

# **Navigation Data Standard**

## **Format Specification**

**NDS Version 2.1.1.9 and 2.1.1.10.1 - 2.1.1.10.6**  
**(Preliminary Release)**



**Navigation Data Standard**  
PSF Physical Storage Format

## Intellectual Property Rights

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

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

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

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

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

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

Document version 2.2 (based on NDS versions 2.1.1.9 and 2.1.1.10.1 - 2.1.1.10.6)

# Table of Contents

<b>1</b>	<b>About this Document</b>	. . . . .	<b>19</b>
1.1	Document History	. . . . .	19
1.2	Purpose and Subjects	. . . . .	36
1.3	Target Audience	. . . . .	38
1.4	NDS Documentation Overview	. . . . .	38
1.5	Use of Modal Verbs	. . . . .	39
1.6	Typographical Conventions	. . . . .	40
<b>2</b>	<b>Introduction to Navigation Data Standard</b>	. . . . .	<b>41</b>
<b>3</b>	<b>Architecture</b>	. . . . .	<b>43</b>
3.1	NDS Database and Its Interfaces	. . . . .	43
3.2	NDS Database Structure	. . . . .	44
3.2.1	Product Database	. . . . .	45
3.2.2	Update Region	. . . . .	45
3.2.3	Building Block	. . . . .	46
3.2.4	Levels	. . . . .	47
3.3	Database Content	. . . . .	49
3.3.1	Features	. . . . .	49
3.3.2	Attributes	. . . . .	52
3.3.3	Metadata	. . . . .	53
3.4	Introduction to NDS DataScript	. . . . .	53
3.5	Extending the Navigation Data Standard	. . . . .	55
<b>4</b>	<b>Database Structure</b>	. . . . .	<b>59</b>
4.1	Product Database Table	. . . . .	59
4.2	NDS Database Supplier Table	. . . . .	61
4.3	Data Model Version Table	. . . . .	62
4.4	Overall Metadata Table	. . . . .	62
4.5	Structure, Data, and Metadata of Product Databases	. . . . .	62
4.5.1	Update Region Table	. . . . .	65
4.5.2	Building Block Table	. . . . .	65
4.5.3	Update Region Building Block Version Table	. . . . .	66
4.5.4	Version Table	. . . . .	67
4.5.5	Building Block Component Version Table	. . . . .	68
4.5.6	Language Table	. . . . .	70
4.6	Logical Mapping of Tables to Building Blocks	. . . . .	72
<b>5</b>	<b>Overall Building Block: Update Region Metadata</b>	. . . . .	<b>73</b>
5.1	Level Metadata	. . . . .	73

5.2	Region Metadata . . . . .	74
5.2.1	Flexible Attribute Maps . . . . .	76
5.3	Region-specific Descriptive Names of Selection Criteria . . . . .	76
5.4	Road Number Class Prefix Table . . . . .	77
5.5	Additional Icons . . . . .	78
5.6	Address Format . . . . .	79
5.7	Region-specific Profiles . . . . .	79
5.8	Time Zones . . . . .	79
<b>6</b>	<b>Attributes . . . . .</b>	<b>81</b>
6.1	Attribute Groups . . . . .	81
6.2	Attribute Layers . . . . .	82
6.3	Attribute Points and Shape Points . . . . .	83
6.4	References between Attributes and Features . . . . .	85
6.5	Common Attributes . . . . .	86
<b>7</b>	<b>Partitioning of Geographic Data . . . . .</b>	<b>89</b>
7.1	WGS 84 and EGM96 . . . . .	89
7.2	Coding of Coordinates . . . . .	89
7.3	Tiling Scheme . . . . .	92
7.3.1	Tiles and Levels . . . . .	93
7.3.2	Tiles and Clipping . . . . .	96
7.3.3	Content Indices for Update Regions . . . . .	99
<b>8</b>	<b>Encryption . . . . .</b>	<b>101</b>
8.1	Scope of Encryption . . . . .	101
8.1.1	Database File Encryption . . . . .	102
8.1.2	Encryption of Data BLOBs . . . . .	103
8.2	Cipher Algorithm . . . . .	103
8.3	Cipher Key Info Structure . . . . .	104
8.4	Encryptable Item Code . . . . .	105
8.5	Decryption . . . . .	105
<b>9</b>	<b>Routing Building Block . . . . .</b>	<b>107</b>
9.1	Building Block Structure and Content . . . . .	108
9.2	Link Feature Class . . . . .	113
9.2.1	Base Links . . . . .	114
9.2.2	Route Links . . . . .	115
9.2.3	Attributes of Link Features . . . . .	117
9.2.4	Road Geometry for Link Features . . . . .	120
9.2.5	References of Link Features . . . . .	120
9.2.6	References of Route Links . . . . .	122
9.3	Road Geometry Line Feature Class . . . . .	122

