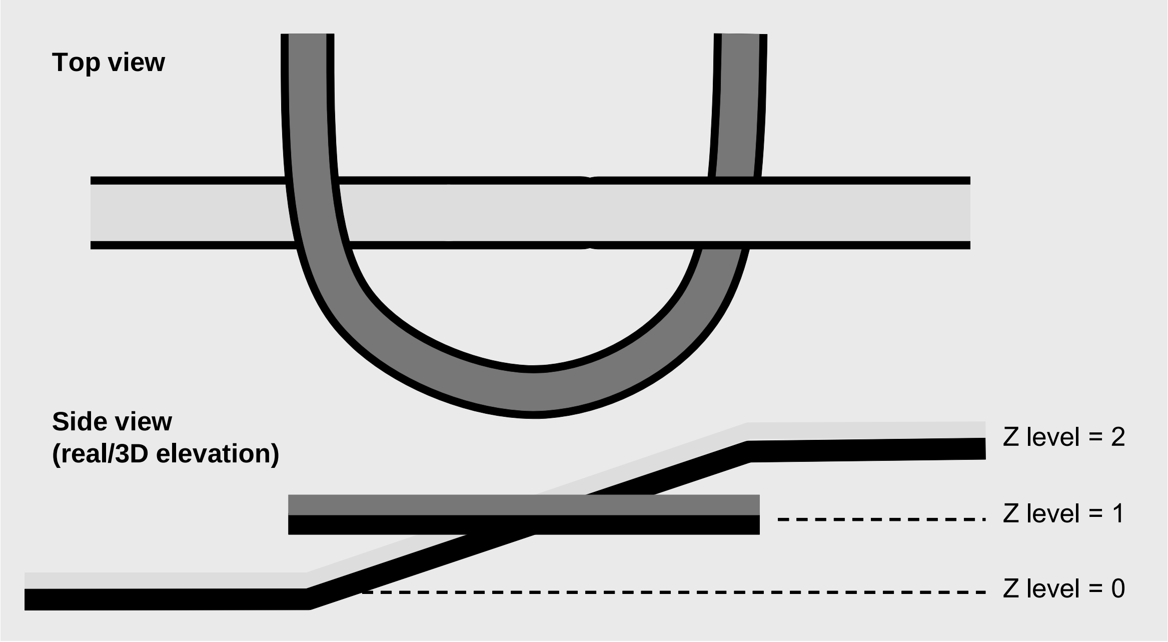


Figure 8-14 shows a Z level road in 2D (top view) and 3D (side view). Figure 8-15 shows how this display can be achieved.

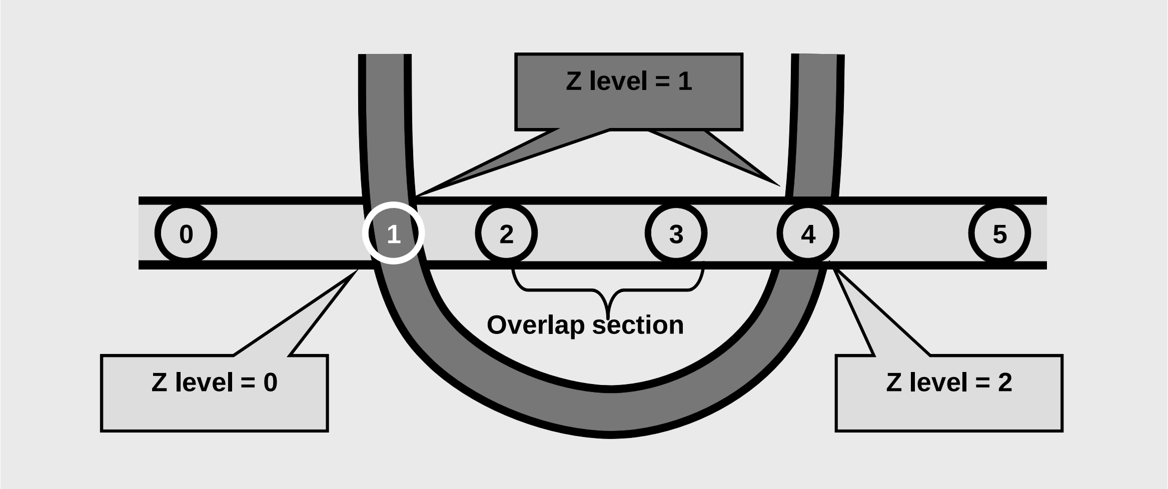
Figure 8-14 Top and side view of two under- and over-passing roads



The dark gray road has Z level 1 for the complete road. Within this road there is a second Z level 1 for the part of (1) to (4). This is necessary to enable the application to find a pair of shape points with identical longitudes and latitudes at different Z levels.

On the light gray road, Z level 0 ends at shape point 1 (for having a match shape point with dark gray road) and Z level 2 starts with shape point 4. Between the shape points 1 and 4 are two redundant shape points building the overlap section. The overlap section avoids artifacts in 2D map rendering.

Figure 8-15 Top view of two under- and over-passing roads with necessary shape points



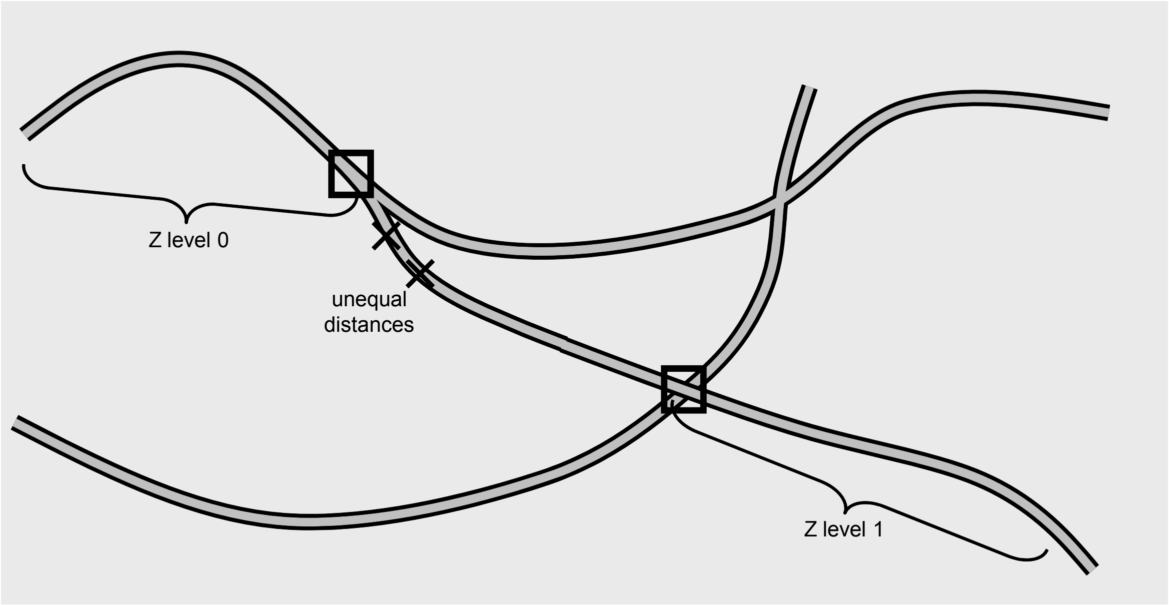
The next important issue shown in Figure 8-15 is the coding of Z level 1 of the dark gray road. The validity range starts at the beginning of the road and ends at a shape point that has to be located exactly at the position (1) where Z level 0 of the light gray road ends. The same situation occurs at point (4).

This is important, because this kind of coding gives the application the possibility to match a shape point of the light gray and a shape point of the dark gray road. For more information, refer to Section 8.5.4 *Common Shape Points for Underpasses* on page 166.

Correcting the Distance between Shape Points

It is recommended to have about the same distance between the shape points near Z level changes. Figure 8-16 shows cumulated shape points behind the start shape point, followed by a straight road section without shape points.

Figure 8-16 Cumulation of shape points



The compiler has two options to adjust the distance between the shape points:

- Adding additional (redundant) shape points, see Figure 8-17

  This option is preferred if the overall distance between the start shape point and the end shape point of the ramp is too short for rendering without artefacts.
- Moving the validity range of the Z levels. In this case the ramp will start at another shape point (compare Figure 8-18 and Figure 8-19).

  This option is suitable for road geometries with a strong cumulation of shape points. By moving the range of the Z level section within the road, only two additional shape points need to be inserted.

Figure 8-17 Insertion of redundant shape points to achieve about equal distances

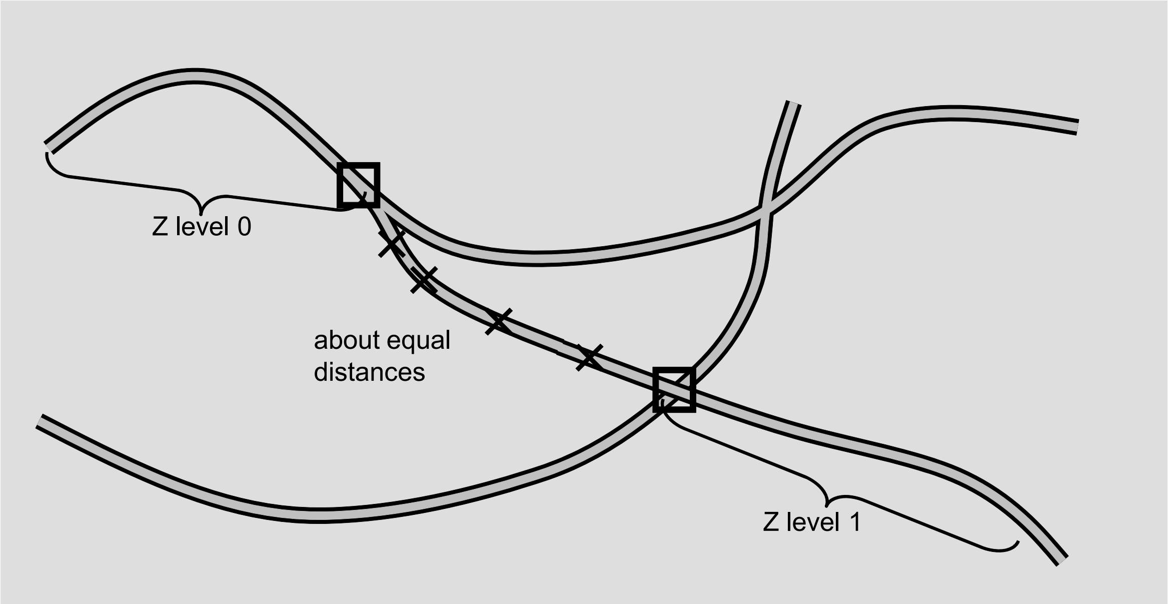


Figure 8-18 Strong cumulation of shape points

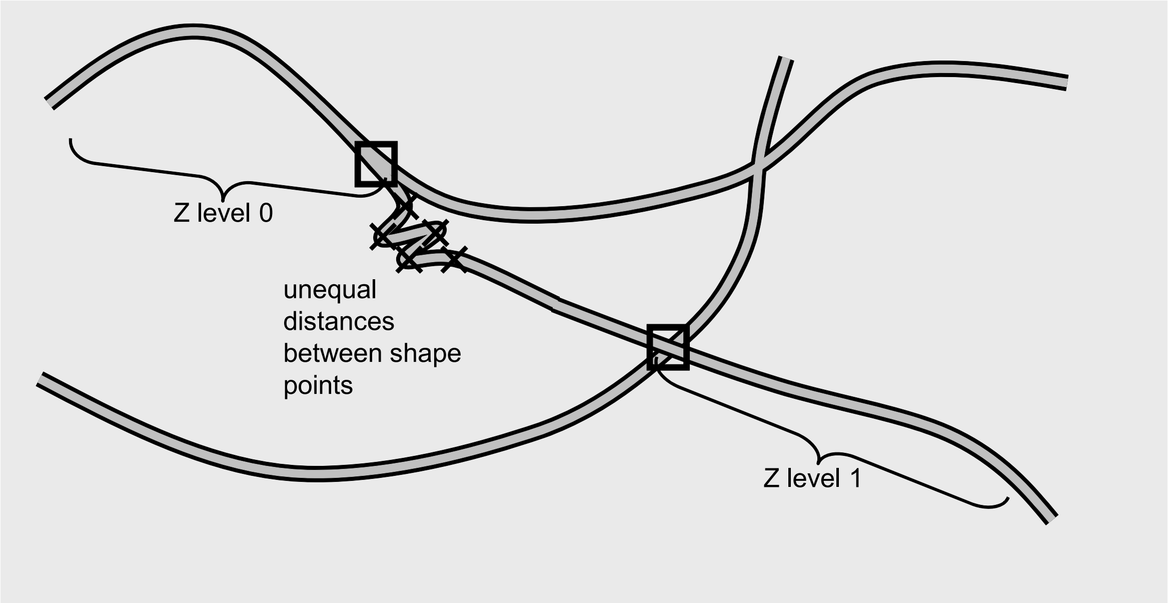
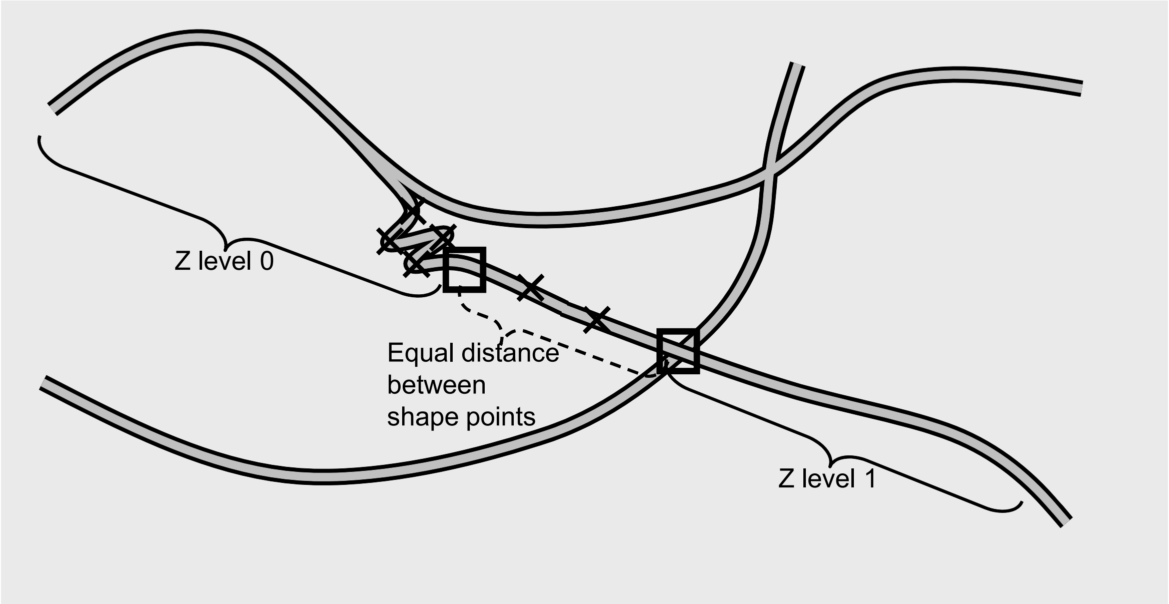


Figure 8-19 Strong cumulation of shape points is not longer part of ramp section

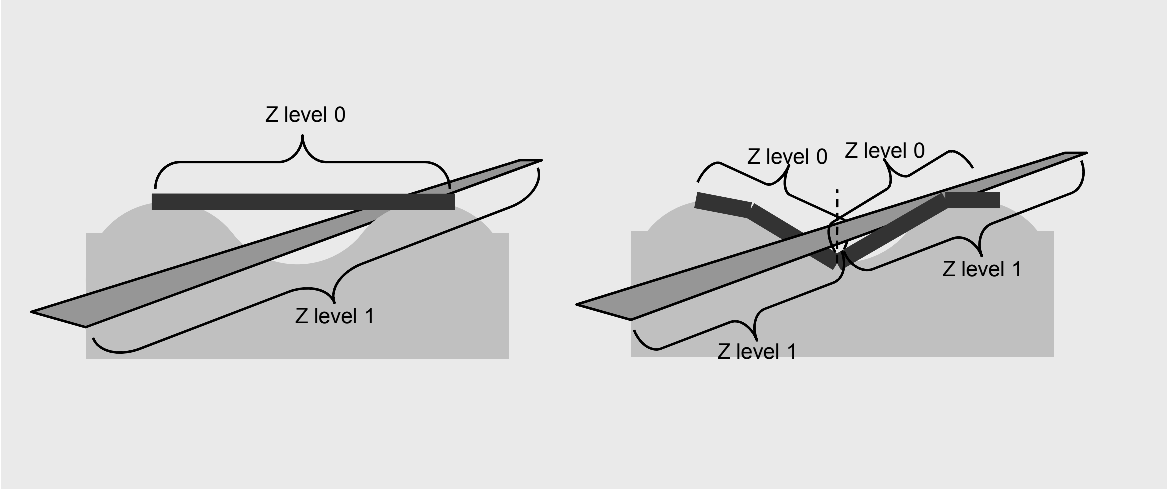


8.5.4 Common Shape Points for Underpasses

Figure 8-20 shows a special case: The dark gray road has Z level 0. The light gray road has Z level 1. For specific terrains (a canyon, for example), the crossing points may be reduced during compilation, for example, by a so-called node killer. In this case, the light gray road would be wrongly rendered below the dark gray road as depicted in Figure 8-20 (a) .

Therefore, an additional shape point with the same x-/y-coordinates must be inserted in both roads. Furthermore, in both roads the shape point must be part of a start/end of a Z level validity range (see Figure 8-20 (b)). This ensures that the application can find the underpass point.

Figure 8-20 Additional shape point for special cases



8.6 Labeling Hints

Labeling hints for features of the basic map display are provided for labeling area features with horizontal text.

Note	Labeling hints for line features are planned for later versions of NDS.

For more information, refer to *NDS – Format Specification*, 11.13 *Labeling Hints* on page 234.

Prioritization of Labels on the Map

The flexible attribute `importance` is assigned to labeling points to enable the prioritization of labels within the set of map features to be displayed on the screen.

For area features, the value for `importance` is defined by the area of the representing polygon:

importance = I_{max} * min { 1.0, ln(A) / ln(A_{max}) } ;

I_{max} denotes the maximum value of importance, which is given as 0xFF due to the bit size of importance. Amax defines the upper limit of the interval of areas, which should be mapped to the value range [0, I_{max}]. A and A_{max} denote areas in square meters. Differences in areas greater than A_{max} are not distinguished by importance. NDS recommends the following value: A_{max} ~ 1011 m^2.

8.6.1 Horizontal Text for Area Features

A hint for horizontal labeling texts for area features consists of a list of labeling positions, so called labeling points. A labeling point is defined by its position (X,Y coordinates within the BMD tile), the highest level on which the point is visible, meaning the least detailed basic map display level containing the respective labeling point, and a circle radius defining an area centered at the point's position.

The radius of a labeling point is defined by the minimum distance to the polygon's outline. Thus, the radii induce a prioritization of the labeling points within one polygon. The best labeling point of a polygon is defined by the point with the biggest incircle. Radius values exceeding the data format are truncated.

The incircle of a labeling point may overlap with adjacent BMD tiles and with incircles of different labeling points.

The labeling points of one horizontal label are sorted by their highest visibility levels first, and by their radii second.

9 Traffic Information Building Block

In order to link traffic-related information from TMC messages to NDS features, it is required to set up references between TMC-specific location information and the respective NDS features in Routing and Basic Map Display building blocks.

The following sections describe how flexible attributes and global TMC lookup tables are used to reference NDS features to TMC location codes.

9.1 Local TMC Attribute Lists

For each tile of the Routing or Basic Map Display building block containing at least one pair of coordinates of a TMC point location, an entry in a local TMC attribute list must be made in the Traffic Information building block. The attributes provide references to features in the Routing and Basic Map Display building blocks. These references belong to one of the point locations together with properties such as internal, external, exit, or entry.

The `TmcTileReferenceTable` must include all levels defined in the Routing and Basic Map Display building blocks. Tiles of higher levels must include all features of the corresponding tile on lower levels.

The local TMC location attribute can store either the TMC location code as defined by the road authority, or the location ID defined in NDS. The latter information requires more space, but might be faster as the additional information about the location table number etc. is not required.

Note Assignments of area locations and segment locations to routing and basic map display features will not be stored.

9.2 Looking up TMC Location Information

The `TmcLocationTableIdTable` and the `TmcLocationListTable` are the entry points for decoding TMC messages. They are built for each update region and must relate to the content of the local TMC attribute lists.

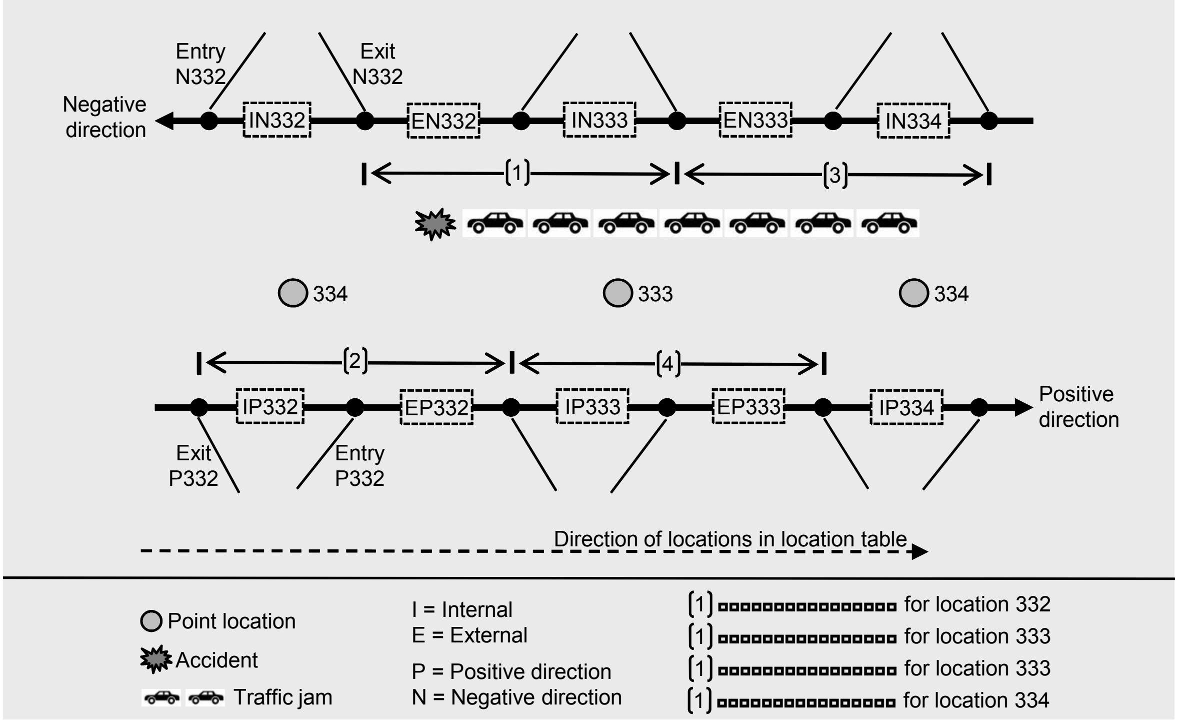
The ID of a location table, identified by the location table number, country code, and extended country code of the TMC message, can be looked up in the `TmcLocationTableIdTable`. This ID can be used to find all TMC location codes (for area, segment, and point locations) in the `TmcLocationListTable`.

The information about which radio station is sending TMC messages in a certain area can be found via the tiles defined in the `TmcStationTileListTable` and the related `TmcStationListTable`.

9.3 Calculation of Distances between Locations

Figure 9-1 depicts, how a Compiler calculates the distance between two TMC point locations, which are stored in the `TmcPointLocationTable` in the `distanceToPosLoc` and `distanceToNegLoc` columns.

Figure 9-1 Calculation of distances between locations

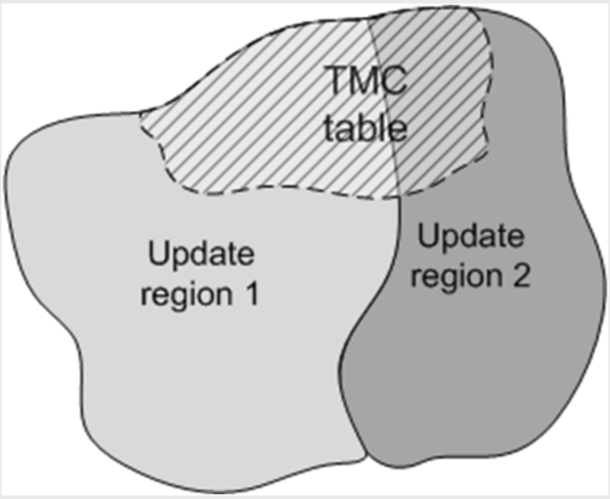


The respective road side representing positive or negative direction is defined according to the TMC standard. A traffic jam would also build up in this direction. The distance between two point locations is calculated as the sum of the length of all corresponding links on level 13.

9.4 Split TMC Location Tables

In most cases the extent of a location table is equal to or smaller than the extent of the update region it belongs to. It is possible that a location table has to be stored in two update regions. This is depicted in Figure 9-2.

Figure 9-2 Split TMC location table



In such a case, the information for all locations of the TMC location table is stored in both update regions. The `TmcTileReferenceTable`, the `TmcExternRefTable`, and the `TmcTileTable` only store the information specific to the respective update region.

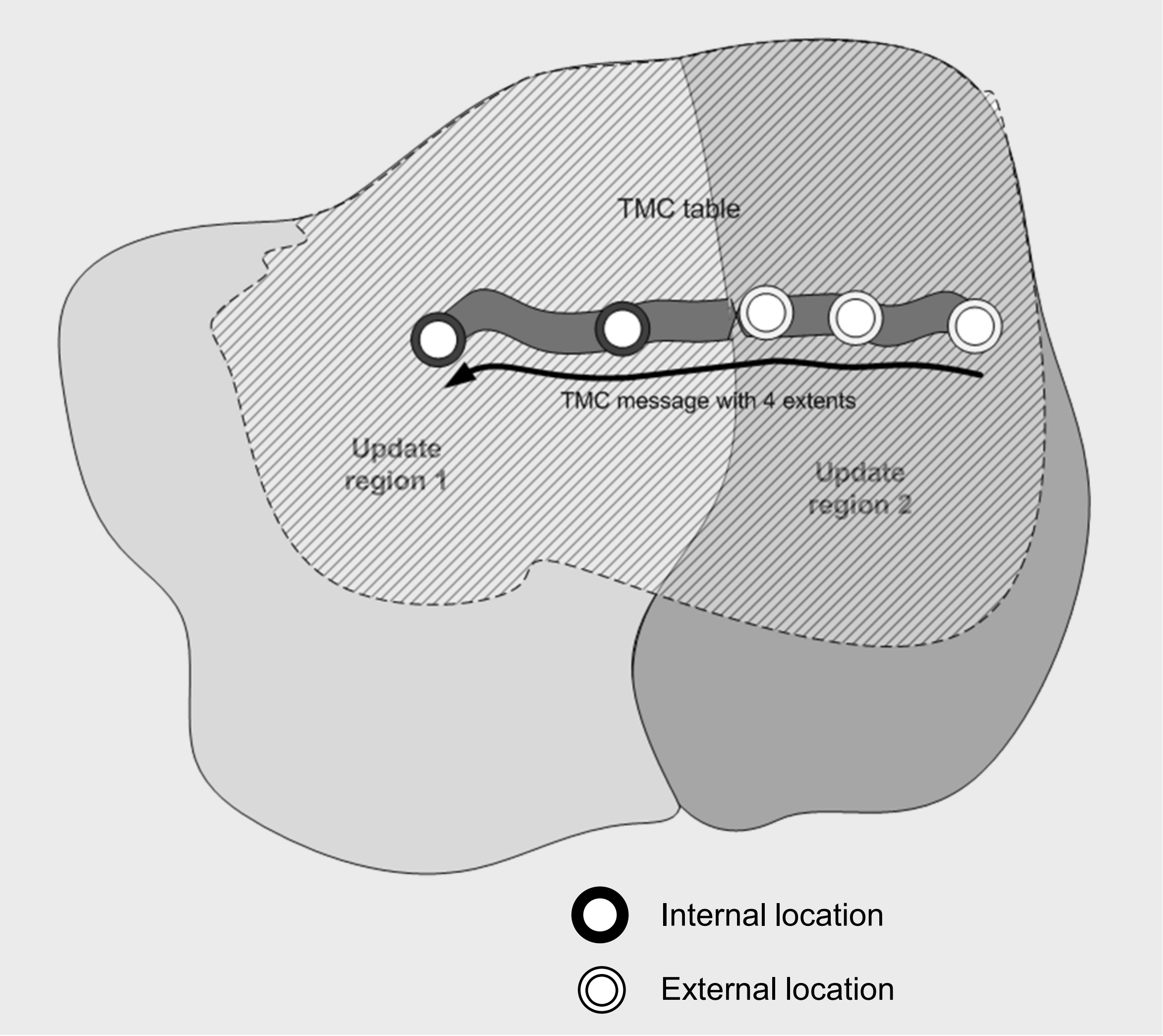
In the `TmcExternRefTable` of update region 1, all location IDs of the locations located in update region 2 and the ID of the other update region are stored. The same applies to update region 2.