---

9.3.1	Attributes of Road Geometry Line Features . . . . .	123
9.3.2	Drawing Order of Road Geometry Line Features . . . . .	125
9.3.3	References of Road Geometry Line Features . . . . .	125
9.4	Intersection Feature Class . . . . .	126
9.4.1	Position . . . . .	127
9.4.2	Transitions . . . . .	127
9.4.3	Number of Links . . . . .	127
9.4.4	Attributes of Intersection Features . . . . .	127
9.4.5	References of Intersection Features . . . . .	128
9.5	References of Routing Features . . . . .	128
9.5.1	Inter-Tile References . . . . .	128
9.5.2	Inter-Building Block References . . . . .	129
9.5.3	Inter-Level References . . . . .	132
9.6	Data Structures for Advanced Driver Assistance (ADAS) . . . . .	134
9.6.1	Attributes for ADAS . . . . .	134
9.6.2	Enhanced ADAS Geometry . . . . .	137
9.6.3	Enhanced ADAS Attributes for Elevation Profiles . . . . .	143
9.6.4	Enhanced ADAS Geometry Information for Intersection Features . . . . .	145
9.7	Generic Profile Attributes . . . . .	145
9.7.1	Flexible Attributes for Speed Profiles . . . . .	146
9.7.2	Example . . . . .	147
9.7.3	Discrete Cosine Transformation . . . . .	149
9.8	Address Points . . . . .	151
9.9	Data Structures for Eco Routing . . . . .	151
9.9.1	Eco Routing Metadata . . . . .	153
9.9.2	Consumption Speed Curve . . . . .	154
9.9.3	Speed Variation . . . . .	157
9.9.4	Slope . . . . .	159
<b>10</b>	<b>Name Building Block . . . . .</b>	<b>163</b>
10.1	Building Block Structure and Content . . . . .	164
10.2	Named Object Features . . . . .	167
10.2.1	Attributes of Named Object Features . . . . .	171
10.2.2	Relations between Named Object Features . . . . .	173
10.2.3	References to Next-valid-character Trees . . . . .	176
10.3	Name String Features . . . . .	177
10.3.1	Attributes of Name String Features . . . . .	177
10.4	Selection Graphs . . . . .	182
10.5	Selection Criteria . . . . .	184
10.6	Next-valid-character Trees . . . . .	186
10.6.1	Selection Graphs and NVC Trees . . . . .	187
10.6.2	Structure of NVC Trees . . . . .	189
10.6.3	Disambiguation in NVC Trees . . . . .	192

---

---

10.6.4	Languages in NVC Trees . . . . .	193
10.6.5	Mixed-case Characters in NVC Trees . . . . .	194
10.6.6	Extended Postal Codes in NVC Trees . . . . .	196
10.7	Address Retrieval . . . . .	196
10.8	References of Named Object Features. . . . .	198
10.9	Address Points . . . . .	199
<b>11</b>	<b>Basic Map Display Building Block . . . . .</b>	<b>201</b>
11.1	Building Block Structure and Content . . . . .	202
11.2	Overview of Basic Map Display Features . . . . .	207
11.3	Point Features . . . . .	208
11.4	Line Features . . . . .	209
11.5	Area Features . . . . .	211
11.6	Icons . . . . .	213
11.7	Drawing Styles . . . . .	213
11.8	Tile Patterns . . . . .	214
11.9	Drawing Order . . . . .	215
11.10	Using Z Levels for Map Rendering . . . . .	216
11.10.1	Advantages of Z Levels . . . . .	216
11.10.2	Data Structures for Z Level Information . . . . .	218
11.11	2.5D City Models . . . . .	223
11.11.1	Flexible Attributes for 2.5D City Models . . . . .	224
11.12	3D Objects Representing Building Footprints . . . . .	226
11.13	Labeling Hints . . . . .	226
11.13.1	Labeling of Line Features . . . . .	227
11.13.2	Labeling of Area Features . . . . .	227
<b>12</b>	<b>Traffic Information Building Block . . . . .</b>	<b>229</b>
12.1	Building Block Structure and Content . . . . .	230
12.2	Traffic Message Channel (TMC) – Introduction to Concept and Structure . . . . .	232
12.2.1	TMC Event Information . . . . .	232
12.2.2	TMC Location Information . . . . .	233
12.2.3	Determining TMC Locations for a Traffic Event . . . . .	235
12.2.4	Hierarchical Structure of TMC Locations . . . . .	236
12.2.5	Linking TMC Information to NDS Features . . . . .	237
12.2.6	TMC Data Structures. . . . .	238
12.3	Transport Protocol Experts Group (TPEG) . . . . .	249
12.3.1	TPEG Event Information . . . . .	249
12.3.2	TPEG Location Information . . . . .	251
12.3.3	TPEG Data Design. . . . .	251
12.4	Vehicle Information and Communication System (VICS). . . . .	252
<b>13</b>	<b>POI Building Block . . . . .</b>	<b>253</b>