It is important that all other data (that is, all except the `TmcTileReferenceTable` and the `TmcTileTable`) is stored in both update regions, so that a complete look-up of locations is guaranteed even if only one update region is installed on the sytem.

Example for Split TMC Location Tables

A TMC message starts at a point location based in one update region and its extent reaches into the current update region. With the normal look-up, all required data can be found and the information from the `TmcExternRefTable` shows where the links located in other update regions can be found.

Figure 9-3 Example for split TMC location table



10 POI Building Block

NDS databases may contain Points of Interests (POIs), which are stored in an integrated or non-integrated POI building block. This chapter introduces the topics that need to be considered to ensure interoperability for POIs. It contains information about the POI metadata, the compilation of third-party POIs or non-integrated POI building blocks, and references to other building blocks of an NDS database.

For general information about the POI concept for NDS, refer to *NDS – Format Specification*, 13 *POI Building Block* on page 261.

10.1 Metadata

There are two possible types of a POI building block:

- Integrated POI building block: In this case, the POI data supplier can access the mandatory building blocks of the NDS database during compilation and can therefore tightly couple the POIs with the data in the mandatory building blocks. The reference to a link, for example, may be stored as a part of POI data in order to assign the POI to an exact location in the routing network.
- Non-integrated POI building block: POI data suppliers without access to the mandatory building blocks of an NDS database during compilation cannot create direct references between POIs and features of the other building blocks. In this case, the application can create the connection by using the latitude/longitude encoded location of the POI and thus connect the POI with the routing network.

For more information on integrated and non-integrated POI building blocks, refer to *NDS – Format Specification*, 13.1 *Building Block Structure and Content* on page 261.

Filling the Metadata of Integrated POI Building Blocks

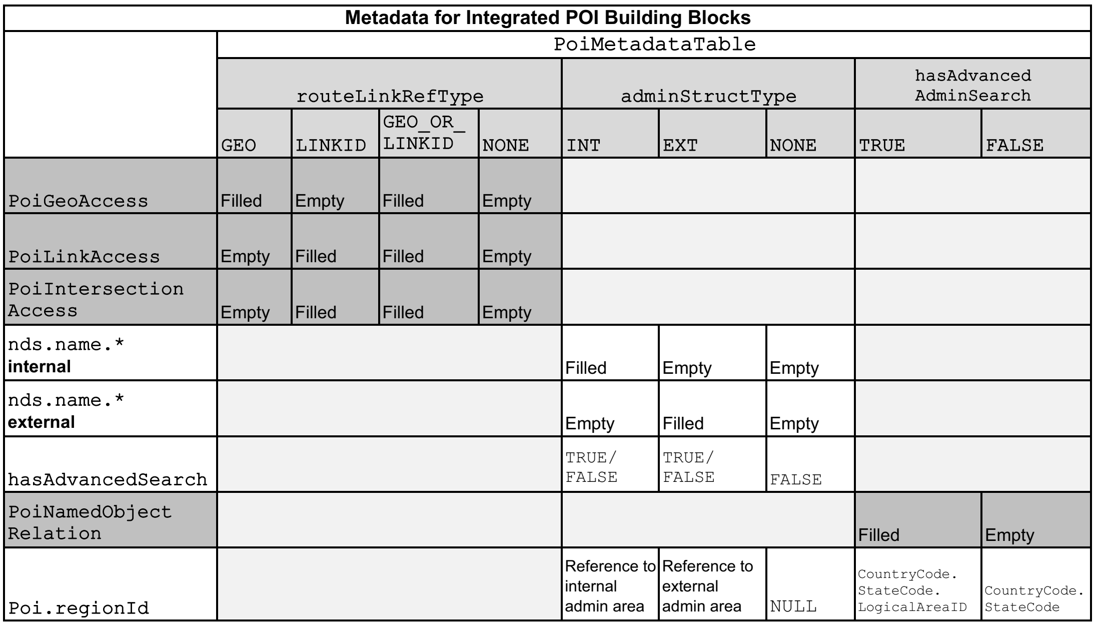
The metadata of integrated POI building blocks is filled as follows:

- `PoiMetadataTable.routeLinkRefType`
  - `GEO`: The `PoiGeoAccess` table may be filled; the `PoiLinkAccess` and `PoiIntersectionAccess` tables must be empty
  - `LINKID`: The `PoiLinkAccess` and `PoiIntersectionAccess` tables may be filled; the `PoiGeoAccess` table must be empty
  - `GEO_OR_LINKID`: The `PoiGeoAccess`, `PoiLinkAccess` and the `PoiIntersectionAccess` tables may be filled.
  - `NONE`: The `PoiGeoAccess`, `PoiLinkAccess`, and `PoiIntersectionAccess` tables must be empty

- `PoiMetadataTable.adminStructureType`
  - EXT: The POI building block references an external administrative structure stored in the mandatory Name building block
  - INT: The POI building block contains an internal administrative structure stored in tables and structures of the `nds.name.*` package
  - NONE: The POI building block does not have an administrative structure
- `PoiMetadataTable.hasAdvancedAdminSearch`
  - `TRUE`: The `PoiNamedObjectRelationTable` may contain references to named objects representing country or state regions, or any administrative or non-administrative region.
  - `FALSE`: The `PoiNamedObjectRelationTable` must contain references to named objects representing country or state regions only. The value is also set to `FALSE` if `PoiMetadataTable.adminStructureType` is set to `NONE`.
- `Poi.regionId`
  - `NULL`: If the POIs are not related to regions or if `PoiMetadataTable.adminStructureType` is set to `NONE`
  - Reference to a country or state region defined in the mandatory Name building block; this is only possible if the `PoiMetadataTable.adminStructureType` is set to `EXT`
  - Reference to a country or state region defined in the POI building block; this is only possible if `PoiMetadataTable.adminStructureType` is set to `INT`
  - Reference to a logical area; this is only possible if the `PoiMetadataTable.hasAdvancedAdminSearch` is set to `TRUE`

Figure 10-1 gives an overview of the metadata for an integrated POI building block.

Figure 10-1 Metadata for integrated POI building blocks



Metadata for Integrated POI Building Blocks

	PoiMetadataTable								
	routeLinkRefType				adminStructType			hasAdvanced AdminSearch	
	GEO	LINKID	GEO_OR_ LINKID	NONE	INT	EXT	NONE	TRUE	FALSE
PoiGeoAccess	Filled	Empty	Filled	Empty					
PoiLinkAccess	Empty	Filled	Filled	Empty					
PoiIntersection Access	Empty	Filled	Filled	Empty					
nds.name.* **internal**					Filled	Empty	Empty		
nds.name.* **external**					Empty	Filled	Empty		
hasAdvancedSearch					TRUE/ FALSE	TRUE/ FALSE	FALSE		
PoiNamedObject Relation								Filled	Empty
Poi.regionId					Reference to internal admin area	Reference to external admin area	NULL	CountryCode. StateCode. LogicalAreaID	CountryCode. StateCode

Filling the Metadata of Non-integrated POI Building Blocks

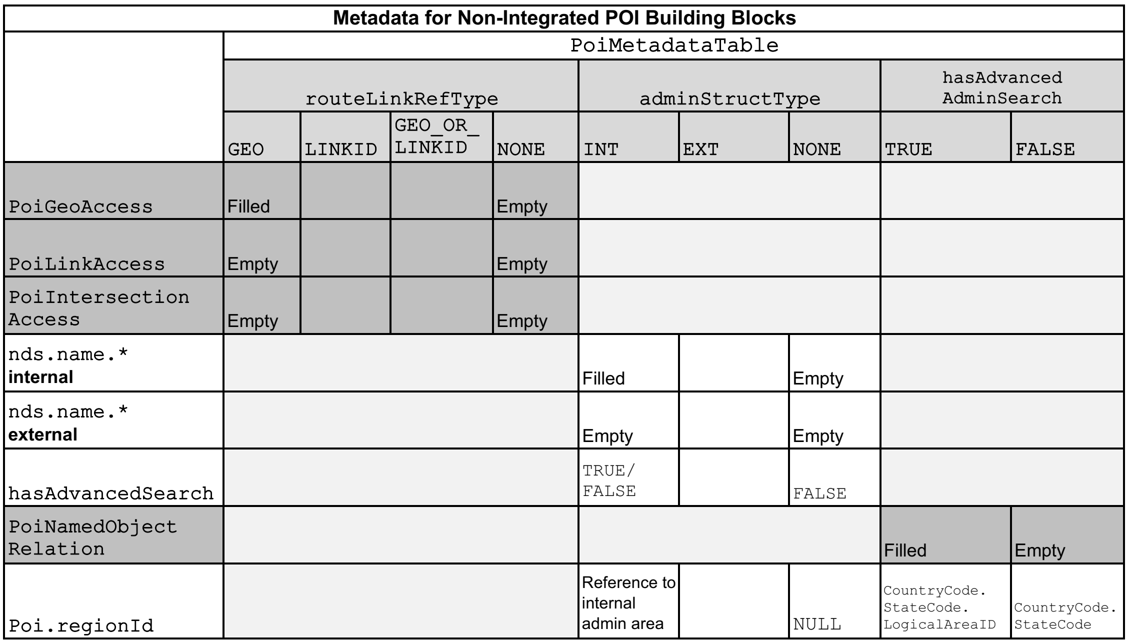
The metadata of non-integrated POI building blocks is filled as follows:

- PoiMetadataTable.routeLinkRefType
  - GEO: The PoiGeoAccess table may be filled; the PoiLinkAccess and PoiIntersectionAccess tables must be empty
  - NONE: The PoiGeoAccess, PoiLinkAccess, and PoiIntersectionAccess table must be empty
- PoiMetadataTable.adminStructureType
  - INT: The POI database contains an internal administrative structure stored in tables and structures of the nds.name.* package
  - NONE: The POI database does not contain an administrative structure
- PoiMetadataTable.hasAdvancedAdminSearch
  - TRUE: The PoiNamedObjectRelationTable may contain references to named objects representing country or state regions, or any administrative or non-administrative region.
  - FALSE: The PoiNamedObjectRelationTable must only contain references to named objects representing country or state regions. The value is also set to FALSE, if PoiMetadataTable.adminStructureType is set to NONE.
- Poi.regionId

- NULL: If the POIs are not related to regions, or if `PoiMetadataTable.adminStructureType` is set to `NONE`
- Reference to a country or state region defined in the mandatory Name building; this is only possible if `PoiMetadataTable.adminStructureType` is set to `INT`
- Reference to a logical area; this is only possible if `PoiMetadataTable.hasAdvancedAdminSearch` is set to `TRUE`

Figure 10-2 gives an overview of the metadata for a non-integrated POI building block.

Figure 10-2 Metadata for non-integrated POI building blocks



Metadata for Non-Integrated POI Building Blocks									
	PoiMetadataTable								
	routeLinkRefType				adminStructType			hasAdvanced AdminSearch	
	GEO	LINKID	GEO_OR_ LINKID	NONE	INT	EXT	NONE	TRUE	FALSE
PoiGeoAccess	Filled			Empty					
PoiLinkAccess	Empty			Empty					
PoiIntersection Access	Empty			Empty					
nds.name.* internal					Filled		Empty		
nds.name.* external					Empty		Empty		
hasAdvancedSearch					TRUE/ FALSE		FALSE		
PoiNamedObject Relation								Filled	Empty
Poi.regionId					Reference to internal admin area		NULL	CountryCode. StateCode. LogicalAreaID	CountryCode. StateCode

10.2 Country and State Code References

POIs are connected to regions by means of the `PoiTable.regionId` field, which is a primary POI attribute (see *NDS – Format Specification*, 13.2.1 *Primary POI Attributes* on page 267). The field value may also be set to `NULL`. In this case, a POI is not related to any region and would thus not be found in region-based searches.

Table 10-1 describes the field structure for any values that do not equal `NULL`.

Table 10-1 Country and state code reference values

Value	Description
Byte 0 – Country code must be greater than 0	Reference to a record in the PoiNamedObjectRelationTable (via region ID) storing a reference to the named object of a country The named object may be stored either within the POI building block or within the Name building block. This is determined in adminStructureType. If such a named object does not exist, the value of the PoiTable.regionId must be set to NULL. The country-based search is defined as follows: Use the country named object ID and search for the country region ID (R_ID) in PoiNamedObjectRelationTable. The byte 0 value for R_ID shall be set to a value greater than 0, while byte 1-3 values are 0, for example, 0x08000000. Since the last 3 bytes are zero, all POIs for a country can be found by searching for a region ID between 0x08000000 and 0x08FFFFFF in the POI table.
Byte 1 – State code or 0	Concatenation of a country code and the state code (byte0.byte1) is a reference to a record in the PoiNamedObjectRelationTable (via region ID) storing a reference to the named object for a state. This named object may be stored either within the POI building block or within the Name building block (determined by adminStructureType). If such a named object does not exist, the state code must be set to 0. The state-based search is defined as follows: Use the state named object ID and search for the state region ID in the PoiNamedObjectRelationTable. The byte 0 and byte 1 value for R_ID shall be set to a value greater than 0, while byte 2 and byte 3 values are set to 0, for example 0x08110000. Since the last 2 bytes are zero, all POIs for a state can be found by searching for a region ID between 0x08110000 and 0x0811FFFF in the POI table.

Value	Description
Byte 2-3 – Logical area ID or 0	Identifies a logical area A logical area is defined by a unique combination of administrative areas on the lowest level of the administrative hierarchy (which covers the whole map), and named areas for the given map. A set of regions for a given map is generated by the following operations: – Intersection of all administrative areas on the lowest level of the administrative hierarchy and named areas – Identification of all disjunctive areas resulting from the intersection For the resulting set of regions, the following must be true: – There are no two regions, which intersect – The union of all regions equals the extent of the original map, thereby spanning the whole area of that map **Note** Postal areas are not included in a unique combination of areas since this would result in a too large number of regions. For generating region sets from the source database, the following approach is recommended: – For each POI, identify the administrative area that the POI belongs to and that has the lowest level in the administrative hierarchy. – For each POI, identify all named areas it belongs to (if there are any) – Group the POIs by common area groups The concatenation of country code, state code, and logical area ID (`byte0.byte1.byte2.byte3`) is a reference to a record in the `PoiNamedObjectRelationTable` (via region ID). This record stores a reference to the named object of an administrative area that represents a logical area. If such an area does not exist, the logical area ID must be set to 0. A search in an arbitrary area is defined as follows: Use the named object ID and search if a region ID (R_ID) can be found in the `PoiNamedObjectRelationTable`: – If the search returns only one `R_ID`, byte 0 and byte 1 are set to a value greater than 0, byte2 and byte3 must not both be set to 0, for example 0x08110015. Since the last 2 or 3 bytes are **not** zero, all POIs for a region can be found by searching for a region ID that equals 0x08110015 in the POI table. – If the search returns several `R_ID`, byte 0 and byte 1 for all are set to a value greater than 0, byte2 and byte3 must not be both set to 0, for example 0x08110021, 0x08110025, 0x08113015. Since the last 2 or 3 bytes are **not** zero, all POIs for a region can be found by searching the POI table for a region ID that is in (0x08110021, 0x08110025, 0x08113015).

10.3 Use of PoiTable.regionId in Searches

The PoiTable.regionId field is used when a user searches for all POIs within a defined administrative area. The application can use the administrative area structure nds.name.* (stored either in the Name or the POI building block) to find the named object that represents that defined administrative area.

If the named object is a country region, the application can search the corresponding country code by reading the region definition BLOB (nds.overall.metadata.UpdateRegionTable.RegionDefinitionBlob). A search for all POIs contained in this region can be executed by means of a "range-based" query: If the country code is, for example, 0x57, the search query must find all POIs with a PoiTable.regionId between 0x57000000 and 0x57FFFFFF.

The same applies if the named object is a state region.

If the named object is an administrative area with a lower level than that of a state, or, in case there are no states, a lower level than that of a country, the application can search for all region IDs of the logical areas related to the named object by using the PoiNamedObjectRelationTable. In a second step, the application can then search for all POIs related to those region IDs. This query can, however, be less performant than the queries for country- or state-related searches.

10.4 POI Category ID

POI categories are used to group POIs under a generic topic, for example, all POIs for petrol stations may be assigned to the POI category petrol_station. POI categories are uniquely identified by the POI category ID which is stored in the catId field of the PoiCategoryTable.

The POI category ID is used as follows:

- Identifier for references between a POI and a POI category in the PoiServiceLocationTable
- Input parameter for relational queries. For searching all restaurant POIs, for example, the ID for the POI category for restaurants would be used as input parameter for the query on the PoiServiceLocationTable.

The catId is defined as an unrestricted 16-bit integer value. This enables the use of optimization techniques for the compilation of a POI database. For strictly hierarchical categorization, for example, methods for managing hierarchical data in a relational database may be applied.

10.5 POI Standard Categories

To ensure interoperability, NDS predefines POI standard categories, that is, predefined POI categories containing specific semantic information for a POI category. The semantic is defined in the `standardCat` attribute which is stored in the `PoiCategoryTable`. The application can use the semantic information for interaction with the driver.

POI databases can contain both, categories with standard semantics and without standard semantics. All categories must be stored in the `PoiCategoryTable`.

For interoperability of custom POI categories, NDS provides a generic standard category (`POICAT_NDSGENERAL`). All custom POI categories shall have an ancestor of another predefined specific POI standard category in the POI category DAG. At least this shall be `POICAT_NDSGENERAL` if no other standard category is applicable.

Note For referencing POI categories, the `catID` must be used. The `standardCat` is used to determine the semantics of a given category.

Note The category ID for POIs which are activation points (meaning they are related to the parent POI as `PoiRelationType.ACTIVATION`) and access points (meaning they are related to the parent POI as `PoiRelationType.ACCESS`) shall always be set to `POICAT_ACTIVATION_POINT` and `POICAT_ACCESS_POINT` respectively.

For more information, refer to the following:

- Documentation of the `standardCat` attribute in the *NDS – Physical Model Description* and *NDS – Format Specification*, *POI Standard Categories* on page 278
- *NDS – Format Specification*, 13.4 *POI Categories* on page 275

Example of the Use of POI Standard Categories

A POI is assigned to the POI category `parking_area`. This POI category is defined as a POI standard category. The application can use the information to recommend a stop on a nearby parking area after a driving time of more than three hours.

10.6 POI Access Tables

Simple access points define a geographic location which can be used as an entry point for accessing the POI. In addition, simple access points provide a mechanism for straightforward referencing from the POI building block to the Routing building block.

For more information, refer to *NDS – Format Specification*, 13.3.2 *Access Relations between POIs* on page 273.

Access points are stored in the following tables:

- `PoiGeoAccessTable`: Used for storing POI entry points
- `PoiLinkAccessTable` and `PoiIntersectionAccessTable`: Used for storing POI entry points and also for straightforward references to the Routing building block, which improve application performance

For more information on access points, refer to *NDS – Format Specification*, 13.3.2 *Access Relations between POIs* on page 273.

For populating the POI access tables, the following rules apply:

- To avoid redundancies, the `PoiGeoAccessTable` shall **not** contain simple access points whose location coincides with the location of the respective POI.
- The `PoiLinkAccessTable` and the `PoiIntersectionAccessTable` may, however, contain simple access points whose location coincides with the POI location. In this case, it makes sense to store direct references to routing links for POIs.

10.7 Scale Ranges for POI Categories and POI Icons

Scale ranges may be assigned to POI categories and to POI icons. The scale range is defined by means of the `scaleLevelId` which references the scale range defined in the `PoiScaleLevelTable`.

For more information, refer to *NDS – Format Specification*, 13.4.1 *Attributes of POI Categories* on page 277.

The use of the respective scale range is defined as follows:

- The scale range assigned to POI categories defines the recommended scale range for displaying icons of that specific category by the application. The POI category `airport`, for example, has a different scale range than the POI category `restaurant` because airports are usually still visible when the map is zoomed out, whereas restaurant POIs would only be displayed when the map is zoomed in.
- The scale range assigned to icons defines the recommended scale for a specific icon. Icons with different sizes may be grouped in one icon set. For each icon size, a suitable scale range is assigned.
- To avoid that POI categories disappear when zooming in a map, the minimum scale denominator for POI categories should be set to `1`. For POI icons, however, the minimum scale denominator may be greater than `1`. This enables the application to replace an icon with a larger icon when zooming in a map.
- The scale range defined for a category must comprise the scale ranges of all icons assigned to the POI category. For example, the POI category `hotel` contains three icons with the following scale denominator: Icon 1: 1 – 20000, Icon 2: 20001 – 50000, Icon 3: 50001 – 100000. The scale range for the POI category `hotel` would then be defined as 1 – 100000.
- If the scale range for a POI category is set to `0`, the POI category would be visible on all scale levels.

10.8 Handling of Predefined Secondary POI Attributes

NDS provides a list of predefined secondary POI attributes. Although data providers can define additional secondary POI attributes, the predefined secondary POI attributes as defined by NDS must be used if the same semantic is intended.

Example A POI has an icon reference and a multimedia reference. When adding a new icon for this POI, the icon reference must be used, and not the generic multimedia reference, as the latter is intended for pictures or videos attached to the POI.

10.9 References from POIs to 3D Objects

A POI may have implicit or explicit references to 3D objects:

- The POI may reference the 3D object explicitly by storing the optional flexible attribute `THREE_D_LANDMARK_REF` in the list of secondary attributes. The attribute value is the spatial subtree ID (`spatialSubTreeId`) of the spatial tree BLOB stored in the `SpatialSubTreeTable`.
- The POI may reference the 3D object implicitly by means of the geographic position of the POI. The application uses this information to find the spatial tree node of the bounding box with a center point nearest to the POI position. In some cases, the search may, however, return the wrong bounding box for the POI. This can occur, for example, in areas with buildings of non-rectangular shape, which are spatially proximate to each other.

  To increase the accuracy of implicit references and avoid referencing the wrong bounding box, the NDS compiler may add the secondary attribute `THREE_D_LANDMARK_DIFF_VECTOR` to the POI. The application resolves the reference by shifting the POI position for the given 3D landmark vector, and then searches for the 3D object, which is as close as possible to the shifted position.

Related Topics:

- 14.3 *References into 3D Objects Building Block* on page 198
- *NDS – Format Specification*, 17.8 *References to 3D Objects* on page 344

10.10 References from POIs to Basic Map Display Features

POIs may reference features of the Basic Map Display building block by storing the following optional attributes in the list of secondary attributes:

- `BMD_POINT_REFERENCE`
- `BMD_LINE_REFERENCE`
- `BMD_AREA_REFERENCE`

The values of these attributes unambiguously identify a BMD feature, regardless if a POI and the referenced feature are located on the same tile or not.

This reference is typically used for highlighting the features associated with a POI, for example, a park area or a river nearby the POI's position.

10.11 Virtual POI Tiles

The POI virtual tile table (DataScript location: `nds.poi.main` > `PoiVirtualTileTable`) associates a packed tile ID with a range of service location IDs (`minId`, `maxId`) which represent all POIs located in that tile.

A packed tile ID may contain IDs of tiles on any level; however, the `PoiVirtualTileTable` shall contain only one level. The more detailed the level is, the more precise the spatial query will be. Also, loading the POI data for one tile from the POI service location table will be faster, as fewer POIs need to be loaded per tile.

Note	Interoperability issues for virtual POI tiles are still under investigation. More information will be added as soon as it becomes available.

Related Topics:

- 13.6.1 *POI Service Locations* on page 281
- *NDS – Format Specification*, 13.6.3 *Virtual Tiles for POIs* on page 285

11 Speech Building Block

This chapter introduces the topics that need to be considered to ensure interoperability for speech data in an NDS database.

11.1 Preference Order of Phonetic Transcriptions

A name string may have multiple phonetic transcriptions. To support the application in choosing the correct transcription for a respective name string, a preference order is specified.

For phonetic transcriptions for POIs, this order is defined in the `orderOfPreference` column of the `PoiAttrNameToPhoneticTraTable`.

For phonetic transcriptions for named objects, the preference order is implicitly specified by storing the entries in the order of preference in the `NamedObjectPhoneticTranscriptionListBlob`. The first entry is the most preferred phonetic transcription, the second entry is the second preferred phonetic transcription, and so on.

In case of phonetic transcriptions for TMC and TPEG, in the `orderOfPreference` column of the `TmcLocationPhoneticTraTable`, the `TmcEventPhoneticTraTable`, and the `TpegEventPhoneticTraTable`.

12 Digital Terrain Model Building Block

A digital terrain model (DTM) is a digital representation of ground surface topography or terrain which may be modeled by a grid of rectangles (raster) or as a triangular irregular network (TIN). In NDS, two possible representations of DTM are available:

- Height maps
- Batched Dynamic Adaptive Meshes (BDAM)

For more information on the two representations of DTM, refer to *NDS – Format Specification*, 15 *Digital Terrain Model Building Block* on page 297. This chapter deals with interoperability issues for DTM by height maps.

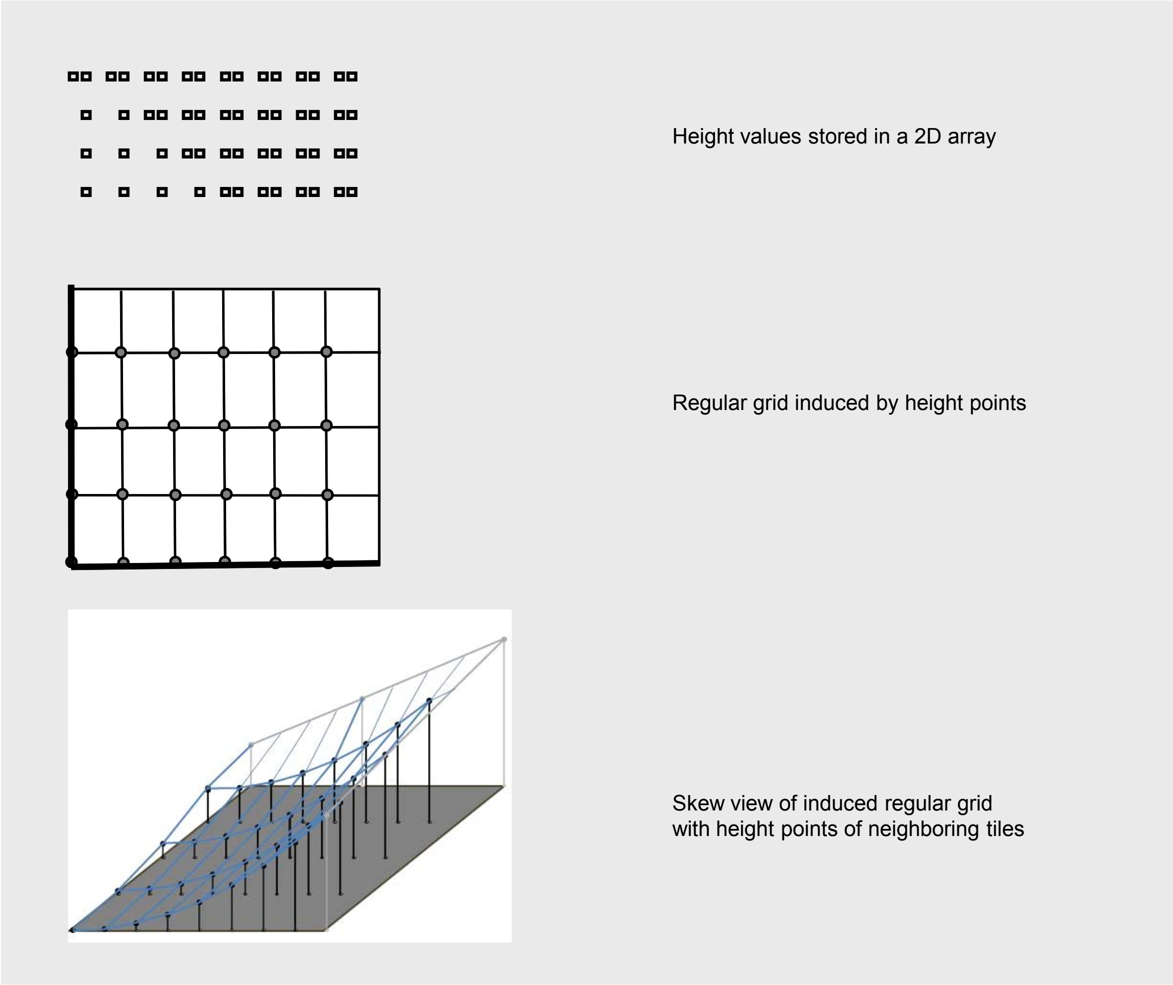
12.1 Height Map Tile Dimensions

Height map tiles consist of a regular grid of $(2^m)x(2^n)$ rectangles with a height value assigned to their south-west corners.

For more information on regular grids, see *http://en.wikipedia.org/wiki/Regular_grid*.

No relation exists between the grid dimensions of different tiles, even if they are neighbors on the same level. Grid points on the north and east boundaries are retrieved from height field tiles that are neighbors to the east, the north and, eventually, to the north-east.

Figure 12-1 Height map tile with height values


Height values stored in a 2D array
Regular grid induced by height points
Skew view of induced regular grid
with height points of neighboring tiles

Grid dimension of neighboring tiles can be different and it can, therefore, be necessary to create grid points via interpolation on the east or north border, or to modify the height values of points on the western or southern border in case the number of the rectangles of neighboring tiles is smaller along the respective border than the extent of rectangles of the current tile. As the interpolation is done at runtime, it is out of scope for NDS.

Because the grid dimensions are restricted to powers of two, the calculation or correction of height values can be done efficiently at runtime, for example, in response to a dynamic change of levels of detail of more distant height map tiles.

Defining the height value to be associated to the south-west corner of rectangles prevents displaced terrains. This effect can occur in tiles with a low level of detail and a prominent mountain peak. To compensate for scaling effects, the horizontal dimension can be made independent of the vertical dimension. This is important for mapping textures or images onto the terrain.

Optimization

For tiles with identical heights, the tile can be marked as empty and one height value shall be given. This is the case, for example, for tiles representing an area that is completely covered by water.

12.2 Storing Height Map Data

According to the south-west rule, the northern and eastern edge of a tile belongs to the neighboring tiles and the data on those edges is, therefore, not stored in the tile (see Section 8.1.1 *Point Features* on page 149). For height maps, the same rule applies: Height points on the northern and eastern edges of a tile are not stored as part of the data of that tile, but of the neighboring tile. If a texel is mapped onto a DTM, the height points on the eastern and northern boundary of a tile have to be extracted from the height maps of the neighboring tiles to the east, the north, and the northeast. For information on texel, see *http://en.wikipedia.org/wiki/Texel_%28graphics%29*.

Not storing height values which are on the northern and eastern edges of the tile can affect performance, because the application is forced to load up to three more tiles to render one single tile. This disadvantage, however, is outweighed by the simpler handling of adjacent height maps with differing resolution, as well as the simpler handling of height map updates.

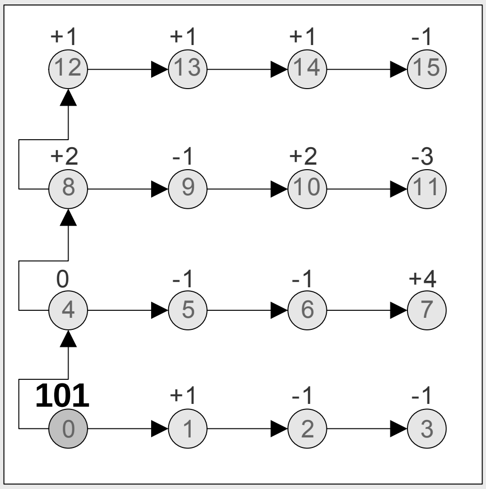
12.3 Traversal of Height Points for the `SIMPLE_ARRAY`

Height map data may be stored as `SIMPLE_ARRAY` (DataScript location: `nds.dtm.heightmap` > `SimpleArrayHeightMapData`). For height map data in simple arrays, the following applies:

- The first point (index 0) of the first line of the height map is located in the south-west corner of a tile. The second point is the point to the right of point 0, etc. The first point of the second line is the point to the north of the first point 0, etc.
- The first point in the array stores the absolute height value.
- The first point of each line stores the height difference to the first point of the previous line.
- The other points in the array store the height difference to the respective previous points.

Figure 12-2 shows the traversal of height points in simple arrays.

Figure 12-2 Traversal of height map points



12.4 Online Calculation of Coord Shift and Error Value for BDAM

The `coordShift` and the error value for BDAM can be calculated online from the level of detail:

`coordShift` = 32 – `NdsLevel`– `coordWidth` | coordShift ≥ 0

13 Orthoimages Building Block

Orthoimages are raster images usually reproduced from aerial or satellite photographs. NDS supports multi-resolution display for orthoimages, which enables the application to display the images of different levels and of different resolutions together in a perspective projection of a 3D scene with high-resolution images in the front and low-resolution images in the background.

13.1 Level of Detail

The level of detail (LOD) is used in computer graphics to decrease the complexity of a 3D object. The less important a detailed resolution is for the viewer, the fewer details need to be displayed. The level of detail for orthoimages is controlled by means of combining images from different levels on the same zoom level. This is achieved by using the mipmapping technique.

For general information on mipmapping, refer to *http://en.wikipedia.org/wiki/Mipmap*.

Orthoimages may be rendered on 2D, 2.5D, and 3D maps:

- For 2.5D and 3D map display: The orthoimages near the point of view are loaded from the lower levels (“smaller tiles”) and the orthoimages further away are loaded from the higher levels (“bigger tiles”). Using only one level is not recommended in this case, because the number of images which need to be loaded and rendered would increase considerably towards the horizon.
- Mipmapping may also be used for 2D map display: The orthoimages near the center point of the view are loaded from the lower levels while those towards the edges are loaded from the higher levels.

13.2 Resolution

The definition of the resolution for orthoimages is important because it affects the rendering performance and the quality of the images on the screen:

- For good rendering performance, the resolution should be kept constant. A variable image size can cause the following issues:
  - Estimation of GPU memory consumption is difficult. The memory requirement would not depend on the screen resolution and camera parameters only, but also on the database content that cannot be calculated in advance.
  - GPU memory fragmentation due to different texture sizes
  - A large texture can slow down low-end GPUs more than several smaller textures
- The variable width of NDS tiles, which depends on the latitude, can, however, require a variable horizontal resolution. Especially for 3D maps, a constant horizontal resolution can lead to undesired effects like Moiré or Blurring as well as to decreased rendering performance.

In order to optimize the image quality and the performance, the vertical resolution should be kept constant, while the horizontal resolution may vary in latitude.

The following rules are defined to support the level of detail concept (see Section 13.1 *Level of Detail* on page 191) as well as to address perfomance and image quality:

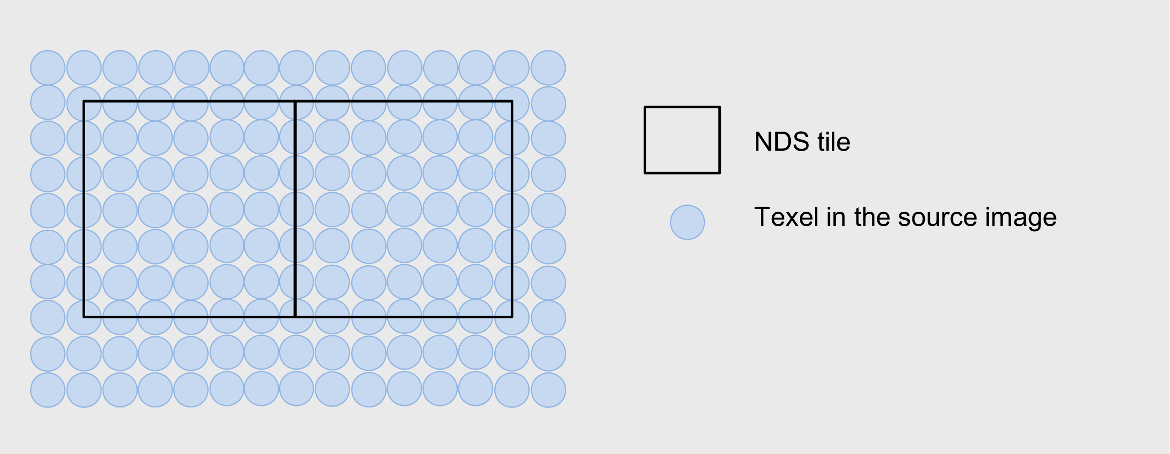
- To support mipmapping, the vertical and horizontal resolution must be defined as "power of two" values (see *http://en.wikipedia.org/wiki/Power_of_two*).
- The same vertical resolution is defined for all images in the metadata of the Orthoimage building block (DataScript location: `nds.overall.metadata` > `OrthoImageMetadata` > `verticalPixResolution`).
- The horizontal resolution is defined as smaller than or equal to the vertical resolution. This also supports non-uniform and uniform mipmapping.

13.3 Handling Orthoimage Tiles

Handling Image Texels at Tile Borders

Orthoimages use the NDS tiling scheme. They are geo-referenced to WGS 84 and are resampled to tiles of an appropriate level, depending on coverage and pixel resolution. The extent of NDS tiles is expressed in WGS 84 coordinates. The extent of the original orthoimages, however, is expressed in image coordinates, meaning rows and columns of texels. In most cases, the NDS tiling scheme does not ideally match the coordinates of the source image. Thus some texels overlap tile borders, as shown in Figure 13-1.

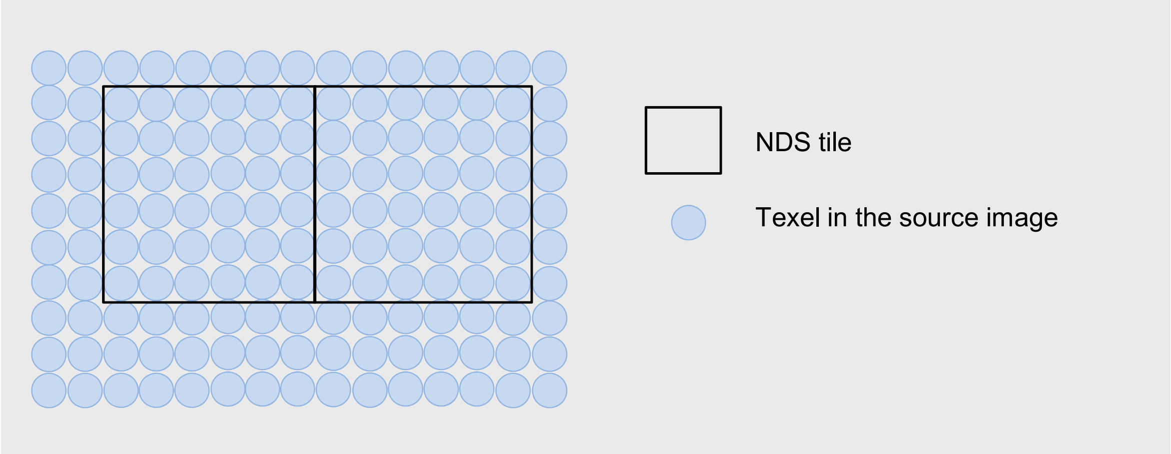
Figure 13-1 Overlapping between NDS tile borders and source image texels



The compiler must handle texels that are located on tile borders in such a way that they are either duplicated (stored in both NDS tiles) or assigned to the tile on the right. Duplicating texels increases data size, while assigning texels to one tile only causes a shift for individual texels, which possibly creates a mismatch between orthoimage data and basic map display or digital terrain data.

For this reason, it is recommended to compile orthoimages in such a way that the orthoimage tiles always ideally match the image coordinates, as shown in Figure 13-2. This can necessitate recompiling the source images.

Figure 13-2 Ideal match between NDS tile and image texels



Handling Orthoimage Tiles at Update Region Borders

If the content of orthoimage tiles overlaps update region borders, both update regions shall contain the complete orthoimage. If, for example, the orthoimage tile 42 lies partly in update region 1 and partly in update region 2, both update region 1 and 2 contain the complete orthoimage. If this redundancy is not wanted for reasons of storage space, orthoimages may be stored in a separate product database, which uses an update region schema that avoids incompletely filled tiles.

14 3D Objects Building Block

3D objects are three-dimensional representations of real world objects. NDS stores the 3D object data in nodes of a spatial tree structure. 3D objects are, therefore, not stored in correspondence with the NDS tiling scheme.

For more information on the modelling of 3D objects in NDS, refer to *NDS – Format Specification*, 17 *3D Objects Building Block* on page 317. This chapter deals with interoperability issues for 3D objects.

14.1 Partitioning of Spatial Trees

NDS uses a spatial index structure to access 3D object features, which are stored in nodes of the spatial tree. To enable applications to load only a small area, spatial trees can be partitioned, that is, they are split into a root subtree and one or more child subtrees (see Figure 14-1). Thus, the partitioning of spatial trees improves performance, and reduces memory consumption.

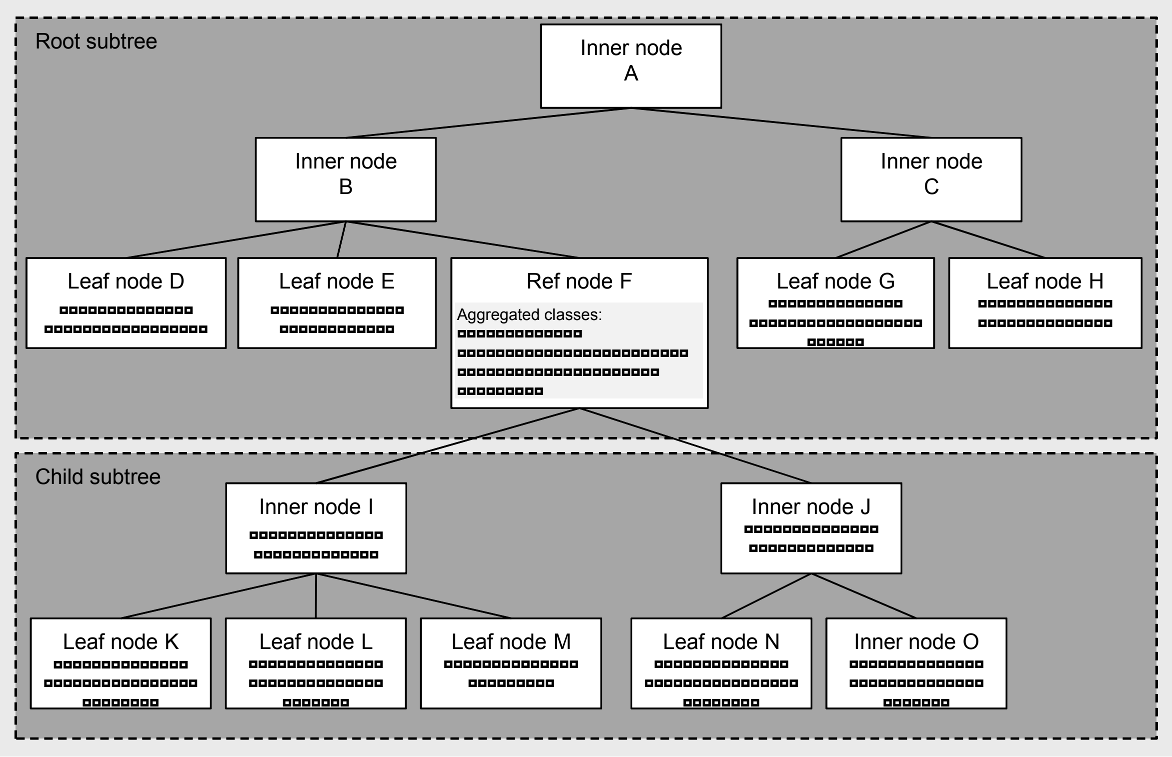
Note The spatial tree has only one root subtree per update region. Thus, ID = 0 is defined as root.

Partitioning is done by converting an inner node into a reference node (see *NDS – Format Specification*, 17.2 *3D Object Feature* on page 320). A reference node has the following additional properties:

- Child subtree ID for loading the BLOB of the child tree
- Aggregated 3D feature class to save the application from loading additional data from the child BLOBs at runtime

Figure 14-1 shows an example of a partitioned spatial tree.

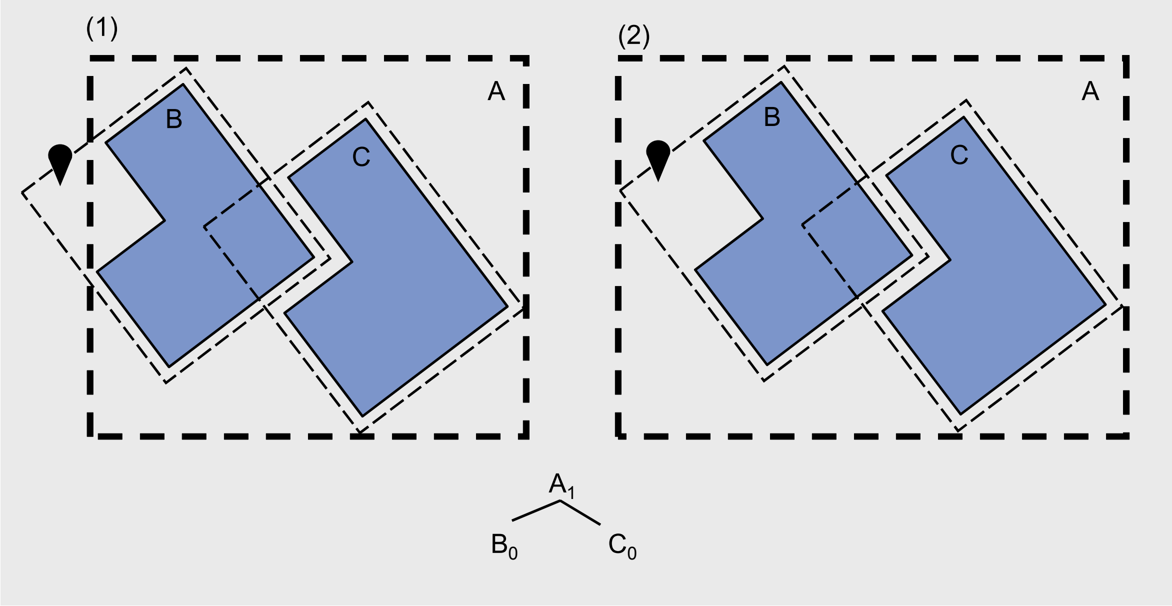
Figure 14-1 Partitioned spatial tree



14.2 Bounding Boxes

Bounding boxes are used to define the spatial extension of 3D objects. By means of the bounding box, an application has information about the spatial extension of the 3D objects of a specific area without actually retrieving the 3D object information itself. The size of a bounding box depends on the size of the 3D object: Bounding boxes contain the belonging 3D object completely for the conjunction of all level of detail representations; bounding boxes of parent nodes are the conjunction of the bounding boxes of the corresponding child nodes. This is depicted in Figure 14-2: The option depicted in (1) must not be used, option 2 shows the correct solution.

Figure 14-2 Parent node bounding box



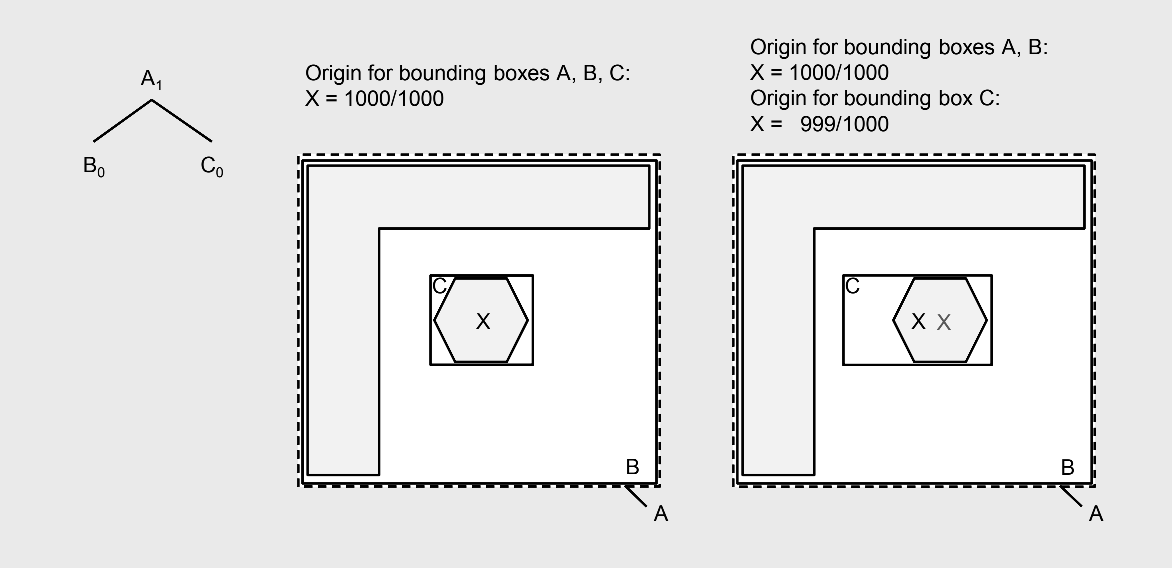
14.2.1 Bounding Boxes Sharing a Center Point

Bounding boxes may overlap. In special cases, the overlapping bounding boxes may share the same center point and the same object hierarchy level. This can cause problems for identifying a 3D object, for example, for references.

In case of bounding boxes, which share the same center point, the compiler must shift the center point coordinate of the bounding box by at least 1 coordinate unit. This is done by enlarging the bounding box to one side by two coordinate units. This is depicted in Figure 14-3.

Note	Overlapping bounding boxes may also occur for 3D objects with different parent nodes.

Figure 14-3 Overlapping bounding boxes


A1
B0
C0
Origin for bounding boxes A, B, C:
X = 1000/1000
C
X
B
A
Origin for bounding boxes A, B:
X = 1000/1000
Origin for bounding box C:
X = 999/1000
C
X X
B
A

14.3 References into 3D Objects Building Block

The object, which references into the 3D Objects building block, has to supply longitude and latitude information. The required accuracy depends on the respective overlapping bounding box with the same object hierarchy level. The closer the center points of the overlapping bounding boxes are, the more precise longitude and latitude information is required.

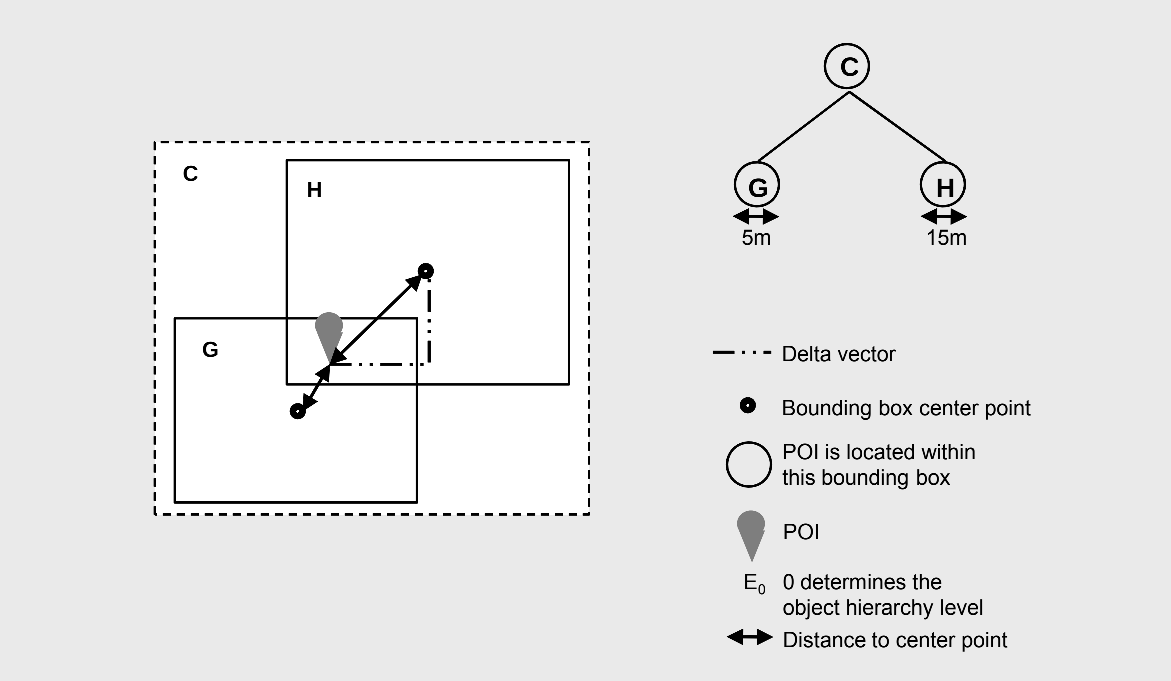
The default value for the object hierarchy level for such references is 0, as the reference would usually point to a building. The object hierarchy level is mandatory for searching the correct reference point.

The bottom height over ground value is optional. This value is required, if a part of a building is referenced, such as the bell tower instead of the whole church.

For POIs referencing into the 3D Objects building block, the compiler calculates the delta vector in addition to the own coordinate of the POI, in order to reach a precision that is high enough to determine the correct 3D object.

For the calculation of the distance to the centre, the Pythagorean theorem shall be used. The calculation shall be done in meters, and not in coordinates due to the difference between longitude and latitude in non-equatorial areas.

Figure 14-4 3D object references by means of coordinates and Delta vector



Related Topics:

- 10.9 *References from POIs to 3D Objects* on page 182
- *NDS – Format Specification*, 17.8 *References to 3D Objects* on page 344

14.4 Accuracy of Body Geometry – Recommendations

NDS recommends the following accuracy for level of details (LOD):

- LOD 0

  LOD 0 should have only rudimentary geometry information and no textures. Buildings should be aggregated if they are located close to each other and if they are of similar height. NDS recommends a tolerance of 5 to 10 meters to the actual geometry.
- LOD 1

  LOD 1 can have more geometry information, but should still not have textures. Buildings should be aggregated if they are next to each other, such as terraced houses. NDS recommends a tolerance of 5 meters to the actual geometry.

- LOD 2

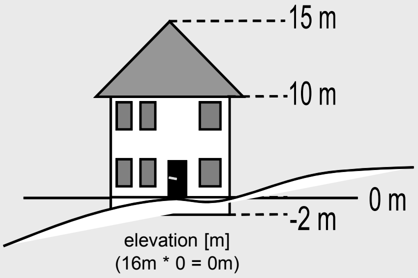
  For LOD 2, textures are optional and the geometry information includes roofs. NDS recommends a tolerance of 1 meter to the actual geometry.
- LOD 3

  For LOD 3, textures should be applied to 3D objects. NDS recommends a tolerance of less than 1 meter to the actual geometry.

14.5 Height over Ground

NDS recommends to have basements with negative height for drawing buildings (see Figure 14-5). Otherwise, buildings could hover in higher BDAM levels. NDS recommends a basement depth of 5 meters. This also applies to template buildings.

Figure 14-5 Negative height value for basements



14.6 Size Recommendations

To ensure a good performance for rendering 3D object data, NDS recommends the following sizes:

- Texture maps

  The recommended size for texture maps depends on the graphic processor. However, NDS recommends that neither width nor height exceed 1024.
- BLOBs

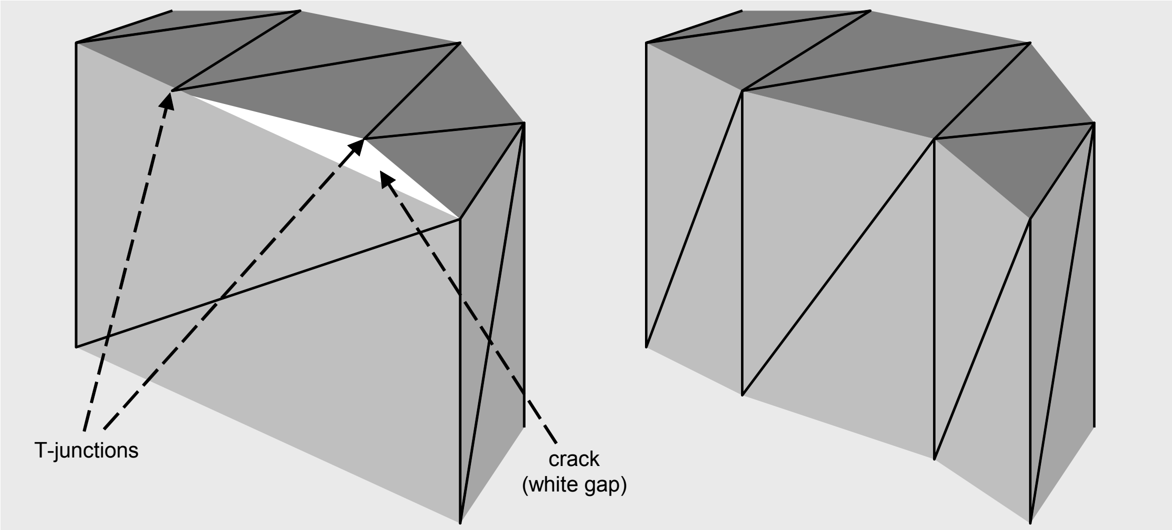
  NDS recommends to keep BLOBs rather small. The BLOB size should not exceed 100 kB.

14.7 Avoiding T-Junctions

A T-junction is formed when two polygons meet along the edge of a third polygon (see Figure 14-6) and can lead to rendering artifacts. In order to avoid discontinuities in form of cracks, the triangulation with T-junctions is not allowed. T-junctions can be avoided by adding further triangles.

T-junctions do not necessarily result in cracks. The results depend on the precision of the graphics hardware, and runtime parameters like the position of the camera.

Figure 14-6 Crack at T-junctions



14.8 Texture Formats

For texture formats, it is recommended to use only one format, which is, ideally, supported by the graphic chip. Otherwise, the formats have to be converted online.

15 Junction Views

Junction views give the driver realistic visual information of a junction and the required maneuver. The information for presenting a junction view image to the user of a navigation system is stored in NDS in a standalone building block.

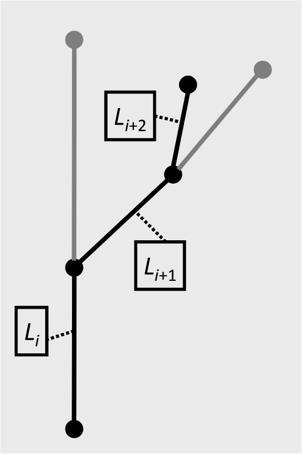
For more information on junction views, refer to *NDS – Format Specification*, 18 *Junction View Building Block* on page 349.

15.1 Drawing Junction Views

The flexible attribute `HAS_JUNCTION_VIEW` may be assigned to the links. It indicates that visual information for a link is available.

The sequence of links on level 13 representing the calculated route from a start point to a destination is called “route”. A route consists of an ordered sequence of links (L_i) with 1 ≤ i ≤ n. L_1 represents the starting link (`fromLink`) and L_n the last link (`toLink`) of the calculated route.

Figure 15-1 Route with sequence of links $L_i < L_{i+1} < L_{i+2}$



The navigation system can either show an image without maneuver information or with an arrow reflecting the required maneuver according to the route information (see Figure 15-1). Both options are illustrated in Figure 15-2.

Note For the image format `JvImageFormat.NDS_3D` the navigation system can decide whether to use 3D arrows (DataScript `JvArrow3D`) or not.

3D arrows can be stored as follows:

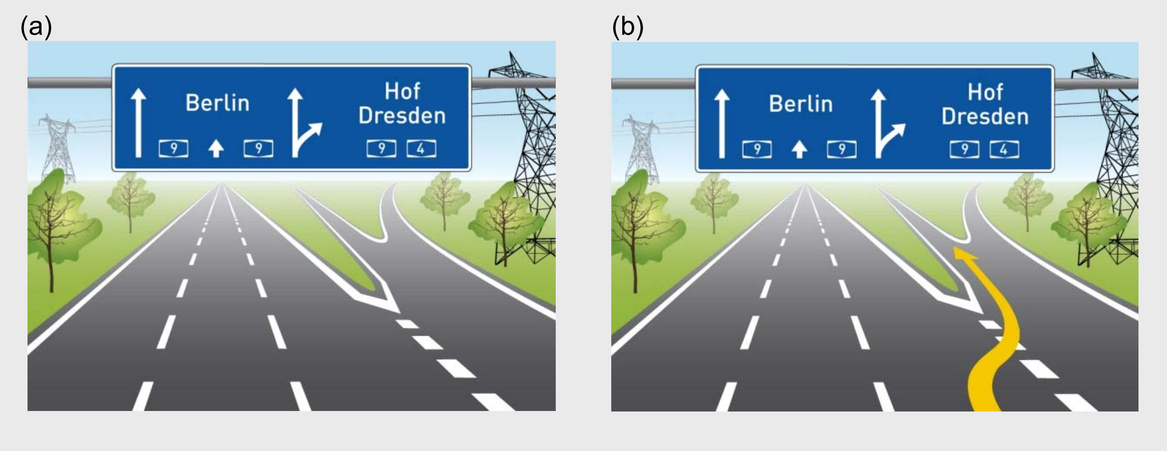
- 3D vectors (DataScript location: `nds.jv.images` > `JvArrowVector3D`)
- Splines (DataScript location: `nds.jv.images` > `JvArrowSpline`)

3D models of a junction view are represented by 3D objects and terrain model. The 3D objects are stored in the following tables:

- `JvSpatialSubTreeTable` (DataScript location: `nds.objects3d.main` > `SpatialSubTreeTable`)
- `JvBodyGeometryBlobTable` (DataScript location: `nds.objects3d.main` > `BodyGeometryBlobTable`)
- Terrain models are stored in `JvDtmBdamTileTable` (DataScript location: `nds.dtm.bdam` > `DtmBdamTileTable`)

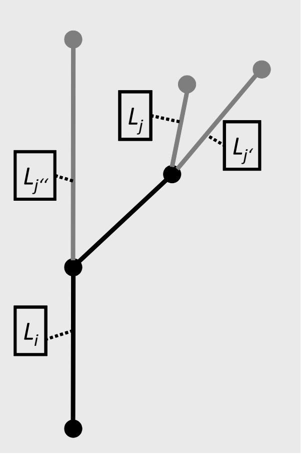
For the image depicted in Figure 15-2 (a) , the `fromLink` information stored in the `JvFromLink` table is sufficient. For the image in Figure 15-2 (b), the relation between `fromLink` and `toLink` must be defined unambiguously to enable the application to show the required maneuver and the arrow. This relation is stored in the `JvFrom2ToLink` table.

Figure 15-2 Junction view image with (a) fromLink information only and (b) fromLink, and toLink information



As the relation between `fromLink` and `toLink` bases on route calculation, the `toLink` information is not available during compilation. The route in Figure 15-4 shows three possible `toLink` (L_j , $L_{j'}$, and $L_{j''}$) for `fromLink` L_i.

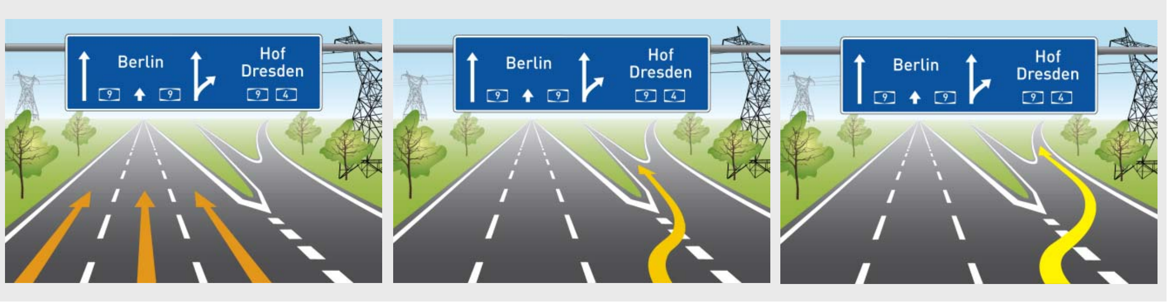
Figure 15-3 Junction with three possible `toLink` *relations (L_j , $L_{j'}$, and $L_{j''}$) for fromLink L_i*



The application generates the composed image required for the given route on the fly in the navigation system.

Figure 15-4 shows the three possible junction views based on the example in Figure 15-3.

Figure 15-4 Three possible images based on the possible fromLink and toLink combinations

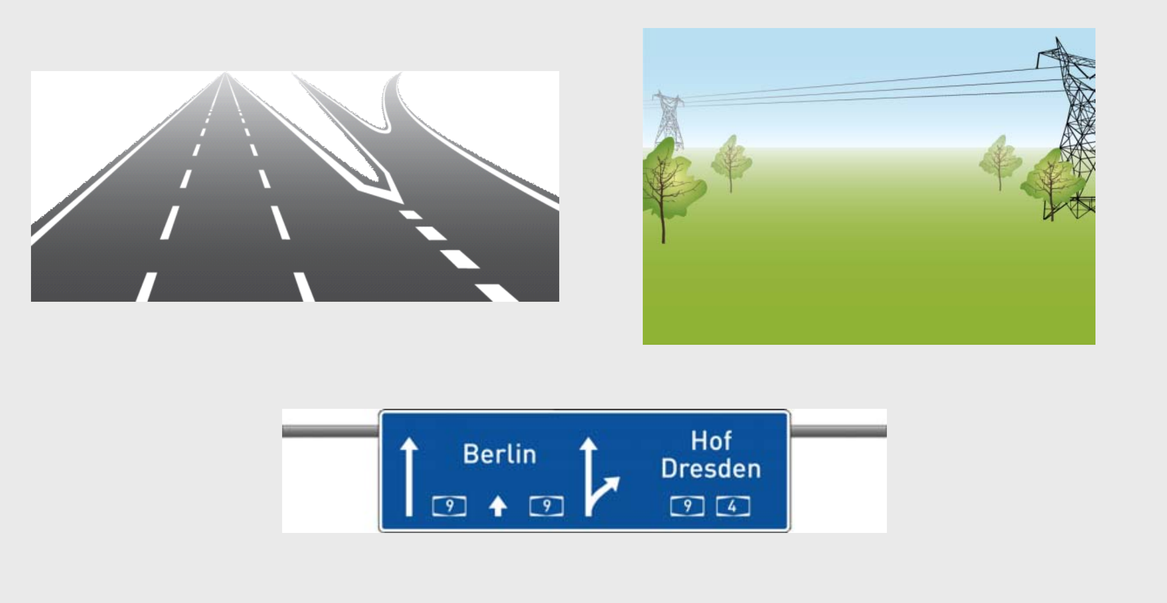


Drawing Order of the Image Components

Images assigned to the `fromLink` may be associated with a specific position on the link. This enables applications to show the images when the vehicle reaches a specific position on the link.

In addition, it is possible to overlay several images at the same position by using a sequence number. Overlaying of images may be applied when the image references stored in the `fromLink` table are stored as individual components, for example, a junction image, a background image, and a signboard image (see Figure 15-5). This enables the use of pattern images that may be referred to by different `fromLinks`. The arrow images may be reused in the same way by different combinations of `fromLink` and `toLink`. This reuse concept helps to reduce database size.

Figure 15-5 Image components, for example, junction image, background image, sign board



To create the junction view picture, each image component has a sequence number and the components are rendered in an ascending order given by the sequence number.

Note The images referenced in the `JvFromLink` table have to be drawn first, then the images referenced in the `JvFrom2ToLink` table.

16 Full-text Search Building Block

Note As modelling of full-text search is currently being optimized, the following chapter is only a draft version and may not represent the current discussions and/or modelling in DataScript. The description of the Full-text Search building block may, therefore, partly deviate from DataScript. Please note that in these cases the DataScript implementation shall prevail.

The Full-text Search (FTS) building block supports the search for features in an NDS database by entering strings instead of the predefined, hierarchical search by means of next-valid-character trees as used for location input (see *NDS – Format Specification*, 10.6 *Next-valid-character Trees* on page 194).

The full-text search supported by NDS bases on the SQLite full-text search module fts3. For more information on the use of fts3, refer to *http://www.sqlite.org/cvstrac/wiki?p=FtsUsage*.

For more information on the Full-text Search building block, refer to *NDS – Format Specification*, 19 *Full-text Search Building Block* on page 353.

16.1 Relation between Full-text Search Data and Name Data

The Full-text Search building block is optional, whereas the Name building block is not. An NDS database must therefore contain a Name building block, but does not need to contain a Full-text Search building block associated with this Name building block. Nevertheless, the application can use, for example, next-valid-character trees to realize a full- and fuzzy-text search module directly on the Name building block. Possibly, this is not as efficient as using an FTS building block.

16.2 Relation between Full-text Search Data and POI Data

The Full-text Search building block is optional and an NDS database may therefore contain a POI database, but no POI FTS database. Nevertheless, the application can depict and select POIs or use a full and fuzzy text search module directly on the POI database. Possibly, this is not as efficient as using the FTS database.

The NDS database may also contain a POI FTS database, but no POI database. In this case, the application can use the POI FTS database for searching POIs. However, if the application wants to retrieve more information on a selected POI, it has to install a POI database containing at least the selected POIs.

A POI FTS database and a POI database do not necessarily have to contain the same number of POIs: The application does not need an entry in the FTS database for each POI and vice versa. It is required, though, that POI IDs used in both databases represent the same real-world POIs.

16.3 SQLite fts3 Index

The following sections describe the structure of SQLite fts3 indices for the Full-text Search building block. The full-text search indices are created during compilation by SQLite fts3, meaning that the compiler calls fts3 during compilation. fts3 then creates the tables for the full-text search index.

Virtual Table `PoiFtsTable`

The POI FTS database contains exactly one virtual table `PoiFtsTable`. This virtual table is later used to access the indices containing the full-text search documents. The compiler creates the table by using the full-text search module fts3 with the following command (the field names give an example):

```
Create Virtual Table poiFts Using fts3
(PoiName, poiCategory, poiStreetName, poiHouseNumber, poiPostalCode, poiCityName,
poiPhone, searchTags);
```

With this command, the fts3 module creates the following three tables in the POI FTS database:

- `poiFTS_content`
- `poiFTS_segdir`
- `poiFTS_segments`

Note The `docId` must be equal to the `poiId`.

It must be ensured that data is only inserted into the virtual table and **not** into the three physical tables `poiFTS_content`, `poiFTS_segdir`, `poiFTS_segments`.

`poiFtsTable_content` Table

This table contains the content inserted into the virtual `PoiFtsTable`. The fts3 module adds the prefix `ci` to each column name of the virtual table where `i` denotes the column index. It also creates a hidden column `docId` which is used as a primary key for the FTS document. The fts3 module creates the `poiFTS_content` table with the following SQL command:

```
Create TABLE poiFTS_content
(docid INTEGER PRIMARY KEY,c0PoiName,c1 poiCategory,c2 poiStreetName,c3 poiHouseNumber,c4
poiPostalCode,c5poiCityName,c6poiPhone,c7searchTags);
```

`poiFtsTable_segdir` and `poiFtsTable_segments` Tables

These tables contain the inverted index structure, which is used to assign to each term a set of document IDs for documents containing the term. This n-to-m relation between terms and documents is stored in SQLite proprietary BLOB structures.

For more information, refer to *http://www.sqlite.org/cvstrac/fileview?f=sqlite/ext/fts3/fts3.c&v=1.25* .

The fts3 module splits the content of a POI document into terms by using either the SQLite default tokenizer (Simple or Porter), or a compiler-specific tokenizer. Which tokenizer is used during the creation of the indices is defined in the metadata for the fts3 module (see Section 16.5 *Metadata of the SQLite fts3 Module* on page 210). NDS does not standardize the tokenizer, but recommends to use a tokenizer which generates terms separated by blanks, (back-)slashes, semicolons, colons, and periods. Furthermore, the tokenizer should delete stop words like “and”, “the”, “a”, or other words which are part of the documents, but are not selective enough. These words remain in the `poiFtsTable_content` table, but are not required in the inverted index.

Note Terms are always stored in capital letters. At query time, the SQLite library changes the query terms to capital letters.

For more information on tokenizer, refer to *http://en.wikipedia.org/wiki/Tokenizer#Tokenizer*.

16.4 Optimization of the `poiFtsTable_content` Table

The following procedure may be used to optimize the `poiFtsTable_content` with regard to storage space.

Note This procedure does not comply with the specification of the fts3 module.

1. Insert all POI records into the virtual table `PoiFtsTable` as described in Section *Virtual Table `PoiFtsTable`* on page 208.
2. Update the `poiFtsTable_content` table with the following command: `UPDATE poiFTS_content SET c7searchTags=""`.
3. Apply the `VACUUM` command to the database.

This procedure at first inserts all POI terms into the `poiFtsTable_segdir` and `poiFtsTable_segments` table. The same information is also contained in the `poiFtsTable_content` table.

In a second step, the additional search strings are deleted from the `poiFtsTable_content` table. The search strings are still stored in the `poiFtsTable_segdir` and `poiFtsTable_segment` table. These search strings can now be regarded as hidden tags.

Note The same approach may be used for other columns.

If no information is found, the application has to access the core POI database to get this information. As it can be time-consuming to do this for all possible hits, the content removal should be done only when the performance decrease is outweighed by the data size reduction.

Furthermore, the `c0PoiName` column should not be set to NULL, as some use cases cannot efficiently be supported by the application in this case (for example NVC search).

The following rules apply:

- the `c7searchTags` shall always be set to `NULL`
- the `ci`…where 1≤i≤6 may be set to `NULL` (full-text search)
- the `c0PoiName` should not be set to `NULL` (NVC search)
- the `docId` shall not be set to `NULL`

Note If an FTS database is updated, for instance by inserting new POIs, you do not have to delete the content of any column of the content table after insertion. Carrying out step 2 without step 3 will not gain space. Step 3, however, might be too time-consuming on the target device and also needs additional secondary storage which might not be available.

16.5 Metadata of the SQLite fts3 Module

Metadata is required to allow for flexibility in the compiler and to ensure interoperability at the same time.

DataScript location: `FtsMetadataTable`

Table 16-1 Content of the `FtsMetadataTable`

Key	Value	Description
`buildingBlockId`	`integer`	Identifies the source data building block for the FTS virtual table in this instance of the FTS building block
`ftsTokenizer`	`SIMPLE|PORTER|USER`	Defines the tokenizer used in the compiler `SIMPLE` and `PORTER` are the tokenizers of the fts3 module. `USER` indicates that a user-defined tokenizer is applied. An application can use a different tokenizer than used in the compiler. This can, however, deteriorate the results.
`searchTagsDescription`	String	A verbal description of all included search strings, for instance: street base names, exonyms for city names, parent category names in English, “known-as” areas, URI

Table 16-2 Content of the `FtsColumnMetadataTable`

Key	Value	Description
`columnName`	`string`	Name of the column from the FTS virtual table in this instance of the Full-text Search building block. The actual column name can be obtained from the SQLite schema of the FTS virtual table.
`dataAvailability`	`NONE|INDEX_ONLY|ALL`	Indicates availability of the data for the column `columnName` of the FTS virtual table. Columns: – may be empty (`NONE`) – may have an index only (`INDEX_ONLY`) – may have an index and content (`ALL`)

17 Using Flexible Attributes and Attribute Groups

The Routing building block of NDS provides many flexible attributes to ensure proper support of routing and guidance functions as well as to model functions for map-based driver assistance systems. Examples are attributes describing route options (`TOURIST_ROUTE_TYPE`, `FERRY_TYPE` or `TYPE_OF_PAVEMENT`) and attributes used to calculate toll costs of a selected route (`TOLL`, `TOLL_VIGNETTE`, `TOLL_CHARGE`, `CURRENCY`).

Attributes can be grouped together flexibly and the resulting attribute group can be assigned to one or more features. For detailed information on the attribute mechanisms provided by NDS, refer to *NDS – Format Specification*, 6 *Attributes* on page 89.

Due to the high number of flexible attributes and the different requirements of the suppliers compiling NDS-compatible products, many different combinations of attributes are possible. This chapter collects recommendations for using and grouping attributes in order to model a specific real-world situation, to calculate specific routing options, or to give special route guidance. The information is structured based on the use cases to be covered.

Note This chapter is a work in progress and not yet complete. For detailed information on the data types and allowed values of the attributes mentioned in this appendix, refer to the *NDS – Physical Model Description*.

17.1 Example of Grouping Flexible Attributes (Routing)

This example lists the attribute groups to be formed and assigned in order to model the following real-world situation:

A small village that lives on tourism and lies on a shortcut route that is very attractive to truck drivers wants to prohibit excessive truck traffic and thus installs the following regulation for its main road:

Prohibited for trucks with a weight of more than 8 tons or a width of more that 3m or a height of more than 4m from Monday up to and including Friday; and prohibited for motorcycles and for vehicles with more than 2.8 tons or a length of more than 8m on Saturdays and Sundays. This regulation does not hold for busses.

To model this situation, the complex regulation needs to be split up into individual passage restrictions, which can be modeled by separate attribute groups. By assigning all these groups to the link feature representing the village‘s main road, a logical OR relation is created between the passage restrictions. Attribute values that are used in more than one group may be stored only once and used repeatedly, as the group number is part of the reference to the attribute group and not part of the value.

The following attribute groups represent the passage restrictions:

1. *Prohibited for TRUCKS with a WEIGHT of more than 8 tons from MONDAY till inclusively FRIDAY*
   modeled by
   `PROHIBITED_PASSAGE` grouped with `FREQUENTLY_USED_VEHICLE_TYPES` (value = `isTRUCK`), `WEIGHT_METRIC` (value 80 = 8tons) and `DAYS_OF_WEEK` (bits set for Monday...Friday)
2. *Prohibited for TRUCKS with a WIDTH of more that 3m from MONDAY up to and including FRIDAY*
   modeled by
   `PROHIBITED_PASSAGE` grouped with `FREQUENTLY_USED_VEHICLE_TYPES` (value = `isTRUCK`), `VEHICLE_WIDTH_METRIC` (value 120 = 3m) and `DAYS_OF_WEEK` (bits set for Monday...Friday)
3. *Prohibited for TRUCKS with a HEIGHT of more than 4m from MONDAY till inclusively FRIDAY*
   modeled by
   `PROHIBITED_PASSAGE` grouped with `FREQUENTLY_USED_VEHICLE_TYPES` (value = `isTRUCK`), `VEHICLE_HEIGHT_METRIC` (value 160 = 4m) and `DAYS_OF_WEEK` (bits set for Monday...Friday)
4. *Prohibited for MOTORCYCLES on SATURDAYS and SUNDAYS*
   modeled by
   `PROHIBITED_PASSAGE` grouped with `FREQUENTLY_USED_VEHICLE_TYPES` (value = `isMOTORCYCLE`), `DAYS_OF_WEEK` (bits set for Saturday...Sunday)
5. *Prohibited for vehicles with a WEIGHT of more than 2.8 tons on SATURDAYS and SUNDAYS; this regulation does not hold for BUSSES.*
   modeled by
   `PROHIBITED_PASSAGE` grouped with `FREQUENTLY_USED_VEHICLE_TYPES` (value = `isBUS` and the `isInclusive` flag set to `FALSE`; that means except busses), `WEIGHT_METRIC` (value 28 = 2.8 tons) and `DAYS_OF_WEEK` (bits set for Saturday...Sunday)
6. *Prohibited for vehicles with a LENGTH of more than 8m on SATURDAYS and SUNDAYS; this regulation does not hold for BUSSES.*
   modeled by
   `PROHIBITED_PASSAGE` grouped with `FREQUENTLY_USED_VEHICLE_TYPES` (value = `isBUS` and the `isInclusive` flag set to `FALSE`; that means except busses), `LENGTH_METRIC` (value 320 = 8m) and `DAYS_OF_WEEK` (Bits set for Saturday...Sunday)

17.2 Using Flexible Attributes for Lanes

Roads are digitized as a single polyline for each carriageway independent of the number of lanes. To achieve a high quality for route guidance, it is not sufficient to encode the number of lanes. The following information is also important:

- Road markings such as arrows and names of destinations on lanes or signposts
- Possibilities to change lanes
- Connectivity of lanes across intersections or along links with a changing number of lanes

If necessary, lane properties (numbering of lanes, marking, connectivity, etc.) along links and across intersections can be specified by attribute groups mainly using validity ranges.

Note For the number of lanes definition, the following rules apply:

- The 15th lane in the `laneMask` corresponds to "15th or more" lanes.
- The value "15" for `normalLanes` corresponds to "15 or more" normal lanes.
- The value "15" for `LaneConnectivityAlongLink.laneNumber` refers to all lanes ≥15.
- The value "15" for `LaneTypeAndNumber.laneNumber` corresponds to ≥15.
- `LaneSeparators`: For all lanes > 15, the same status for open to right and left is assumed as for lane 15.
- `NumConnectedLanes`: As a maximum of 16 lanes can be defined, the value "16" means that 16 or more lanes are connected. Although a maximum of only 15 lanes is possible per link, more than 15 lane elements (connected lanes) can exist, because more than one link can be connected to one lane on a link.

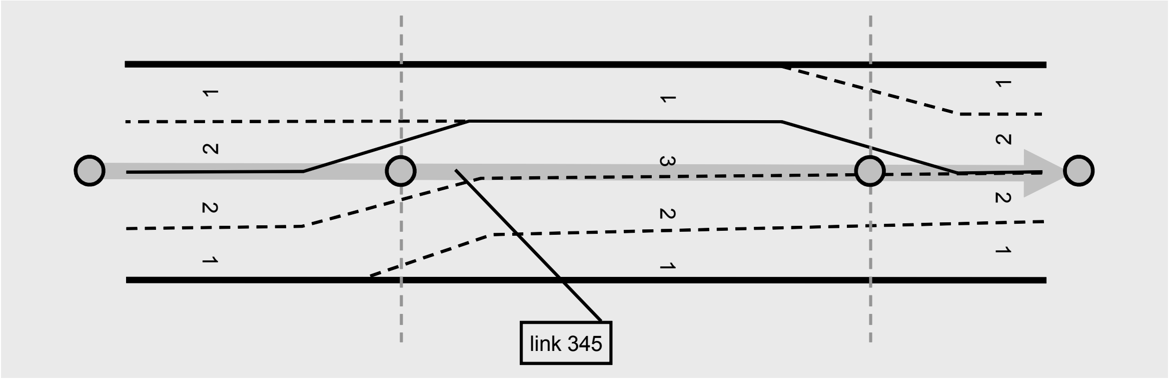
The sections below describe the different lane attributes and give examples.

Numbering of Lanes

Lanes are always numbered separately for each driving direction from the curb to the middle of the road. Right hand traffic lanes are therefore numbered from the right to the left relating to the driving direction. Left hand traffic lanes are numbered from left to right relating to the driving direction.

Note If a lane starts or ends at the curb, the numbers of the remaining lanes change as shown in Figure 17-1. Exit lanes are an exception to this numbering scheme (see Section *Exit Lanes* on page 223).

Figure 17-1 Numbering of lanes



For the link 345 in Figure 17-1 in positive direction (from left to right) the number of lanes is changing from 2 to 3 and back to 2. This is encoded by three attribute groups:

Table 17-1 Example of attribute groups for numbering of lanes

Attribute	Value
First attribute group	
NUM_LANES	2 normal, no exit lanes
VALIDITY_RANGE	validityRangeProperty=CONTAINS_START_AND_END LinePosition 0 LinePosition 1
Second attribute group	
NUM_LANES	3 normal, no exit lanes
VALIDITY_RANGE	validityRangeProperty=CONTAINS_START_AND_END LinePosition 1 LinePosition 2
Third attribute group	
NUM_LANES	2 normal, no exit lanes
VALIDITY_RANGE	validityRangeProperty=CONTAINS_START_AND_END LinePosition 2 LinePosition 3

Lane Markings

Lane markings are often arrow symbols indicating the allowed maneuvers per lane at the next intersection (see Figure 17-2). Lane markings are therefore useful for route guidance.

Figure 17-2 Lane markings, example

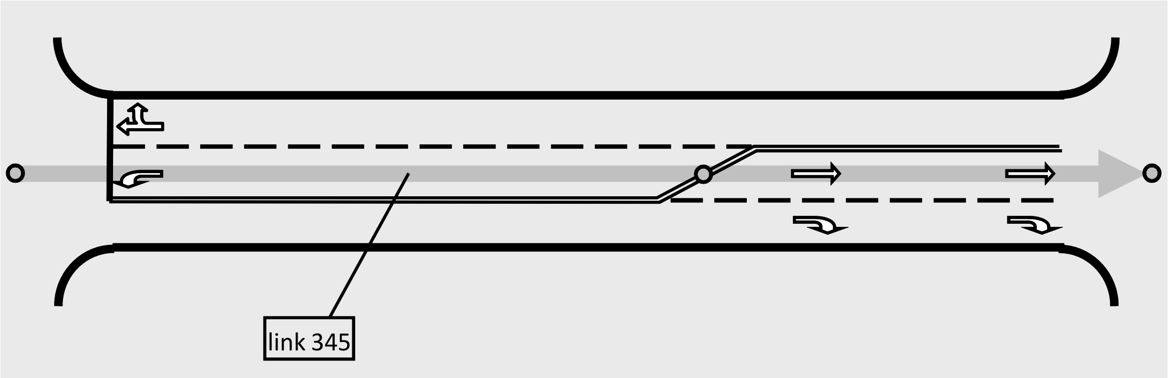


Table 17-2 shows how attribute groups are used to specify lane markings per lane.

Table 17-2 Example of attribute groups for lane markings

Attribute group	Value	Result

`NUM_LANES`
`VALIDITY_RANGE`
	2,0,0 (2 normal, no exit lanes)
	`validityRangeProperty=`
	`CONTAINS_START_AND_END`
	`LinePosition 1`
	`LinePosition 2`
		Defines two lanes from the first shape point to the end node

`LANE_MARKING`
`LANE_MASK`
`VALIDITY_RANGE`
	`isRightTwo8`
	`isFirstLane`
	`validityRangeProperty=`
	`CONTAINS_START_AND_END`
	`LinePosition 1`
	`LinePosition 2`
		Defines a right arrow for the first lane

`LANE_MARKING`
`LANE_MASK`
`VALIDITY_RANGE`
	`isAheadTwo8`
	`isSecondLane`
	`validityRangeProperty=`
	`CONTAINS_START_AND_END`
	`LinePosition 1`
	`LinePosition 2`
		Defines a straight arrow for the second lane

Information If lanes exist, but are not marked on the road, flexible attributes of type `LANE_MARKING` set to `FALSE` shall be assigned to the lanes. If there is no information available whether lane markers exist, no `LANE_MARKING` attribute shall be assigned to the lane.

Lane Dividers

Lanes can be separated by physical barriers, such as poles or grass verges. To model physical lane dividers, NDS provides the flexible attribute `PHYSICAL_LANE_DIVIDER`, which can be assigned to intersections, transitions, links or lane masks.

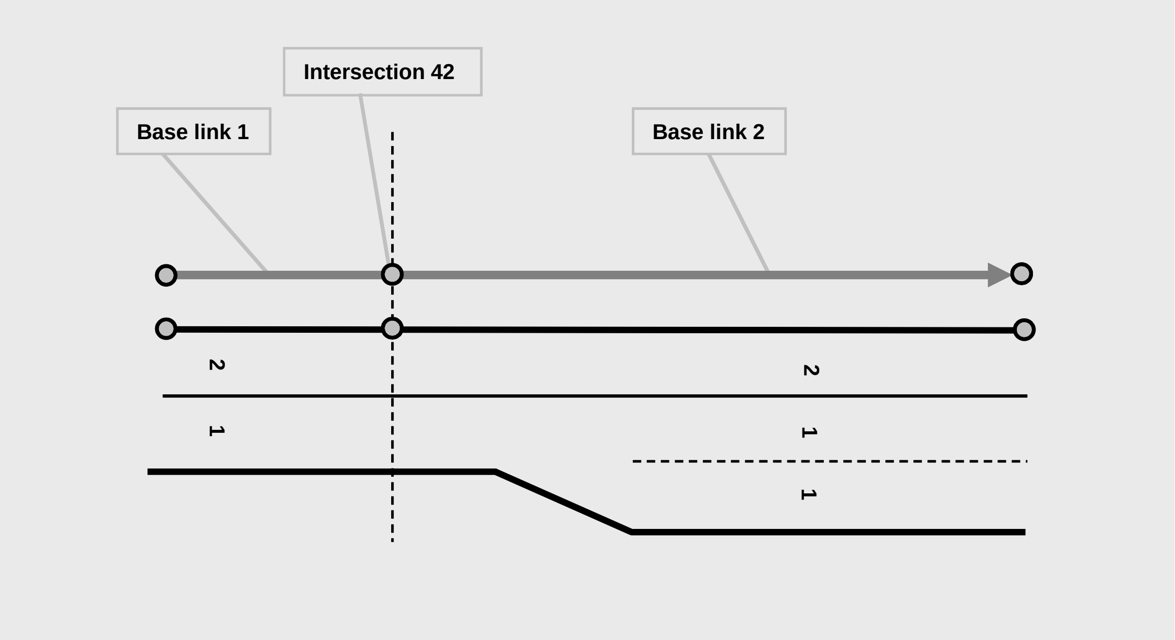
Note	This attribute only needs to be assigned if there is a physical divider or if it is not known whether a divider exists. If the lanes are not separated by dividers, the attribute does not need to be instantiated.

Connectivity of Lanes Across Intersections: Default Case without Lane Connectivity Attribute

If a base link or a road geometry line has a certain number of lane atributes, and is connected with only one other single base link or road geometry line, which has exactly the same number of lanes attribute, the lane connectivity does not need to be stored. The reason is that the topology is implicitly defined. Such situations happen typically at tile borders, or where a road is split due to the change of another attribute.

Figure 17-3 shows a case, where the lane connectivity from base link 1 via intersection 42 to base link 2 is uniquely defined. Thus, the attribute `LANE_CONNECTIVITY_ACROSS_INTERSECTION` is not required.

Figure 17-3 Default case without lane connectivity attribute



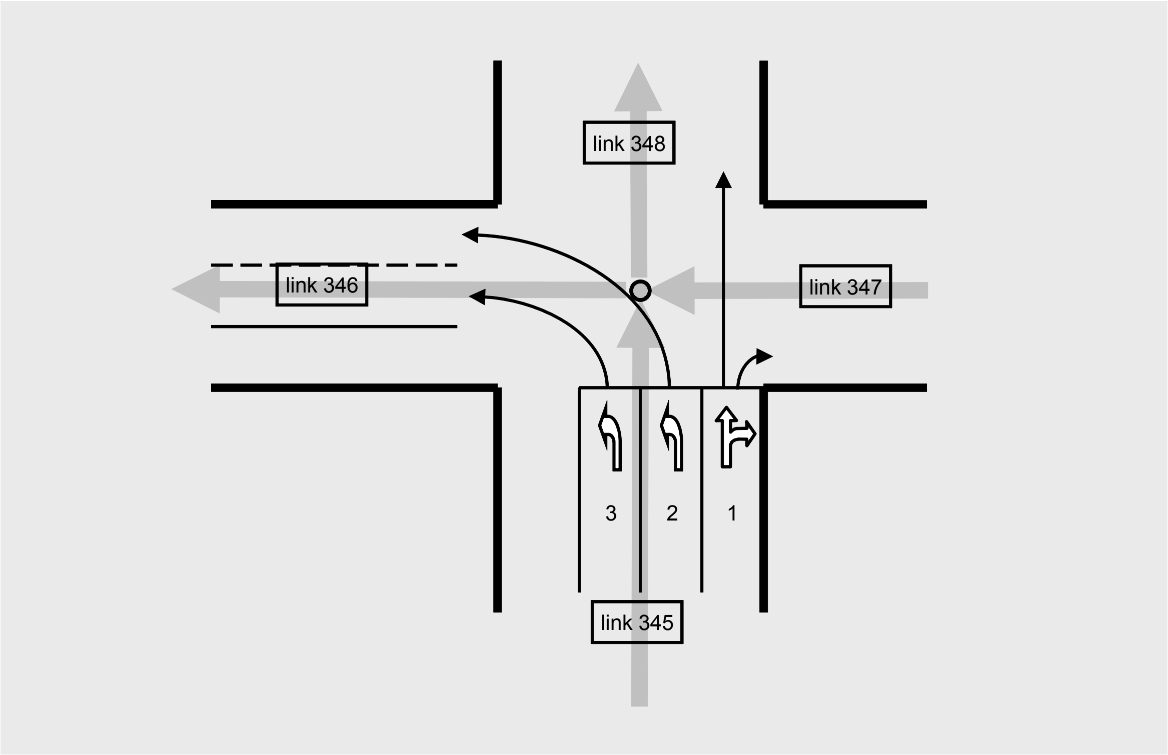
The following two subsections give examples, where attributes for lane connectivity are required.

Connectivity of Lanes Across Intersections: with Lane Connectivity Attribute

Figure 17-4 shows a simple intersection. The link 345 has three lanes in forward direction. Two of them are marked as a turn-left lanes and the third is marked as turn-right or straight ahead lane. Thus, the following references from lanes of link 345 to link 346, 347, 348 are necessary:

- Lane 1 of link 345 to link –347 (direction end-to-start, 1 lane) and link 348 (no lane information available in the database for this link)
- Lane 2 of link 345 to lane 1 of link 346
- Lane 3 of link 345 to lane 2 of link 346

Figure 17-4 Connectivity of lanes across intersections



Three `LANE_CONNECTIVITY_ACROSS_INTERSECTION` attributes are assigned to the intersection.

Table 17-3 Sample settings for `LANE_CONNECTIVITY_ACROSS_INTERSECTION` *attribute*

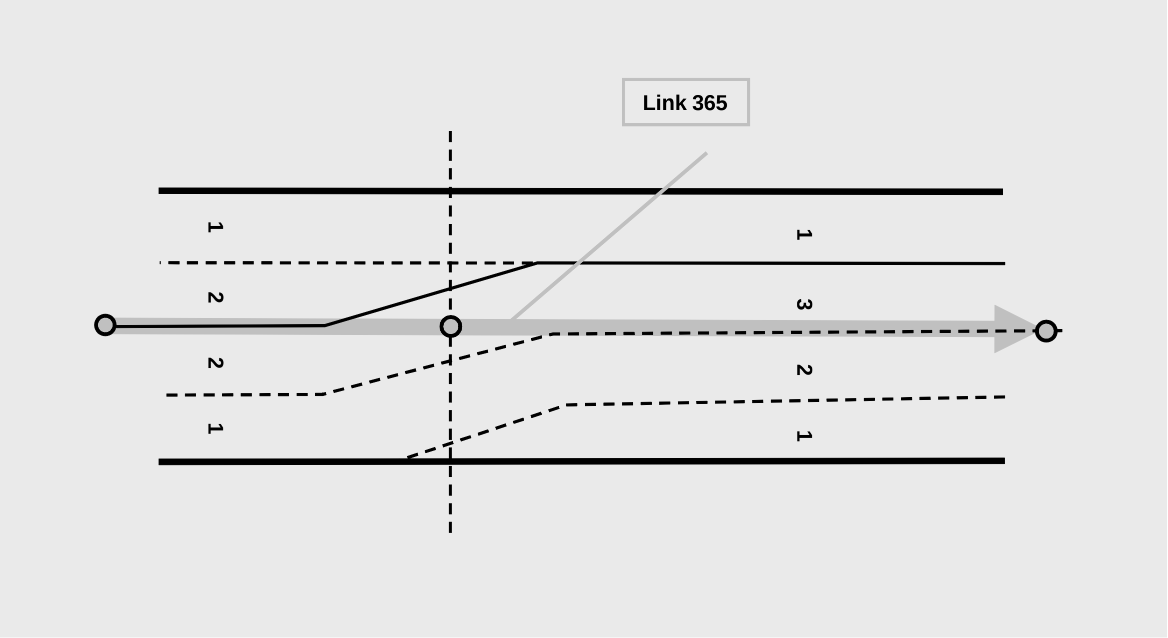
Setting	Value	Comment
First `LANE_CONNECTIVITY_ACROSS_INTERSECTION` **attribute**		
`SourceLaneSpecifier`	normal lane 1	Lane 1 is connected
`NumConnectedLanes`	2	To the two lanes specified below

Setting	Value	Comment
`destLanes`	normal lane 1, base link –34 7 normal lane 1, base link 348	Lane 1 on link –347 (has only one lane) lane type = `NO_LANE_INFORMATION_AVAILABLE`, no lane number is given on link 348 (has no lane information)
Second `LANE_CONNECTIVITY_ACROSS_INTERSECTION` **attribute**		
`SourceLaneSpecifier`	normal lane 2	
`NumConnectedLanes`	1	
`DestLanes`	normal lane 1, base link 346	
Third `LANE_CONNECTIVITY_ACROSS_INTERSECTION` **attribute**		
`SourceLaneSpecifier`	normal lane 3	
`NumConnectedLanes`	1	
`DestLanes`	normal lane 2, link 346	

Connectivity of Lanes Along Links

The connectivity of lanes along a link can be encoded in a similar way as for lanes across intersections. Figure 17-5 shows an example of a link with a changing number of lanes. The number of lanes is changing from two to three in forward direction because a new lane begins at the first shape point. According to the lane numbering, lane 1 (in front of the shape) is connected to lane 2 (beyond the shape) and lane 2 is connected to lane 3.

Figure 17-5 Connectivity of lanes along links



Two attributes of the attribute type `LaneConnectivityAlongLink` define the connectivity. The attributes defining the number of lanes are assumed.

Attribute groups defining the number of lanes along link +365:

- `NUM_LANES`: 2,0,0 (2 normal, no exit lanes)

  ```
  VALIDITY_RANGE: validityRangeProperty=CONTAINS_START_AND_END
  LinePosition 0
  LinePosition 1
  ```
- `NUM_LANES`: 3,0,0 (3 normal, no exit lanes)

  ```
  VALIDITY_RANGE: validityRangeProperty=CONTAINS_START_AND_END
  LinePosition 1
  LinePosition 2
  ```

Table 17-4 shows the attribute groups defining the connectivity between the lanes.

Table 17-4 Example of attribute groups for connectivity between lanes

Attribute	Value	Comment
First attribute group		
`LANE_CONNECTIVITY_ALONG_LINK`	normal lane, 1	Lane 1 is connected
`LANE_MASK`	`isSecondLane`	To the second lane

`VALIDITY_RANGE`	`validityRangeProperty=`
	`CONTAINS_START`
	`LinePosition 1`
		At the first shape point

Second attribute group		
`LANE_CONNECTIVITY_ALONG_LINK`	normal lane, 2	Comment: Lane 2 is connected
`LANE_MASK`	`isThirdLane`	To the third lane

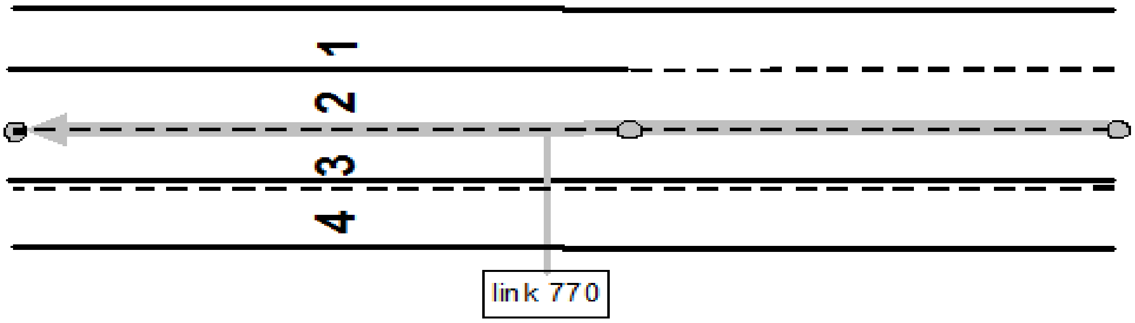
`VALIDITY_RANGE`	`validityRangeProperty=`
	`CONTAINS_START`
	`LinePosition 1`
		At the first shape point

Lane Separator

The attribute `LANE_SEPARATORS` indicates whether a change of lane is allowed or not. Lane changes can be restricted to one direction or allowed for both directions. Figure 17-6 shows an example of applying the attribute. The figure shows a carriageway with traffic flow in one direction.

Note For bidirectional traffic flow combined dashed and solid lines must be interpreted differently for the both directions.

Figure 17-6 Lane separators



Based on the example given in Figure 17-6, flags in the `LaneSeparator` attribute define the allowed lane changes, see Table 17-5.

Table 17-5 Example of attribute groups for lane

Attribute	Property/flag	Comment
First attribute group		
`LANE_SEPARATORS`	`LaneOpen[0].openToTheRight = FALSE`	
	`LaneOpen[0].openToTheLeft = TRUE`	Lane change to second lane allowed
	`LaneOpen[1].openToTheRight = TRUE`	Lane change to first lane allowed
	`LaneOpen[1].openToTheLeft = TRUE`	Lane change to third lane allowed
	`LaneOpen[2].openToTheRight = TRUE`	Lane change to second lane allowed
	`LaneOpen[2].openToTheLeft = FALSE`	
	`LaneOpen[3].openToTheRight = TRUE`	Lane change to third lane allowed
	`LaneOpen[3].openToTheLeft = FALSE`	

`VALIDITY_RANGE`	`validityRangeProperty=`
	`CONTAINS_START_AND_END`
	`LinePosition 0`
	`LinePosition 1`
		Valid for the range from start node to first shape point

Second attribute group		
`LANE_SEPARATORS`	`LaneOpen[0].openToTheRight = FALSE`	
	`LaneOpen[0].openToTheLeft = FALSE`	
	`LaneOpen[1].openToTheRight = FALSE`	
	`LaneOpen[1].openToTheLeft = TRUE`	Lane change to third lane allowed
	`LaneOpen[2].openToTheRight = TRUE`	Lane change to second lane allowed
	`LaneOpen[2].openToTheLeft = FALSE`	
	`LaneOpen[3].openToTheRight = TRUE`	Lane change to third lane allowed

Attribute	Property/flag	Comment
	`LaneOpen[3].openToTheLeft = FALSE`	

`VALIDITY_RANGE`	`validityRangeProperty=`
	`CONTAINS_START_AND_END`
	`LinePosition 1`
	`LinePosition 2`	

17.2.1 Special Lanes

Lanes can have special functions, for example

- Freeway entrances or exits
- Passing in both directions
- Turns and merging into the traffic flow on wide roads in both directions

Lanes that are reserved for busses, taxis, etc. can be described by an additionally grouped vehicle type attribute as used for other restrictions.

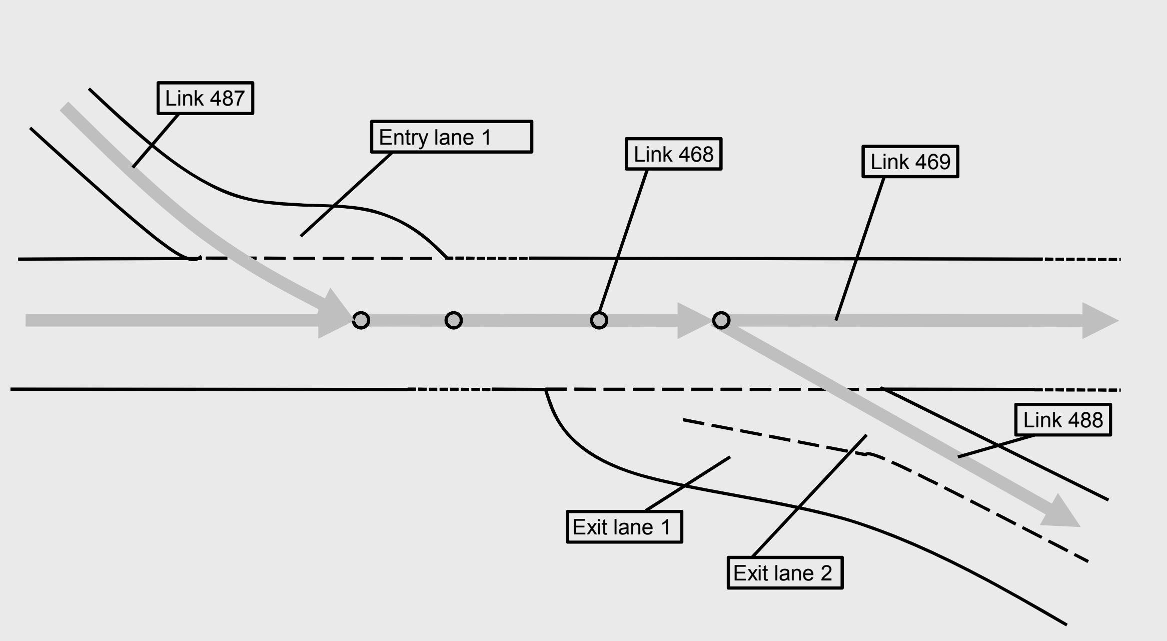
Exit Lanes

Lanes of freeway entrances and exits play a special role. They are not counted with the lanes of the carriageway, but separately for the left and the right side. A link can therefore have normal lanes, entry/exit lanes to the left and entry/exit lanes to the right (see Figure 17-7). Entry/exit lanes are numbered in the same sequence from curb to center as are normal lanes. So the outermost lane is exit lane 1, the next lane is exit lane 2, and so on.

Figure 17-7 shows a freeway entry and a freeway exit. Link 468 on the freeway includes an entry lane on the left. The length of the entry lane is defined by means of a validity range starting at the link's start node and ending at the first shape point. The entry is described by means of a first attribute group for link 468 containing the validity range and the number of lanes (two plus one exit (= entry) lane). For the next validity range from the first to the second shape point, link 468 has two normal lanes. The third validity range from the second shape point to the end node has two normal lanes and two exit lanes to the right.

Link 469 also has two exit lanes for the first 80 m, which is described in attribute group 1 for link 469. In attribute group 2 for link 469, the number of lanes is reduced to two normal lanes.

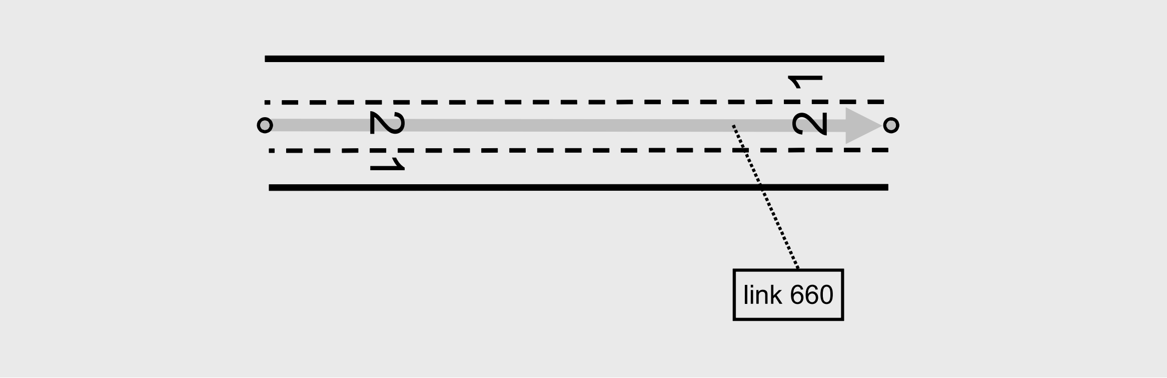
Figure 17-7 Entry and exit lanes at a freeway



Passing Lanes

Lanes in the middle of a road can be allowed for passing in both directions (see Figure 17-8). The value 2 is specified for the attribute NUM_LANES (numbering of lanes) for both directions and grouped with an additional attribute LANE_PASSING. This attribute indicates that the middle lane can be used in both directions.

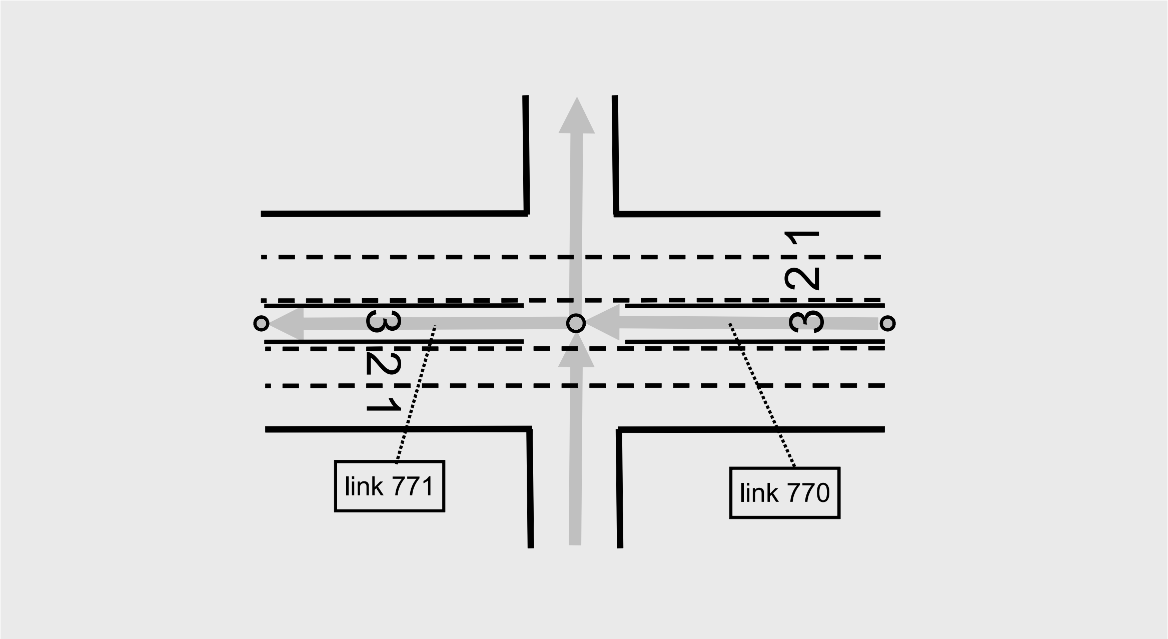
Figure 17-8 Passing lane



Turning Lanes

Turning lanes are used to facilitate left turns (at right hand traffic). The car can turn into a middle lane without giving the right of way for traffic from the right going straight ahead. From the middle lane, the car can merge to the right lane(s). This is described in the same way as passing lanes, but with the special attribute LANE_TURNING. Figure 17-9 gives an example of turning lanes.

Figure 17-9 Turning lane



17.3 Modelling Express Roads and Express Lanes

An express lane is a traffic lane, or, in case of an express road, a set of lanes with a limited number of entrance and exit points. Express lanes or roads are physically separated from the normal lanes or roads.

The information whether a lane is an express lane, or if a road is an express road, is relevant for Routing and for Route Guidance. For route calculation, express lanes must be identified to either avoid or prefer them when calculating a route. If a road has express and normal lanes it can be used in both cases.

The flexible attribute `IS_COMPLETE_EXPRESS_ROAD` belongs to the Routing layer and can be assigned to directed and undirected base and route links on all levels. If the attribute value is `TRUE`, the link is an express road, meaning that it only consists of express lanes and the complete road is physically separated from the normal road. If the attribute value is FALSE, the road represented by the respective link has express lanes and normal lanes. If the attribute is not assigned to a link, no express lanes exist.

In the Route Guidance layer, the flexible attribute `EXPRESS_LANE` can be grouped

with the `LANE_MASK` and `EXIT_LANE_MASK` attributes to indicate that a lane is an express lane. The `EXPRESS_LANE` attribute is only assigned to directed base links and road geometry lines on level 13.

17.4 Using Flexible Attributes for Signposts

Signpost information along roads is attached to links and road geometry lines on level 13 of the road network to store the information where the sign is positioned in reality. As signposts are used for route guidance, they are only available on level 13.

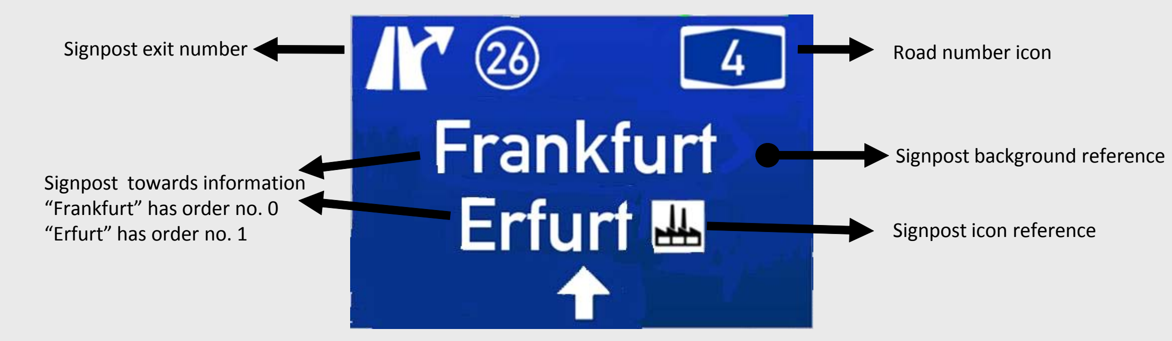
NDS provides the following flexible attributes for signposts:

- `SIGNPOST_LINK_REFERENCE` refers to the first base or route link beyond real-world intersections, which belongs to the indicated direction or route to destination
- `VALIDITY_RANGE` indicates the location of the signpost along the base link or road geometry line
- `SIGNPOST_TOWARD_INFO_REFERENCE` refers to the name of a destination indicated on a signpost for one direction. An attribute group with the signpost information may contain one or more towards references. To put the towards information on the signpost in a specific order, `signpostInfoOrder` can be used.
- `SIGNPOST_DIRECTIONAL_INFO_REFERENCE` refers to a road number leading to a destination indicated on a signpost for one direction
- `SIGNPOST_ICON_REFERENCE` refers to an icon representing a destination indicated on the signpost for one direction
- `SIGNPOST_EXIT_NUMBER` refers to an exit number leading to a motorway exit indicated on a signpost
- `SIGNPOST_LINK_REFERENCE` refers to a link that must be passed in order to reach a destination displayed on the signpost
- `SIGNPOST_BACKGROUND_REFERENCE` refers to an icon that forms the signpost's background
- `SIGNPOST_SPEECH_REFERENCE` is used for speech output for signpost names. The assigned name features are referenced by phonetic transcriptions stored in the Speech building block (see *NDS – Format Specification*, 14 *Speech Building Block* on page 289).

The signpost information components may be grouped to a complex attribute. There can be a set of components for each direction indicated on the signpost.

Figure 17-10 gives an overview of signpost elements.

Figure 17-10 Signpost Elements



Example: Traffic Junction with Signposts

Figure 17-11 shows an example of a traffic junction with signposts. The signpost attached to link 3825 contains the following information:

- Exit number C66 to the right leads to A-City and B-City
- A-City and B-City can be reached via road I4
- C-City can be reached by going straight ahead without using the exit

From the routing data, the application can use the following information:

- A-City can be reached on I4 East via link 3922
- B-City can be reached on I4 West via link 3924
- C-City can be reached via link 3826

The attributes describing the information on the signpost are assigned to link 3825.

Figure 17-11 Traffic node with signposts

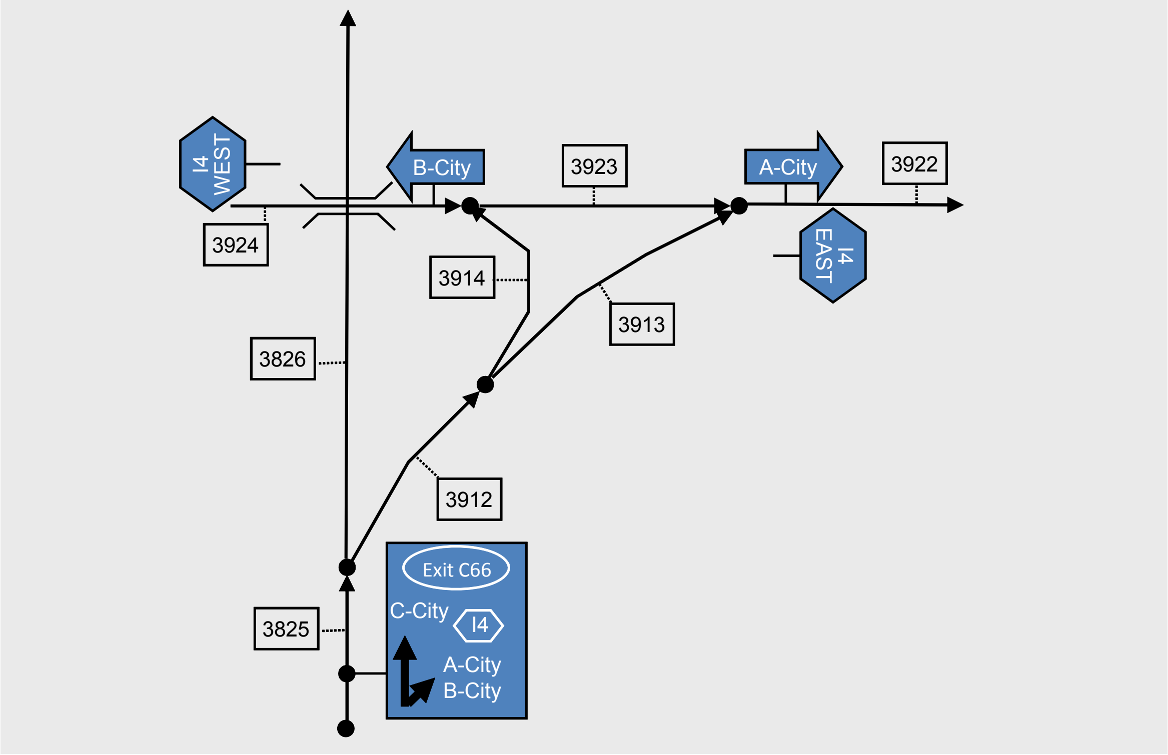


Table 17-6 and Table 17-7 show the corresponding group of attributes attached to link 3825.

Table 17-6 Group of attributes attached to link 3825 (direction to A-City and I4 East)

Attribute type	Value	Description
SIGNPOST_LINK_REFERENCE	DirectedLinkReference = +3922	References the name string to the base or route link 3922. Based on this information, route guidance can provide the advice "Turn right to A-City on I4-East" for a route that includes the links 3826 and 3922.
SIGNPOST_TOWARD_INFO_REFERENCE		Specifies a reference to a signpost name as towards information (A-City in the example) For more information, refer to Section 7.6.4 *Assigning Named Objects to Routing and Basic Map Display Tiles and Features* on page 146
SIGNPOST_DIRECTIONAL_INFO_REFERENCE		Specifies a reference to a signpost name as directional information (I4-East in the example) For more information, refer to Section 7.6.4 *Assigning Named Objects to Routing and Basic Map Display Tiles and Features* on page 146
SIGNPOST_EXIT_NUMBER		Specifies an alphanumerical string identifying an exit number, for example, C66
VALIDITY_RANGE	validityRangeProperty = CONTAINS_START_AND_END LinePosition 1 LinePosition 1	Indicates the signpost location; in the example this is the first shape point of link 3825

Table 17-7 Group of attributes attached to link 3825 (direction to B-City and I4 West)

Attribute type	Value	Description
SIGNPOST_LINK_REFERENCE	DirectedLinkReference = –3924	References the name to the base or route link 3924 Based on this information route guidance can provide the advice "Turn right to B-City on I4-West" for a route that includes the links 3826 and –3924.

Attribute type	Value	Description
SIGNPOST_TOWARD_INFO_REFERENCE		Specifies a reference to a signpost name as towards information ("B-City" in the example). For more information, refer to Section 7.6.4 *Assigning Named Objects to Routing and Basic Map Display Tiles and Features* on page 146
SIGNPOST_DIRECTIONAL_INFO_REFERENCE		Specifies a reference to a signpost name as directional information ("I4-West" in the example). For more information, refer to Section 7.6.4 *Assigning Named Objects to Routing and Basic Map Display Tiles and Features* on page 146
SIGNPOST_EXIT_NUMBER		Specifies an alphanumerical string identifying an exit number, for example, C66
VALIDITY_RANGE	validityRangeProperty = CONTAINS_START_AND_END LinePosition 1 LinePosition 1	Indicates the signpost location (first shape point of link 3825)

For the direction to C-City via the link 3826, a third group of attributes is attached to link 3825.

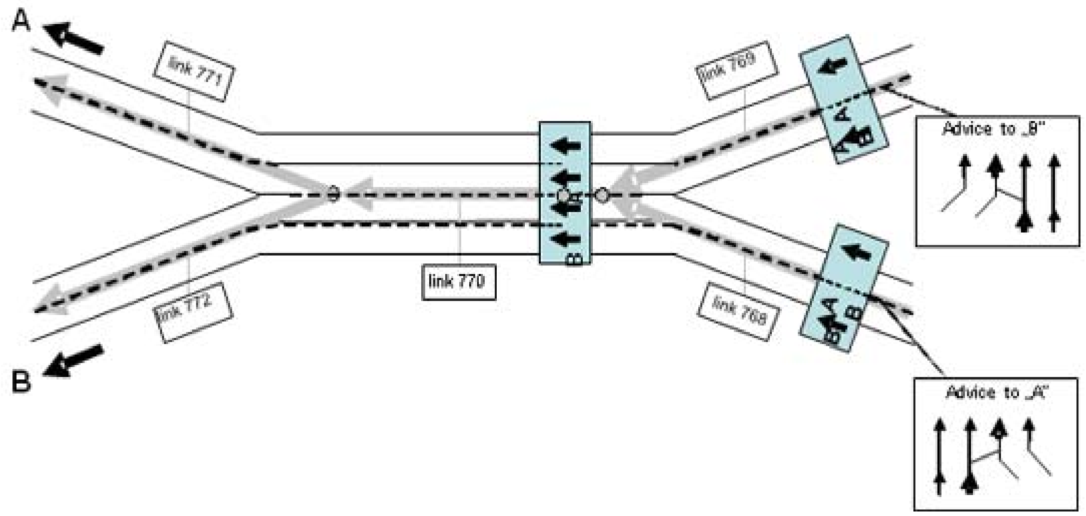
Table 17-8 Group of attributes attached to link 3825 (direction to C-City via link 3826)

Attribute type	Value	Description
SIGNPOST_LINK_REFERENCE	DirectedLinkReference = 3826	References the name to the base or route link 3826; based on this information, route guidance can provide the advice "follow the road to C-City" for a route that includes the link 3826
SIGNPOST_TOWARD_INFO_REFERENCE		Specifies a reference to a signpost name as toward information ("C-City" in the example) For more information, refer to Section 7.6.4 *Assigning Named Objects to Routing and Basic Map Display Tiles and Features* on page 146
VALIDITY_RANGE	validityRangeProperty = CONTAINS_START_AND_END LinePosition 1 LinePosition 1	Indicates the signpost location (first shape point of link 3825)

Lanes and Signposts

Signposts can contain information that is relevant for different lanes separately (see Figure 17-12). Attribute groups are used to describe the signpost information and to assign it to a specific lane.

Figure 17-12 Lane separators and signposts



One signpost is located at the first shape point of link 770. It contains towards information for link 771 and link 772. The signpost has the following attributes:

- Towards information for link 771:
  - `SIGNPOST_LINK_REFERENCE` `directedLinkReference` = +771 grouped with `SIGNPOST_DIRECTIONAL_INFO_REFERENCE`.
  - Lane masks specify the lanes: `LANE_MASK`, `isFirstLane`, `isSecondLane`
  - The `VALIDITY_RANGE` attribute defines the position:
    `validityRangeProperty = CONTAINS_START`
    `LinePosition 1`
- Towards information for link 772:
  - `SIGNPOST_LINK_REFERENCE` `directed_link_reference` = 772 grouped with `SIGNPOST_DIRECTIONAL_INFO_REFERENCE`
  - `signpostNameTblIndex` = 2 (B)
  - `LANE_MASK` `isThirdLane`, `isFourthLane`
  - `VALIDITY_RANGE`: `validityRangeProperty = CONTAINS_START`
    `LinePosition 1`

17.5 Modelling Toll Systems

Toll systems are used in various countries, and a variety of different toll systems, payment methods, and payment conditions exists.

Note	The present NDS version does not yet support all toll systems. Some toll systems, such as electronic toll systems, will be integrated in a later version.

The general information whether toll must be paid for a road is stored by means of the fixed attribute `toll` (DataScript location: `nds.common.fixedattributes` > `SharedRoadAttributes`). Flexible attributes may be used additionally to provide more detailed toll information, such as method and provider for toll payments. Examples would be paying toll per credit card or cash. In order to store detailed toll information, the fixed attribute `toll` must be set to classify the link as toll road.

Table 17-9 Data structures for toll information

Data	Type	Description
Data for modelling length-based toll costs		
`TollCostTable`	SQL table	Stores the information required to define the toll costs between the entry and exit toll gate
`TOLL_GATEID`	Flexible attribute	Defines a unique ID for a toll gate for adding toll costs
Data for avoidance of toll and map display		
`toll` (direction)	Fixed attribute	Defines if a link is part of a toll road or has a toll booth and the direction for which toll has to be paid
`TOLL`	Flexible attribute	Describes that a route link on upper level is partly a toll road
Data for modelling toll booth and gate indication		
`NUM_TOLL_BOOTHS`	Flexible attribute	Indicates that a toll gate is available and its position on level 13; gives the number of toll booths
`TOLL_BOOTH_MASK`	Flexible attribute	Provides the possibility to define attributes for one or more toll booths, for example payment type
`NUM_TOLL_GATES`	Flexible attribute	Indicates the number of toll gates on an aggregated upper-level link; used for route cost calculation
Data for modelling toll entry and toll exit		
`TOLL_ENTRY`	Flexible attribute	Indicates the entry to a closed toll system
`TOLL_EXIT`	Flexible attribute	Indicates the exit from a closed toll system
Data for modelling fixed fee for a whole country (vignette)		
`TOLL_VIGNETTE`	Flexible attribute	Indicates that a vignette has to be bought to use the road; a vignette is a small sticker affixed to motor vehicles, which allows them to use specific roads, such as motorways
Data indicating toll payment types		
`TOLL_CHARGE`	Flexible attribute	Indicates the amount and currency of the toll charge
`CURRENCY`	Flexible attribute	Indicates the currency for the toll charge
`TOLL_PAYMENT_TYPE`	Flexible attribute	Indicates how the toll charge can be paid, for example cash, credit card
`TOLL_PAYMENT_PROVIDER_ID`	Flexible attribute	Indicates the payment provider as defined in the region metadata

Data	Type	Description
TOLL_PAYMENT_PROVIDER_DEFINITION	Flexible attribute	Indicates the provider of payment methods at a specific toll gate or toll booth; this is used for toll gates or booths, which require provider-specific cards

The following use cases give examples, how these flexible attributes can be used to model toll systems in NDS.

17.5.1 Define Detailed Toll Information for Roads and/or Lanes

Figure B-13 shows an example of toll lanes with different payment types. NDS provides the following flexible attributes to store detailed toll information:

- tollPaymentType: Contains information on the payment type, such as cash or credit card.
- tollPaymentProviderId: Contains a reference to the the payment provider definition in the region metadata. The payment provider can be a credit card company, for example.

Table 17-10 shows the attributes and attribute groups assigned to the lanes in Figure 17-13.

Figure 17-13 Example of Toll Lanes with Different Payment Types

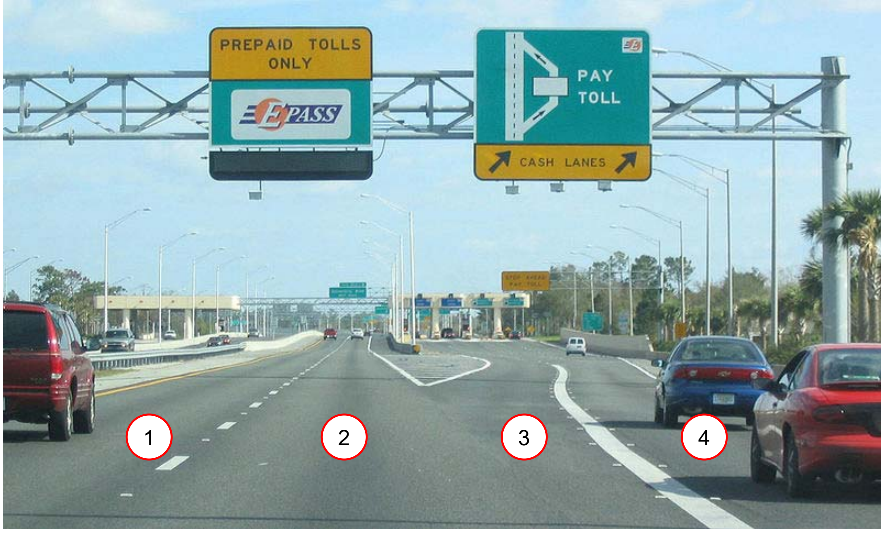


Table 17-10 Group of attributes attached to lanes 1 and 2

Attribute type	Value	Description
toll	IN_POSITIVE_DIRECTION	Classifies the road as a toll road
tollPaymentType	2 (debit card)	Specifies that toll is automatically paid by means of a debit card

Attribute type	Value	Description
tollPaymentProviderId	8	Identification of the debit card company name used for toll payment The value 8 refers to „EPASS“ as tollPaymentProviderDefinition

Table 17-11 Group of attributes attached to lanes 3 and 4

Attribute type	Value	Description
toll (fixed attribute)	IN_POSITIVE_DIRECTION	Classifies the road as toll road
tollPaymentType	0 (cash coins and bills)	Specifies that toll is to be paid in cash and that coins and bills can be used

17.5.2 Modelling Closed Toll Collection Systems

In some countries, for example Japan, toll fees depend on the distance the driver covers on the toll road, a so-called closed toll collection system. The fee to be paid at the toll booth depends on the distance between the toll gates at which the driver enters and leaves the toll road. The driver receives a ticket when entering the toll road. Upon arrival at the exit toll gate, the distance between the entry and the exit is calculated by means of the information on the ticket.

Closed toll collection systems are modeled by means of the following attributes:

- Assign the fixed attribute `toll` for all relevant directions to all links and lines that belong to the toll road.
- Mark the toll gates at the entries and exits to the closed toll collection system with the flexible attributes `TOLL_ENTRY` and `TOLL_EXIT.`

To model the fees between the entry and exit toll gates, the following attributes are used:

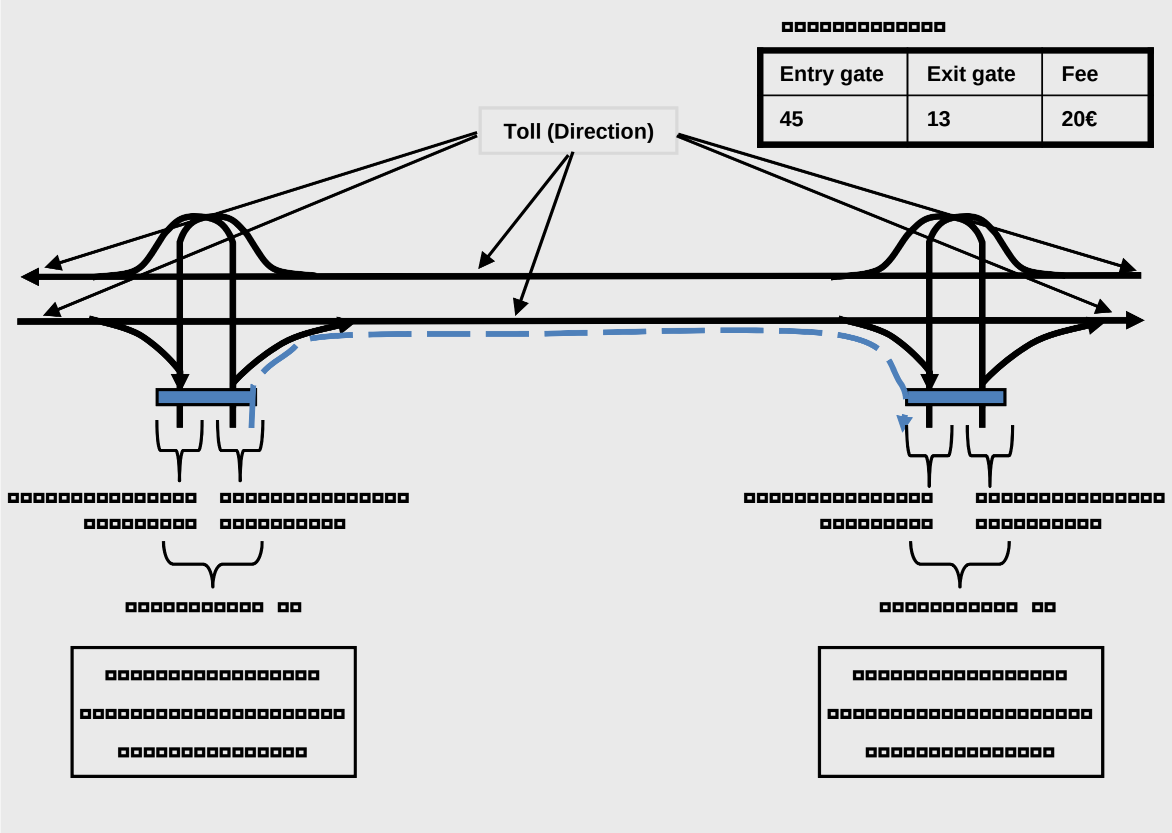
- Assign the flexible attribute `TOLL_GATEID` to the link at which the toll gate is located.
- Add other flexible attributes, such as payment provider ID, toll payment type, and so on, as required (optional).
- Create a `TollCostTable` to store the fees for traveling from one toll gate to another (DataScript location: `nds.routing.main`).

The information about toll fees can be used by routing applications to calculate cost-optimized routes.

Note If the source data does not provide the data for the fee, the gate IDs can also not be stored.

Figure 17-14 depicts the modelling of fees between toll gates in closed toll collection systems.

Figure 17-14 Length-based toll costs in closed toll collection system



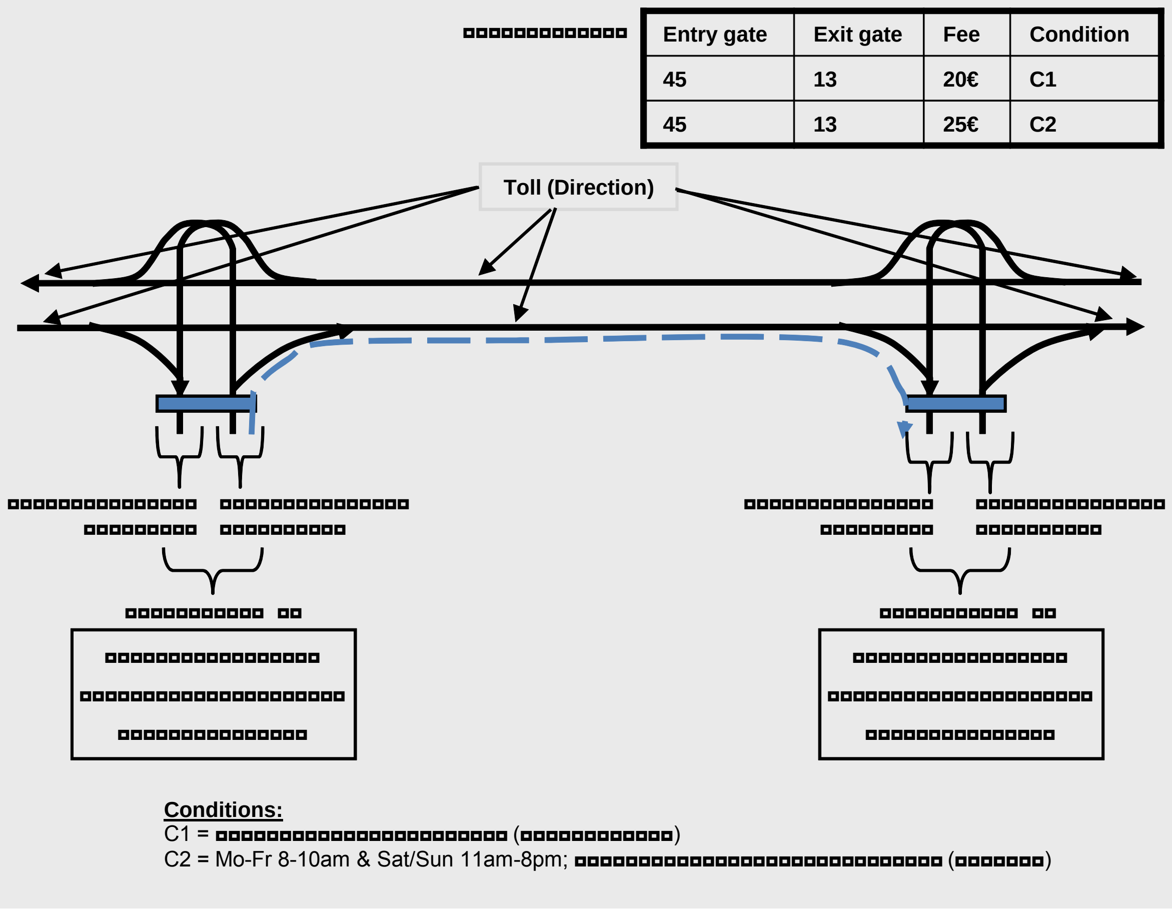
Entry gate	Exit gate	Fee
45	13	20€

More complex situations can be modeled by means of conditions in the toll cost table. Such conditions can be, for example:

- Four persons need to pay a specific fee (condition C1, 20€ in the example).
- Trucks only have to pay toll from Monday to Friday between 8 am and 10 am and on weekends from 11 am to 8 pm (condition C2, 25€ in the example)

The conditions are defined by a list of grouped flexible attributes (see 17.5.1 *Define Detailed Toll Information for Roads and/or Lanes* on page 233). Figure 17-15 shows an example for such a complex situation.

Figure 17-15 Length-based toll costs with conditions



Entry gate	Exit gate	Fee	Condition
45	13	20€	C1
45	13	25€	C2

17.5.3 Modelling Open Toll Systems

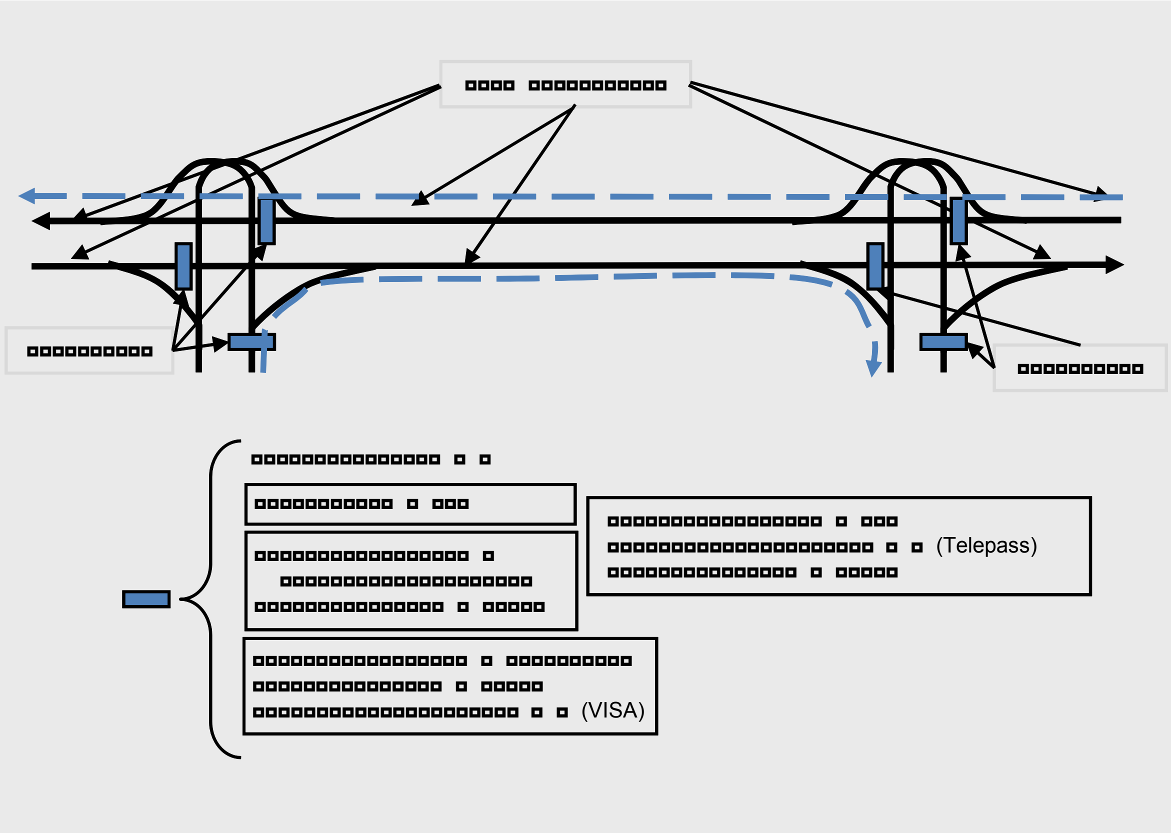
In open toll collection systems (so called “open road tolling” or “free-flow tolling”), the tolls are collected without the use of toll booths. The drivers can drive through the toll gates at highway speeds without having to slow down to receive a ticket or to pay the toll. The toll collection is usually conducted through either the use of transponders or automatic plate recognition.

Open toll systems are modeled by means of the following attributes:

- Assign the fixed attribute `toll` for all relevant directions to all links and lines that belong to the toll system.
- Assign the flexible attribute `TOLL_ENTRY` to the toll gate to indicate that there is a toll gate (optional).
- Add other flexible attributes, such as payment provider ID, toll payment type, and so on, as required (optional).
- Use attribute groups to model specific conditions, for example time-dependent toll cost (see section 17.5.1 *Define Detailed Toll Information for Roads and/or Lanes* on page 233).

This is depicted in Figure 17-16.

Figure 17-16 Open toll system with fixed and variable fees



17.5.4 Modelling Toll Systems Using Vignette

A vignette is a small, colored toll sticker that has to be affixed on the vehicle. The vignette indicates that the required road toll has been paid. Vignettes are used in several countries. In Switzerland, for example, drivers using motorways and expressways are required to purchase an annual vignette. In Austria, a vignette is also required to use motorways and expressways, but it can be purchased for shorter periods, for example for a validity period of five days or two months.

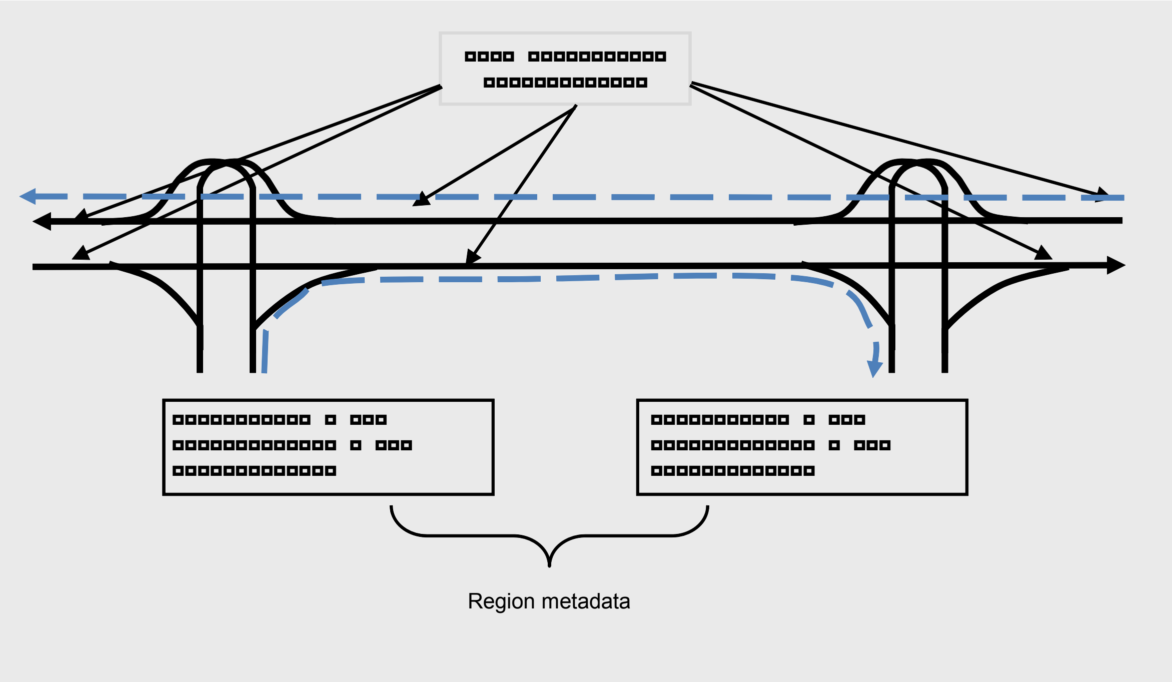
Vignettes can usually be obtained in and outside the respective countries at gas stations and labeled points.

A Vignette-based toll system is modeled in NDS as follows:

- Assign the fixed attribute `toll` for all relevant directions to all links and lines that belong to the toll system.
- Assign the flexible attribute `TOLL_VIGNETTE` to all links where the Vignette applies.

This is depicted in Figure 17-17.

Figure 17-17 Toll system using vignette



17.6 Warning Sign Attributes

Warning signs can be assigned as flexible attributes to road geometry lines and base links grouped with a position given by a `VALIDITY_RANGE` attribute. Warning signs are collected in the attribute type `WARNING_SIGN` (DataScript location: `nds.common.flexibleattributes`). The different signs are defined as constants. New constants may be added as required.

The `WARNING_SIGN` attribute contains all types of warnings known from traffic signs. Examples are:

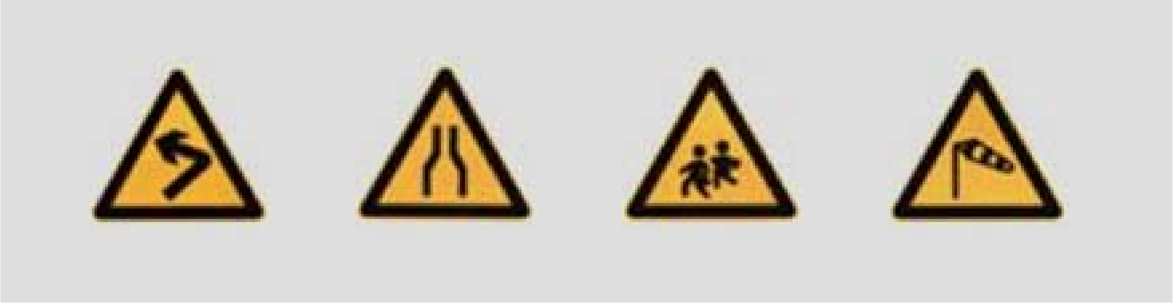
- General hazard
- Curves
- Narrow roads
- Warnings regarding other road users, mostly slow road users such as pedestrians, bikes, etc.

Right-of-way signs, parking signs, and so on, are not included.

Icons for Country-specific Warning Signs

Each country uses different warning signs. Figure 17-18 shows some Japanese warning signs.

Figure 17-18 Examples of Japanese warning signs



In order to use country-specific images for the different types of warning signs, icons may be assigned to warning sign types. For this purpose, corresponding attribute groups consisting of a `WARNING_SIGN` constant and an `ICON_REFERENCE` attribute may be defined as `groupedAttributeList` in the region metadata (DataScript location: `nds.overall.metadata > RegionMetadataTable`), see also *NDS – Format Specification*, 5.2.1 *Flexible Attribute Maps* on page 84.

The icons for the warning signs shall be stored in icon tables in the Basic Map Display building block.

Speed Limit Signs

For speed limit signs and end of speed limit signs, the flexible attribute is used as follows:

- If stored in the region metadata (DataScript location: `nds.overall.metadata` > `RegionMetadataTable` > `groupedAttributeList`), the warning sign must be grouped with the `ICON_REFERENCE` attribute, representing the icon for an empty speed limit sign.

  For end of speed limit signs, the warning sign must be grouped with the `ICON_REFERENCE` attribute, representing the icon for an empty end of speed limit sign.
- If stored at a link, the warning sign is grouped with `SIGN_POSITION` and **not** with `ICON_REFERENCE` as this would increase the data size unnecessarily.

The numeric value for the speed limit sign and the end of speed limit sign is taken from the speed limit attribute that is assigned to the link.

17.7 Storing Legal Speed Limits for Countries

Each country uses different values for the legal speed limits, see also *http://en.wikipedia.org/wiki/Speed_limits_by_country*. To store country-specific speed limits, corresponding attribute groups may be defined as `groupedAttributeList` in the region metadata (DataScript location: `nds.overall.metadata > RegionMetadataTable`).

An attribute group for defining a legal speed limit may consist of the following attributes:

- Speed limit (`SPEED_LIMIT`)
- Vehicle type (example: `BIG_VEHICLES`)

- Legal speed limit range: Within town, outside built-up areas/highways, expressway, motorway (LEGAL_SPEED_LIMIT_RANGE)
- Optional: Weight or other car property restrictions (example: WEIGHT_PER_AXLE_METRIC)
- Optional: Time (example: TIME_RANGE_OF_DAY)
- Optional: Weather conditions (WEATHER)
- Optional: Single or dual lanes (SINGLE_OR_DUAL_CARRIAGEWAY)

For detailed information about flexible attribute maps, refer to *NDS – Format Specification*, 5.2.1 *Flexible Attribute Maps* on page 84.

17.8 Parking Places as Motorway Ramps (Routing and Guidance)

An exit ramp from a motorway leads into a parking place. From this parking place, drivers can:

- Leave the motorway
- Go back to the motorway they came from
- Drive onto another motorway

Table 17-12 shows the attributes that are assigned to the link representing the ramp in order to model this real-world situation.

Table 17-12 Attribute group for parking places as motorway ramps

Attribute name and type	DataScript location	Value
linkType (fixed)	nds.common.fixedattributes > SharedRoadAttributes	RAMP
PARKING (flexible)	nds.common.flexibleattributes	Values for ParkingFacilityType and ParkingInOutType Example: – ParkingFacilityType = OPEN_PARKING_SPACE – ParkingInOutType = PARKING_ENTRANCE_EXIT
complexInters ection (fixed)	nds.common.fixedattributes > RoutingRoadAttributes	TRUE

17.9 Use of Bypasses, Side Streets, or Shortcuts as Routing Options

In Japanese navigation devices, the use of bypasses, side streets and shortcuts can be specified as a routing option. These types of road allow drivers to:

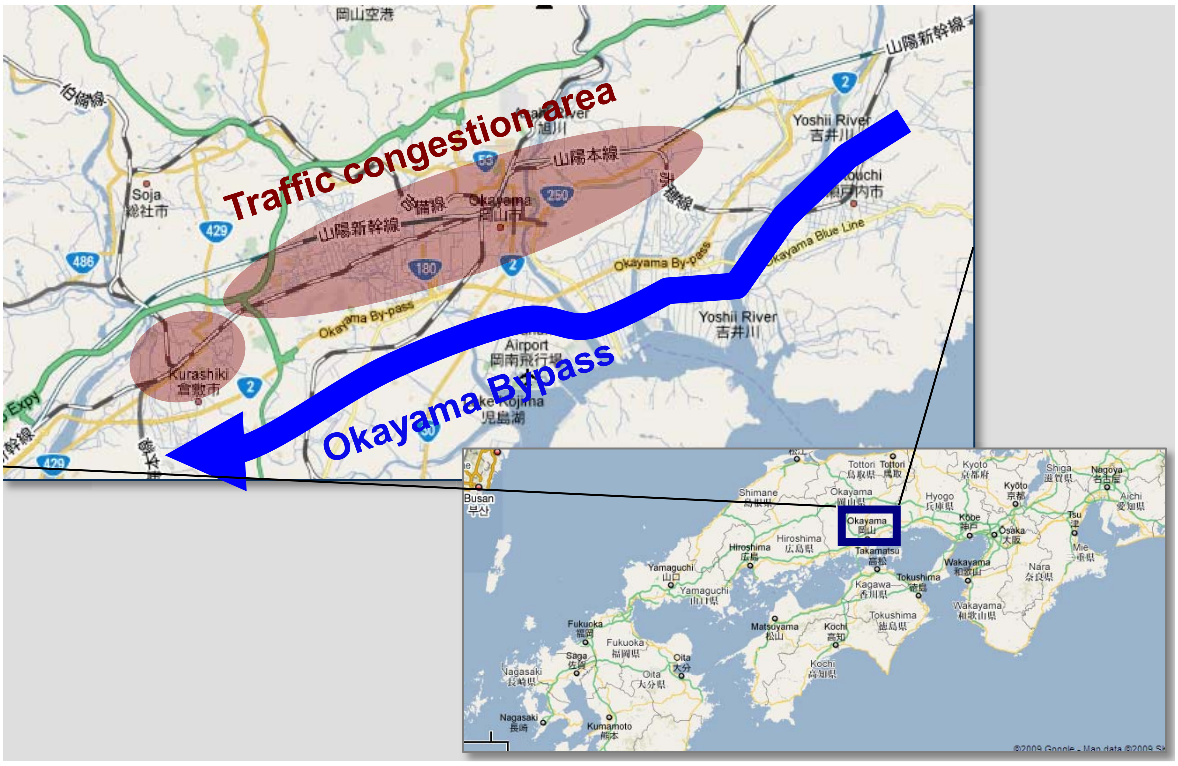
- Avoid congested built-up areas by using a road that bypasses the respective town or village, see Figure 17-19

  In Japan, the links of bypass roads are classified by the government.
- Avoid a link (or a set of links) that is usually prone to traffic congestion or links/ intersections showing a high accident rate by using backroads or side streets

  The links are classified by the data supplier based on local knowledge.
- Avoid dense traffic, for example, intersections between major points along the route, by using smaller roads marked as shortcuts

  The links are classified by the data supplier based on local knowledge.

Figure 17-19 Example of a bypass road in Japan



NDS provides three flexible attributes to model such roads and facilitate these routing options (DataScript location: `nds.common.flexibleattributes`). These attributes may be assigned to links.

- BYPASS
- SHORTCUT
- SIDE_STREET

These attributes shall only be used if the data supplier can offer such data and only in order to enable/disable non-dynamic routing options.

17.10 Dynamic Status Update for Roads under Construction

Links representing roads under construction are assigned the flexible attribute UNDER_CONSTRUCTION which may be grouped with a time domain attribute like DATE_RANGE_OF_YEAR to model the duration of the construction work. However, as construction may be delayed, the following approach may be used to update the status of roads under construction in an incremental update:

- Assign global feature IDs to the links representing roads under construction. For this purpose, NDS provides the flexible attribute GLOBAL_FEATURE_ID, see also *NDS – Format Specification*, *Identifying Features* on page 58.
- Create a RoutingNewlyOpenedLinkTable (DataScript location: nds.routing.main).
- Store the links' global feature IDs together with the opening date of the road in this table.

When the status of roads under construction changes, the RoutingNewlyOpenedLinkTable can be added or updated in an incremental update. The application can query the status of roads under construction with a simple query, for example:

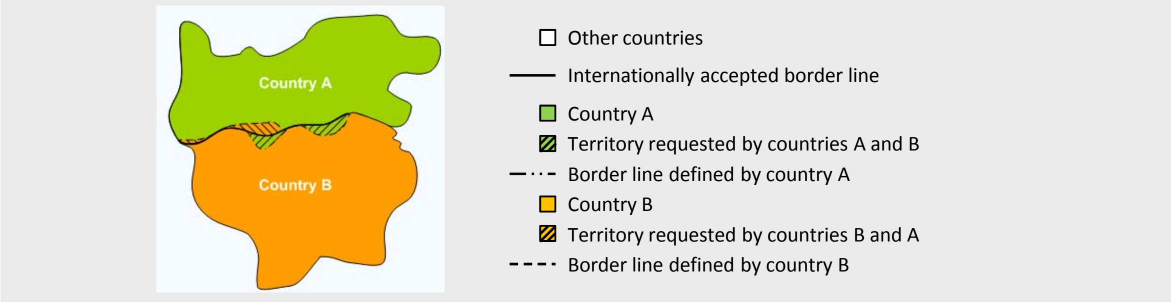
```
select * from RoutingNewlyOpenedLinksTable where globalFeatureId = [value of
GLOBAL_FEATURE_ID].
```

If there is no result, the road has not been opened yet. If there is a result, the road has been opened.

17.11 Geopolitical Borders

Geopolitical borders denote administrative boundaries between countries. Between some countries, border disputes exist. Examples are the Kashmir and the Cyprus border disputes. Figure 17-20 shows an example of a disputed borderline between two countries.

Figure 17-20 Disputed borderline between two countries – example


Country A
Country B
Other countries
Internationally accepted border line
Country A
Territory requested by countries A and B
Border line defined by country A
Country B
Territory requested by countries B and A
Border line defined by country B

The application has to display the borderline for disputed borders according to the acceptance or rejection by the respective country, in which the driver is located. Therefore, the border displayed in basic map display depends on the country, in which the car is positioned.

NDS provides the following data types to model geopolitical borders:

The `DisputantDefinitionTable` stores the metadata for geopolitical borders (see Table 17-13; DataScript location `nds.bmd` > `main`).

Table 17-13 Metadata for geopolitical borders

Metadata	Description
`disputantId`	Identifies the country or group of countries that share the same opinion about geopolitical borders
`versionId`	Version ID; used for incremental updates
`isoCountryCode`	Used to determine regions and map these regions to country specifics within an application
`isoSubCountryCodes`	Defines the names of the main subdivisions (provinces, states, for example) of all countries

The BMD feature class `LINE_BORDER` indicates the line of the border, which is internationally accepted (DataScript location: `nds.bmd` > `common`). The flexible attributes `BORDER_IS_DISPUTED_BY` and `BORDER_IS_ACCEPTED_BY` are assigned to the line feature and specify the countries that accept or dispute this border.

The BMD feature class `LINE_BORDER_DISPUTED` indicates the line of the border that deviates from the internationally accepted border. Again, the flexible attributes `BORDER_IS_DISPUTED_BY` and `BORDER_IS_ACCEPTED_BY` indicate the countries that accept or dispute this deviating border.

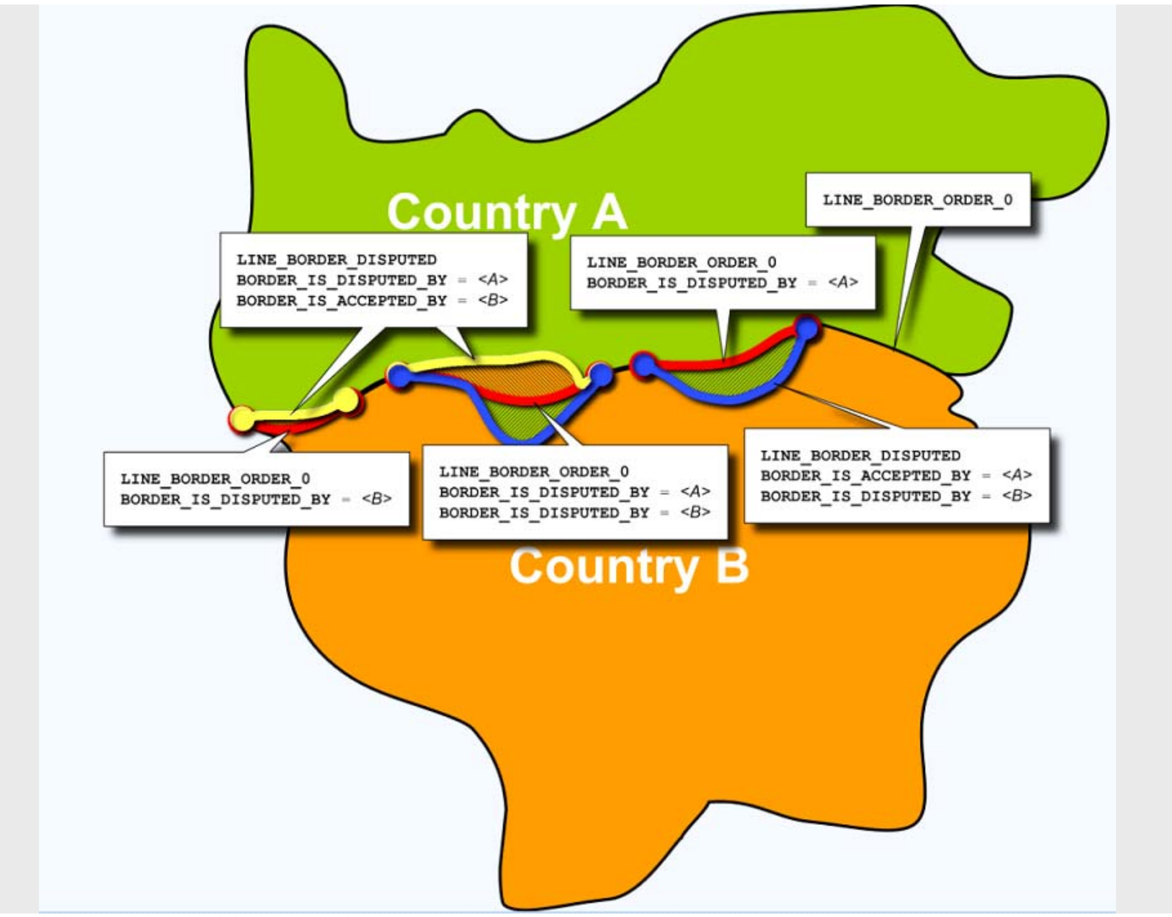
The flexible attributes contain attribute lists with IDs identifying those parties, who accept or dispute the respective border.

Figure 17-21 shows how the BMD feature classes and flexible attributes are used to model the deviating borders. The application can use this information to:

- Display the borderline in all countries that are contained in the attribute list of the flexible attribute `BORDER_IS_ACCEPTED_BY`
- Hide the borderline in the countries that are contained in the attribute list of `BORDER_IS_DISPUTED_BY`

Based on the line features and flexible attributes in the example, the application displays the yellow border in country B, but not in country A. The blue border is displayed in country A, but not in country A.

Figure 17-21 Flexible attributes assigned to map lines to display disputed borderlines


Country A
LINE_BORDER_ORDER_0
LINE_BORDER_DISPUTED
BORDER_IS_DISPUTED_BY = <A>
BORDER_IS_ACCEPTED_BY = <B>
LINE_BORDER_ORDER_0
BORDER_IS_DISPUTED_BY = <A>
LINE_BORDER_ORDER_0
BORDER_IS_DISPUTED_BY = <B>
LINE_BORDER_ORDER_0
BORDER_IS_DISPUTED_BY = <A>
BORDER_IS_DISPUTED_BY = <B>
LINE_BORDER_DISPUTED
BORDER_IS_ACCEPTED_BY = <A>
BORDER_IS_DISPUTED_BY = <B>
Country B

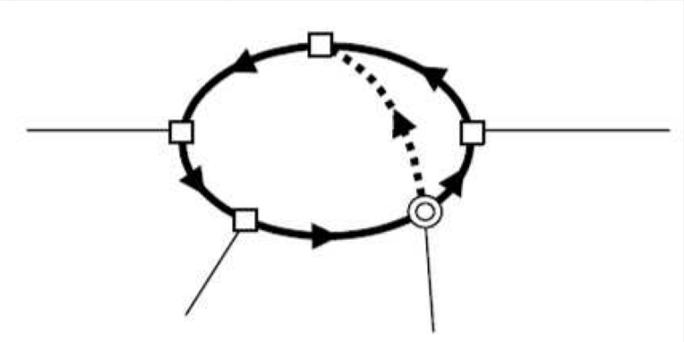
17.11.1 Treaty Line

Treaty lines denote administrative boundaries, which represent a border agreed upon in an international treaty. An example for a treaty line is the Golan Heights border between Syria and Israel. To classify treaty lines for map display, the BMD feature class `LINE_BORDER_TREATY` may be used.

17.12 Modeling Roundabouts with Middle Traversal

For roundabouts that offer the possibility to drive through the middle, the flexible attribute `ROUNDABOUT_INTERIOR` can be assigned to the link traversing the roundabout. This is depicted in Figure 17-22.

Figure 17-22 Roundabout with Middle Traversal



Flexible attribute assigned to the
• Solid line: □□□□□□□□□□
• Dotted line: □□□□□□□□□□□□□□□□□□□

17.13 Skipping Attributes During Parsing of an Attribute Map List

When parsing the attributes in an attribute map list, the application may encounter attributes, which it either does not need or does not know. The following procedure describes how an application can skip the parsing of these attributes in an `AttributeMapList`.

An attribute map list consists of two arrays:

- `attrTypeRef`

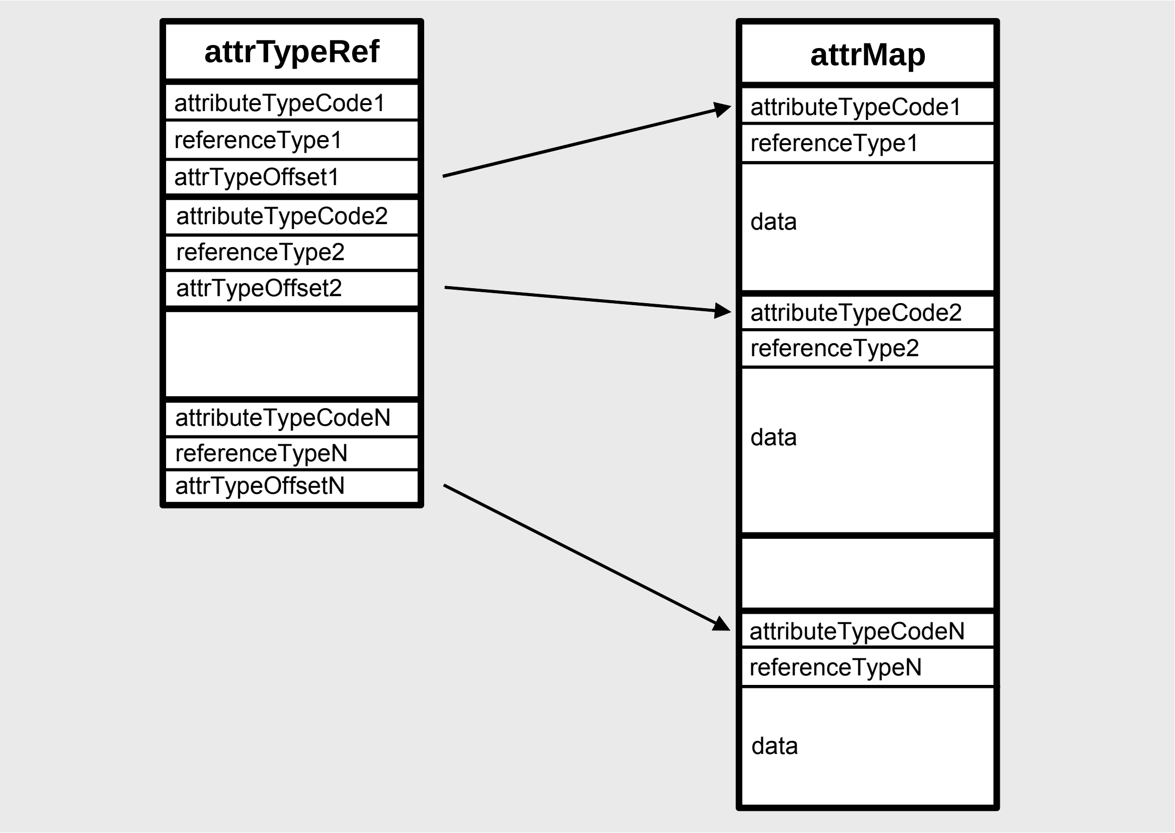
  This array contains a list of offsets for each pair of `attributeTypeCode` and `referenceType` in the `attrMap`.
- `attrMap`

  This array contains values for each pair of `attributeTypeCode` and `referenceType`.

The two arrays have the same number of elements and contain the `attributeTypeCode` and `referenceType` combination in the same order. The `attrTypeOffset` contains the byte position of the respective pairs of `attributeTypeCode` and `referenceType` within the BLOB. The `byteOffset` is determined from the start of the BLOB. This is depicted in Figure 17-23.

The application parses the `attrTypeRef` array before or at the same time as the `attrMap` array until it reaches an `attributeTypeCode` in the `attrTypeRef` array that is unknown or needs to be skipped. To skip the attribute, the application sets the bit position (8 * offset) of the bitstream reader for the `attrMap` BLOB to the `attrTypeOffset` of the next `atrributeTypeCode`.

Figure 17-23 Parsing attributes in an AttributeMapList



A Glossary

This chapter explains the most important terms and key concepts for Navigation Data Standard.

Aggregated link

Filtering of less important roads results in a high number of intersections with two adjacent links only. These intersections can be removed by merging the adjacent links. The resulting new links are called aggregated links.

Baseline map

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

Clothoid

Clothoids are a method to describe enhanced ADAS geometry. NDS defines a clothoid segment by means of the coordinates of its start and end point, and the start and end tangent.

Curvature

Curvature values can be stored in the ADAS layer as attributes attached to shape or attribute points of road features with linear interpolation between the values.

Data carrier

Generally, a data carrier is a physical medium to store digital data persistently. This can be, for example, a DVD, a memory stick, or a hard disk.

In the context of NDS, a data carrier is a medium for storing data complying with the NDS format. The data carrier can be installed permanently in an NDS-compliant system, for example, on a hard disk, or it can be used to transport data to an NDS-compliant system, for example, using a DVD.

Input data

Data provided for a compiler to be converted to an NDS-compliant storage format.

NDS database

An NDS database is the set of data complying with NDS format that is stored in a particular navigation system supporting NDS. An NDS database may consist of one or more product databases.

Product database

A product database contains navigation data for one or more update regions. An NDS database may consist of one or more product databases.

Profile

In Navigation Data Standard, the term **profile** is used as follows:

- ADAS profiles: A purely local profile, meaning a continuous course of attributes along the coordinates of a link.
- Generic profile attributes: A time profile formed by properties of features that vary over time. One example are speed profiles, which describe varying driving speeds over the course of a day for a given road section and travel direction.

South-west rule

Navigation Data Standard defines a tile as follows: If (x1, y1) is the south-west corner of a tile and (x2, y2) is its north-east corner, then all points (x, y) with x1 ≤ x < x2 and y1 ≤ y < y2 are uniquely assigned to the tile. The western and southern borders belong to the tile whereas the eastern and northern borders belong to the neighboring tiles.

Texel

Acronym of texture cell. Fundamental unit of texture space: Textures are represented by arrays of texels.

Update region

Represents a geographic area in a database that can be subject to an update. Any database that complies with Navigation Data Standard may be logically divided into update regions which are completely disjoint but may overlap at defined points, namely the gateways connecting 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 vendors or car manufacturers.

B List of Indices for NDS Relational Tables

The following list indicates the columns in the relational tables in NDS for which an index should be provided.

Note The list is a work in progress and not yet complete.

Table B-1 List of indices for NDS relational tables

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
OverallMetadataTable overallMetadataTable	PK: (dataModelVersionId)	Find overall metadata by data model version	PK
BmdTileTable bmdTileTable	PK: (id)	Find a tile by ID	PK
	Index on: (versionID)	For updates; *can be created by the application before the update starts*	temp
BmdFeatureClassHierarchyTable bmdFeatureClassHierarchyTable	PK: (parentClass, childClass)	*Most likely it is not necessary to search for a parent and a child at the same time*	PK
	Index on: (childClass)	Find all parents for the given child	opt
BmdTilePatternTable bmdTilePatternTable	PK: (tilePatternId)	Find a tile pattern by pattern ID	PK
	Index on: (versionID)	For updates; *can be created by the application before the update starts*	temp
ScaleLevelTable bmdScaleLevelTable poiScaleLevelTable tiScaleLevelTable	PK: (scaleLevelId)	Find a scale level by scale level ID	PK
	Index on: (minScaleDenominator, maxSaleDenominator)	Find scale levels in the given range Find scale levels greater than the given minimal value	opt
	Index on: (maxScaleDenominator)	Find scale levels smaller than the given maximal value	opt

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
IconTable bmdIconTable poiIconTable tiIconTable signpostIconTable nameIconTable	PK: (iconID)	Find an icon by icon ID	PK
	Index on: (iconSetId)	Find an icon by icon set ID	
	Index on: (usageTypeMask)	*Rarely used; most queries would use (iconSetId, scaleLevelId) for the main filtering and simply scan through the intermediate results if additional filter are set, for example, usageTypeMask == TWO_D*	
	Index on: (versionID)	For updates; *can be created by the application before the update starts*	temp
IconCollectionTable bmdIconCollectionTable poiIconCollectionTable tiIconCollectionTable signpostIconCollectionTable	PK: (collectionId)	Find a collection by collection ID	PK
	Index on: (versionID)	For updates; *can be created by the application before the update starts*	temp
	Index on: (collectionName)	Find a collection by name *Not necessary for a very small table*	opt
	Index on: (languageCode)	Find a collection by language *Not necessary for a very small table*	opt
RoutingTileTable routingTileTable	PK: (id)	Find a tile by ID	PK
	Index on: (versionID)	For updates; *can be created by the application before the update starts*	temp
RoutingAuxTileTable routingAuxTileTable	PK: (id)	Find a tile by ID	PK

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
	Index on: (versionID)	For updates; *can be created by the application before the update starts*	temp
RoutingNewlyOpenedLinkTable routingNewlyOpenedLinkTable	PK: (globalFeatureId)	Find a newly opened road by global feature ID	PK
TollCostTable tollCostTable	PK: (gateIdFrom, gateIdTo, sequenceNumber)	Find toll cost by from and to toll gate IDs	PK
NamedObjectTable namedObjectTable	PK: (namedObjectId)	Find a named object by named object ID	PK
	Index on: (versionID)	For updates; *can be created by the application before the update starts*	temp
NvcTable nvcTable poiNvcTable	PK: (namedObjectId, selectionCriterion, nvcSubTreeId)	Find the root NVC subtree (ID is 0) for the given named object ID Find an NVC BLOB by named object ID, selection criterion, and NVC tree ID. Find an NVC BLOB by named object ID (useful for updates)	PK
SelectionMetadataTable selectionMetadataTable	PK: (id)	Find a selection metadata for the given update region ID	PK
SelectionGraphTable selectionGraphTable	PK: (id)	Find a selection graph for the given update region ID	PK
PoiTable poiTable	PK: (poiId)	Find a POI by POI ID	PK
	Index on: (versionID)	For updates; *can be created by the application before the update starts*	temp
	Index on: (regionID, postalCode)	Find a POI by region ID Find a POI by region ID and postal Code	opt
	Index on: (postalCode)	Find a POI by postal code	opt
	Index on: (phone)	Find a POI by phone number	opt
PoiRelationTable poiRelationTable	PK: (childPoiId, parentPoiId)	Most likely it is not necessary to search for parent and child ID at the same time	PK

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
	Index on: (parentPoiId)	Find all children for a given POI	mand
	Index on: (relationType)	Find all `LOGICAL_ACCESS` relations in a given bounding box	opt
`PoiCategoryTable` `poiCategoryTable`	PK: (catId)	Find a category by ID	PK
	Index on: (standardCat)	Find a category ID for the given standard category *Usually not necessary; the application can create a mapping between standard category and category ID once during initialization*	opt
	Index on: (scaleLevelId)	Find all categories, which can be displayed in the current scale level	opt
	Index on: (activationRadius)	Find categories, which can be activated from the current position	opt
	Index on: (selectionEntry)	Find categories, which can be displayed in the initial list of categories	opt
`PoiCatRelationTable` `poiCatRelationTable`	PK: (parentCatId, childCatId)	Find all children for the given category; *most likely it is not necessary to search for parent and child at the same time*	PK
	Index on: (childCatId)	Find all parents for a given category	opt
`PoiServiceLocationTable` `poiServiceLocationTable`	PK: (serviceLocationId)	Find all service locations for the given POI; *most likely it is not necessary to search for the given POI and the given category*	PK
	Index on: (poiId)	Find all service locations for the given POI ID	opt
	Index on: (catId)	Find all service locations for the given category	mand
	Index on: (mortonCode, catId)	Find all service locations in a given area Find all service locations in a given area by category ID	mand

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
PoiNameStringTable poiNameStringTable	PK: (nameStringId)	Find a name for the given name string ID	PK
	Index on: (nameStringId, languageCode)	Find a name by name string ID Find a name by name string ID and language code *Usually used when joining* poiToNameStringTable *and* poiNameStringTable*, not necessary for small number of languages*	opt
	Index on: (nameString, languageCode)	Find a name for a given input string Find a name for a given input string and the given language code	mand
PoiNameStringRelationTable poiNameStringRelationTable	PK: (originNameStringId, targetNameStringId)	Find IDs of all name strings, which are related to the given origin name string; *most likely it is not necessary to search for origin and target at the same time*	PK
	Index on: (originNameStringId, nameStringRelationType)	Find IDs of all name strings, which are related to a given origin name string ID	opt
PoiToNameStringTable poiToNameStringTable	PK: (poiId, nameStringId)	Find name string ID related to the given POI ID; *most likely it is not necessary to search for a poi-to-name relation with a given POI ID and the name string ID*	PK
	Index on: (poiId, nameStringUsageType)	Find IDs of all name strings, which are related to the given POI ID with the given usage type. *Usually not needed since there is a small number of usage types for the given name.*	opt

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
PoiNameToPhoneticTraTable `poiNameToPhoneticTraTable`	PK: (nameStringId, languageCode, phoneticTraType, orderOfPreference)	Find a phonetic representation for the given name string ID, language code, phonetic representation type, and order of preference. *Most likely it is not necessary to search for a combination of a phonetic transcription ID and a name string*	PK
PoiCatNameStringTable `poiCatNameStringTable`	PK: (nameStringId)	Find a name by name string ID	PK
	Index on: (nameStringId, languageCode)	Find a name by name string ID Find a name by name string ID and language code *This is usually used when joining* `poiCatToNameStringTable` *and* `poiCatNameStringTable`*; not necessary for small number of languages*	opt
	Index on: (nameString, languageCode)	Find a name for a given input string Find a name for a given input string and a given language code	mand
PoiCatNameStringRelationTable `poiCatNameStringRelationTable`	PK: (originNameStringId, targetNameStringId)	Find IDs of all name strings, which are related to the given origin name string; *most likely there is no need to search for the origin and the target at the same time*	PK
	Index on: (originNameStringId, nameStringRelationType)	Find IDs of all name strings which are related to a given origin name string; *usually not needed since there is a small number of target names for the given origin*	opt

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
PoiCatToNameStringTable poiCatToNameStringTable	PK: (catId, nameStringId)	Find a name string ID related to the given cat ID; *most likely it is not necessary to search for cat-to-name relation with the given cat ID and the name string*	PK
	Index on: (catId, nameStringUsageType)	Find IDs of all name strings, which are related to the given category ID with the given usage type; *usually not needed since there is a small number of usage types for the given name*	opt
ObjectToCatTable poiNamedObjectToCatTable	PK: (namedObjectId,catId)	*Most likely there is no need to search for the named object and category at the same time*	PK
	Index on: (catId)	Find all named objects containing given category	opt
	Index on: (namedObjectId)	Find all categories contained by the given named object	opt
PoiCatNameToPhoneticTraTable poiCatNameToPhoneticTraTable	PK: (nameStringId, languageCode, phoneticTraType, orderOfPreference)	Find a phonetic representation for the given name string ID, language code, phonetic representation type, and order of preference *Most likely it is not necessary to search for the combination of a phonetic transcription ID and a name string ID*	PK
PoiAttrNameStringTable poiAttrNameStringTable	PK: (nameStringId)	Find a name by name string ID	PK
	Index on: (nameStringId, languageCode)	Find a name by name string ID Find a name by name string ID and language code *This is usually used when joining* poiAttrToNameStringtable *and* poiAttrNameStringTable*; not necessary for small number of languages*	opt

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
	Index on: (nameString, languageCode)	Find a name for the given input string Find a name for the given input string and the given language code	mand
PoiAttrNameStringRelationTable poiAttrNameStringRelationTable	PK: (originNameStringId, targetNameStringId)	Find IDs of all name strings, which are related to the given origin name string; *most likely it is not necessary to search for origin and target at the same time*	PK
	Index on: (originNameStringId, nameStringRelationType)	Find IDs of all name strings related to the given origin name string with the given relation. *Usually not needed since there is a small number of target names for the given origin*	opt
PoiAttrToNameStringTable poiAttrToNameStringTable	PK: (attrId, valueId, nameStringId)	Find a name string ID related to the given attribute ID Find a name string ID related to the given attribute ID and value ID. *Most likely it is not necessary to search for attribute-to-name relation with a given attribute ID and name string*	PK
	Index on: (attrId, valueId, nameStringUsageType)	Find IDs of all name strings, which are related to the given attribute and value ID with the given usage type *Usually not needed since there is a small number of usage types for the given name*	opt
PoiAttrNameToPhoneticTraTable poiAttrNameToPhoneticTraTable	PK: (nameStringId, languageCode, phoneticTraType, orderOfPreference)	Find a phonetic representation for the given name string ID, language code, phonetic representation type, and order of preference *Most likely there is no need to search by using all the fields comprising the PK*	PK

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
PoiNamedObjectRelationTable poiNamedObjectRelationTable	PK: (regionId, namedObjectId)	Find a named object for the given region ID *Most likely there is no need to search by using all the fields comprising the PK*	PK
	Index on: (namedObjectId)	Find a region for the given named object	mand
PoiGeoAccessTable poiGeoAccessTable	PK: (poiId, mortonCode)	Find all access points for the given POI ID *Most likely there is no need to search for a geo access with the given ID and the given Morton code*	PK
	Index on: (mortonCode)	Find all access points in a given area; *this can be used for pre-fetching access points*	opt
PoiLinkAccessTable poiLinkAccessTable	PK: (poiId, linkId, position)	Find link IDs for the given POI ID. *Most likely there is no need to search for a link access with the given poi ID and the given link ID*	PK
	Index on: (linkId)	Find for a given link ID	opt
	Index on: (position)	Find link IDs for a given POI ID	mand
PoiIntersectionAccessTable poiIntersectionAccessTable	PK: (poiId, intersectionId, tileId)	Find an intersection access for the given POI ID *Most likely there is no need to search for an intersection access with the given POI ID and the given intersection ID and the given tile ID*	PK
	Index on: (poiId)	Find an intersection access for a given POI ID	mand
	Index on: (tileId, intersectionId)	Find all intersection access for a given tile ID Find all intersection access for a given tile ID and intersection ID	opt
PoiMetadataTable poiMetadataTable	PK: (id)	Fine POI metadata by ID	PK

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
PoiCatToAttrTable poiCatToAttrTable	PK: (catId, attrId)	Find all attributes for the given category ID *Most likely there is no need to search by using all the fields comprising the PK*	PK
	Index on: (attrId)	Find all categories for a given attribute ID	opt
PoiVirtualTileTable poiVirtualTileTable	PK: (tileId)	Find all tile versions for a given packed tile ID	PK
	Index on: (minId)	Find all tile IDs greater than given min ID	opt
	Index on: (maxId)	Find all tile IDs less than given max ID	opt
	Index on: (versionID)	For updates; *can be created by the application before the update starts*	temp
	Index on: (isDirty)	For updates; *can be created by the application before the update starts*	temp
DtmHeightMapTileTable dtmHeightMapTileTable	PK: (id)	Find a tile by tile ID	PK
	Index on: (versionId)	For updates; *can be created by the application before the update starts*	temp
DtmBdamTileTable dtmBdamTileTable jvDtmBdamTileTable	PK: (id)	Find a tile by tile ID	PK
	Index on: (versionID)	For updates; *can be created by the application before the update starts*	temp
OrthoimageTileTable orthoimageTileTable	PK: (id)	Find a tile by tile ID	PK
	Index on (versionId)	For updates; *can be created by the application before the update starts*	temp
BuildingBlockTable buildingBlockTable	(buildingBlockId)	Find a building block by ID	PK
	Index on BuildingBlockType	Find a building block by type; not needed for a small table	temp

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
	Index on BuildingBlockName	Find a building block by name; not needed for a small table	temp
UpdateRegionTable updateRegionTable	PK: (updateRegionId)	Find an update region for a given ID	PK
	Index on (versionId)	For updates; *can be created by the application before the update starts*	temp
	Index on (isDirty)	For updates; *can be created by the application before the update starts*	temp
VersionTable versionTable	PK: (versionId)	Find a version by ID	PK
	Index on (versionName)	For updates; *can be created by the application before the update starts*	temp
	Index on (compilerVersion)	For updates; *can be created by the application before the update starts*	temp
	Index on: (compilerConfiguration)	For updates; *can be created by the application before the update starts*	temp
	Index on: (creationDateTime)	For updates; *can be created by the application before the update starts*	temp
BBlockCompVersionTable bBlockCompVersionTable	PK (updateRegionId, buildingBlockId, sequenceNumber, tableName, columnName)	Find a version by update region id, building block id, sequence number, table name and/or column name	PK
	Index on: (tableName, columnName)	Find versions by table name and/or column name	opt
UrBuildingBlockVersionTable urBuildingBlockVersionTable	PK: (updateRegionId, buildingBlockId, sequenceNumber)	Find update region metadata for all building blocks for the given update region ID Find update region metadata for the given building block ID and update region ID *Most likely it is not necessary to search by using all fields of the PK*	PK

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
	Index on: (versionId)	For updates; *can be created by the application before the update starts*	temp
	Index on: (buildingBlockId)	Find update region metadata for a given update building block ID	opt
	Index on: (isDirty)	For updates; *can be created by the application before the update starts*	temp
	Index on: (isPartiallyFilled)	For updates; *can be created by the application before the update starts*	temp
GlobalGatewayTable globalGatewayTable	(gatewayId, updateRegionId, tileId)	Find all update regions for the given gateway ID; needed for routing. *Most likely there is no need to search by using all the fields comprising the PK*	PK
	Index on: (updateRegionId)	Find all gateways for a given update region ID	opt
ColorTable colorTable	PK: (colorId)	Find a color by color ID	PK
FontTable fontTable	PK: (fontId)	Find a font by font ID	PK
NdsDatabaseSupplierTable ndsDatabaseSupplierTable	PK: (ndsDbDupplierId)	Find a supplier by supplier ID	PK
ProductDbTable productDbTable	PK: (ndsDbSupplierId, productId)	Find product data by supplier Find product data by supplier and product	PK
	Index on: (versionId)	For updates; *can be created by the application before the update starts*	temp
	Index on: (productName)	Find product data by name; *not needed if the table is small*	opt
	Index on: (isOverviewMap)	Find overview maps; *not needed if the table is small*	opt
	Index on: (baselineMapId)	For updates; *can be created by the application before the update starts*	opt
DataModelVersionTable dataModelVersionTable	PK: (dataModelVersionId)	*Table has only one row, no need to search*	PK

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** ***Comment***	**Type**
SpatialSubTreeTable spatialSubTreeTable jvSpatialSubTreeTable	PK: (spatialSubTreeId)	Find a spatial subtree by subtree ID	PK
BodyGeometryBlobTable bodyGeometryBlobTable jvBodyGeometryBlobTable	PK: (blobId)	Find a body geometry by body geometry ID	PK
TextureMapTable object3DTextureMapTable jvTextureMapTable	PK: (textureMapId)	Find a texture map by texture map ID	PK
TmcTileTable tmcTileTable	PK: (itileId, locTableId)	Find TMC tile for the given tile ID and location table ID *Most likely there is no need to search by using all the fields comprising the PK*	PK
	Index on: (locTableId)	Find TMC tiles for the given location table ID	mand
CipherKeyInfoTable cipherKeyInfoTable	PK: (keyInfoId)	Find a key info by key info ID	opt
EncryptableItemCodeTable encryptableItemCodeTable	PK: (eic)	*No need to search by* eic	PK
	Index on: (tableName, columnName)	Find eic for a given table name and column name	opt
LanguageTable languageTable	PK: (languageCode)	Find language by internal language code	PK
	Index on: (isoCountryCode)	Find language by ISO country code	opt
	Index on: (isoLanguageCode)	Find language by ISO language code	opt
JvFromLinkTable jvFromLinkTable	PK: (fromLink, position, seqNr)	Find a junction view for the given 'from' link *Most likely there is no need to search by using all the fields comprising the PK*	PK
	Index on: (versionId)	For updates; *can be created by the application before the update starts*	temp

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
JvFrom2ToLinkTable jvFrom2ToLinkTable	PK: (fromLink, toLink, pathNr, seqNr)	Find image by image ID *Usually used when joining jvFromLink/jvFromToLink with jvImages tables* *Most likely there is no need to search by using all the fields comprising the PK*	PK
	Index on: (versionId)	For updates; *can be created by the application before the update starts*	temp
JvImageTable jvImageTable	PK: (imageId, dayLight, weather, width, height, colorDepth, jvImageFormat)	Find image by image ID *Usually used when joining jvFromLink/jvFromToLink with jvImages tables* *Most likely there is no need to search by using all the fields comprising the PK*	PK
	Index on: (versionId)	For updates; *can be created by the app, just before the update starts*	temp
	Index on: (dayLight)	Find junction images for the day or night *Rarely used; queries would use (imageId) for main filtering and simply scan through the intermediate results if additional filter are set, lfor example dayLight or weather*	opt
	Index on: (weather)	Find junction images for the given season *Rarely used; queries would use (imageId) for main filtering and simply scan through the intermediate results if additional filter are set, for example, dayLight or weather*	opt
JvTileVersionTable jvTileVersionTable	PK: (packedTileId)	Find all versions of all junction views for a given packed tile ID	PK
	Index on: (versionId)	For updates; *can be created by the application before the update starts*	temp

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
	Index on: (isDirty)	For updates; *can be created by the application before the update starts*	temp
FtsMetadataTable ftsPoiMetadataTable ftsRoadNameMetadataTable ftsCrossroadNameMetadataTable	PK: (buildingBlockId)	Find FTS metadata by building block ID	PK
FtsColumnMetadataTable ftsPoiColumnMetadataTable ftsRoadNameColMetaTable ftsCrossroadNameColMetaTable	PK: (columnName)	Find metadata for a given column	PK
PhoneticTranscriptionAuxTable phoneticTranscriptionAuxTable	PK: (phoneticTranscriptionAuxId)	Find phonetic transcription auxiliary by the given ID	PK
NamedObjectPrerecordedVTable NamedObjectPrerecordedVTable	PK: (prerecordedVoiceId)	Find prerecorded voice for the given ID	PK
NamedObjectToPrerecordedVTable namedObjectToPrerecordedVTable	PK: (namedObjectId, nameStringIx, prerecordedVoiceType, prerecordedVoiceId)	Find a prerecorded voice for the given name object ID, name string index, voice type and voice ID *Most likely there is no need to search by using all the fields comprising the PK*	PK
	Index on: (namedObjectId)	Find all name string indexes with their types for the given named object ID	opt
PoiNameToPrerecordedVTable poiNameToPrerecordedVTable	PK: (nameStringId, languageCode, prerecordedVoiceType, orderOfPreference, prerecordedVoiceId)	Find prerecorded voice for the given name string ID, language, prerecorded voice type, and ordered of preference *Most likely there is no need to search by using all the fields comprising the PK but by name string ID only*	PK

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
PoiCatNameToPrerecordedVTable poiCatNameToPrerecordedVTable	PK: (nameStringId, languageCode, prerecordedVoiceType, orderOfPreference, prerecordedVoiceId)	Find prerecorded voice for the given name string ID, language, prerecorded voice type, and order of preference *Most likely there is no need to search by using all the fields comprising the PK but by names string ID only*	PK
PoiAttrNameToPrerecordedVTable poiAttrNameToPrerecordedVTable	PK: (nameStringId, languageCode, prerecordedVoiceType, orderOfPreference, prerecordedVoiceId)	Find prerecorded voice for the given name, string ID, language, prerecorded voice type, and order of preference *Most likely there is no need to search by using all the fields comprising the PK but by name string ID only*	PK
PoiPrerecordedVTable poiPrerecordedVTable	PK: (prerecordedVoiceId)	Find prerecorded voice for the for the given ID	PK
TileContentIndexTable tileContentIndexTable	PK: (updateRegionId,buildingBlockId,levelNumber)	Find tile content index for the given update region, building block, and level	PK
PoiPhoneticTraTable poiPhoneticTraTable	PK: (phoneticTranscriptionId, grammaticalCase)	Find phonetic representation for the given ID and grammatical case	PK
TiPhoneticTraTable tiPhoneticTraTable	PK: (phoneticTranscriptionId)	Find the phonetic transcriptions by ID	PK
TiPrerecordedVTable tiPrerecordedVTable	PK: (prerecordedVoiceId)	Find prerecorded voice by ID	PK
NamedObjectPhonemeTable namedObjectPhonemeTable	PK: (namedObjectId)	Find phoneme list for the given named object ID	PK
PoiFtsTable poiFtsTable	FTS3 index	*Not a relational index; SQLite FTS3 module creates a special index for FTS search*	opt
PoiFtsDoubleMetaphTable poiFtsDoubleMetaphTable	FTS3 index	Not a relational index; SQLite FTS3 module creates a special index for FTS search	opt
RoadFtsTable roadFtsTable	FTS3 index	Not a relational index; SQLite FTS3 module creates a special index for FTS search	opt
RoadFtsDoubleMetaphTable roadFtsDoubleMetaphTable	FTS3 index	Not a relational index; SQLite FTS3 module creates a special index for FTS search	opt

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
CrossroadFtsTable crossroadFtsTable	FTS3 index		opt
CrossroadFtsDoubleMetaphTable crossroadFtsDoubleMetaphTable	FTS3 index	Not a relational index; SQLite FTS3 module creates a special index for FTS search	opt
TmcAreaLocationTable tmcAreaLocationTable	PK: (locId)	Find TMC area location by location ID	PK
TmcCrossReferenceTable tmcCrossReferenceTable	PK: (locIdRoadOne, locIdRoadTwo)	*Most likely there is no need to search by using all the fields comprising the PK but by names string id only*	PK
TmcEventCostInfoTable tmcEventCostInfoTable	PK: (eventCode)	Find event cost info by event code	PK
TmcEventNameTable tmcEventNameTable	PK: (eventCode, quantifier, languageCode)	Most likely there is no need to search by using all the fields comprising the PK but by names string id only	PK
	Index on: (eventCode)	Find event name by event code	opt
TmcEventPhoneticTraTable tmcEventPhoneticTraTable	PK: (nameStringId,languageCode,phoneticTraType, orderOfPreference)	Find phonetic transcription by name string ID, language code phonetic transcription type and order of preference	PK
	Index on: (nameStringId,languageCode)	Find phonetic transcription by name string ID and language code	opt
TmcEventPrerecordedVTable tmcEventPrerecordedVTable	PK: (nameStringId,languageCode,prerecordedVoiceType, orderOfPreference,prerecordedVoiceId)	Find prerecorded voice by name string ID, language code prerecorded voice type, order of preference, and prerecorded voice ID	PK
	Index on: (nameStringId,languageCode)	Find prerecorded voice by name string ID and language code	opt
TmcEventTable tmcEventTable	PK: (eventCode)	Find event by event code	PK
TmcExternRefTable tmcExternRefTable	PK: (locId)	Find TMC external references by location ID	PK

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
TmcLocationCoordinateTable tmcLocationCoordinateTable	PK: (locId, seqNumber)	*Most likely there is no need to search by using all the fields comprising the PK but by names string ID only*	PK
	Index on: (locId)	Find TMC location coordinate by location ID	opt
TmcLocationListTable tmcLocationListTable	PK: (locId)	Find TMC location by location ID	PK
TmcLocationPhoneticTraTable tmcLocationPhoneticTraTable	PK: (nameStringId,languageCode,phoneticTraType, orderOfPreference)	Find phonetic transcription by name string ID, language code, phonetic transcription type, and order of preference	PK
	Index on: (nameStringId,languageCode)	Find phonetic transcription by name string id and language code	opt
TmcLocationPrerecordedVTable tmcLocationPrerecordedVTable	PK: (nameStringId,languageCode,prerecordedVoiceType, orderOfPreference,prerecordedVoiceId)	Find prerecorded voice by name string id, language code, prerecorded voice type, order of preference, and prerecorded voice ID	PK
	Index on: (nameStringId,languageCode)	Find prerecorded voice by name string ID and language code	opt
TmcLocationTableIdTable tmcLocationTableIdTable	PK: (locTableId)	Find location table by location table ID	PK
TmcLocationTypeTextTable tmcLocationTypeTextTable	PK: (locCategory, locType, locSubtype, locTableId)	*Most likely there is no need to search by using all the fields comprising the PK but by names string ID only*	PK
	Index on: (locCategory)	Find location type text by location category	opt
TmcNameTable tmcNameTable	PK: (nameId, nameType, languageCode)	*Most likely there is no need to search by using all the fields comprising the PK but by names string ID only*	PK
	Index on: (nameId)	Find location name by name ID	opt
TmcPointLocationTable tmcPointLocationTable	PK: (locId)	Find TMC point location by location ID	PK
TmcSegmentLocationTable tmcSegmentLocationTable	PK: (locId)	Find TMC segment location by location ID	PK

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
TmcStationListTable tmcStationListTable	PK: (id)	Find station by station ID	PK
TmcStationTileListTable tmcStationTileListTable	PK: (id, tileId)	*Most likely there is no need to search by using all the fields comprising the PK but by names string ID only*	PK
	Index on: (id)	Find station tile list by station ID	opt
	Index on: (tileId)	Find stations by tile ID	opt
TmcStringTable tmcStringTable	PK: (stringId)	Find string by string ID	PK
TmcSupplementaryInfoTable tmcSupplementaryInfoTable	PK: (supplementaryInformationCode, languageCode)	*Most likely there is no need to search by using all the fields comprising the PK but by names string ID only*	PK
TmcTileReferenceTable tmcTileReferenceTable	PK: (locId, tileId)	Most likely there is no need to search by using all the fields comprising the PK but by names string ID only	PK
	Index on: (tileId)	Find TMC tile of a location by location ID	opt
TmcTileTable tmcTileTable	PK: (tileId, locTableId)	Most likely there is no need to search by using all the fields comprising the PK but by names string ID only	PK
	Index on: (tileId)	Find TMC location table by tile ID	opt
	Index on: (locTableId)	Find TMC table by tile ID	opt
TpegEventPhoneticTraTable tpegEventPhoneticTraTable	PK: (nameStringId,languageCode,phoneticTraType, orderOfPreference)	Find phonetic transcription by name string ID, language code, phonetic transcription type, and order of preference	PK
	Index on: (nameStringId,languageCode)	Find phonetic transcription by name string ID and language code	opt

Table Schema **Table names**	**PK – default index** **Index on**	**Use of index /** *Comment*	**Type**
TpegEventPrerecordedVTable tpegEventPrerecordedVTable	PK: (nameStringId,languageCode,prerecordedVoiceType, orderOfPreference,prerecordedVoiceId)	Find prerecorded voice by name string ID, language code, prerecorded voice type, order of preference. and prerecorded voice ID	PK
	Index on: (nameStringId,languageCode)	Find prerecorded voice by name string id and language code	opt
TpegTecEventNameTable tpegTecEventNameTable	PK: (eventCode,tableNumber,languageCode,quantifier)	Find TPEG event by event code, table number, language code, and quantifier	PK
TemplateBodyGeometryTable templateBodyGeometryTable	PK: (templateBodyGeometryId)	Find template body geometry for the given ID	PK
RoadGeometry3DTileTable bezierCurveTileTable path3DTileTable jvPath3DTileTable	PK: (id)	Find 3D road geometry for the given ID	PK

C Index

D Index of DataScript Terms