---

---

13.1	Building Block Structure and Content . . . . .	253
13.2	POI Feature Class . . . . .	259
13.2.1	Primary POI Attributes . . . . .	259
13.2.2	Secondary POI Attributes . . . . .	260
13.3	Relations between POIs . . . . .	263
13.3.1	Part-of Relations between POIs . . . . .	264
13.3.2	Access Relations between POIs . . . . .	265
13.3.3	Logical-access Relations between POIs . . . . .	265
13.3.4	Guidance-point Relations between POIs . . . . .	266
13.3.5	Activating POIs along a Route . . . . .	266
13.4	POI Categories . . . . .	267
13.4.1	Attributes of POI Categories . . . . .	269
13.5	Representing POIs as Icons . . . . .	270
13.5.1	Drawing Icons for POIs . . . . .	271
13.5.2	Overlapping POI Icons in Map Display . . . . .	271
13.6	Accessing POIs for Location Input and Map Display . . . . .	272
13.6.1	POI Service Locations . . . . .	273
13.6.2	Relations between POI and Regions . . . . .	273
13.6.3	Virtual Tiles for POIs . . . . .	277
13.6.4	Next-valid Character Trees for POIs . . . . .	278
13.7	Versioning of POIs . . . . .	279
<b>14</b>	<b>Speech Building Block . . . . .</b>	<b>281</b>
14.1	Building Block Structure and Content . . . . .	282
14.2	Grammatical Structures for Phonetic Transcriptions . . . . .	286
14.3	Optimization Structures for Phonetic Transcriptions . . . . .	287
<b>15</b>	<b>Digital Terrain Model Building Block . . . . .</b>	<b>289</b>
15.1	Height Maps . . . . .	290
15.1.1	Structure and Content of Height Maps . . . . .	290
15.1.2	Partitioning of Height Information into Tiles and Levels . . . . .	291
15.1.3	Content of a Single DTM Tile . . . . .	292
15.1.4	Compression for Height Maps . . . . .	294
15.2	Batched Dynamic Adaptive Meshes (BDAM) . . . . .	294
15.2.1	Assigning BDAM Patches to NDS Tiles . . . . .	296
15.2.2	Building Block Structure and Content . . . . .	302
<b>16</b>	<b>Orthoimages Building Block . . . . .</b>	<b>305</b>
16.1	Tiling Scheme for Orthoimages . . . . .	306
16.2	Building Block Structure and Content . . . . .	306
<b>17</b>	<b>3D Objects Building Block . . . . .</b>	<b>309</b>
17.1	Building Block Structure and Content . . . . .	309

---

17.2	3D Object Feature . . . . .	312
17.2.1	Properties of 3D Object Features . . . . .	313
17.2.2	Bounding Box . . . . .	315
17.2.3	Object Hierarchy Level . . . . .	319
17.2.4	Feature Classes for 3D Objects . . . . .	320
17.2.5	Identifying 3D Objects . . . . .	320
17.2.6	References . . . . .	322
17.3	Geometry Information for 3D Objects . . . . .	322
17.3.1	Elevation Information for 3D Objects . . . . .	324
17.3.2	Level of Detail for 3D Objects . . . . .	325
17.3.3	Body Geometry Type . . . . .	326
17.3.4	Template Body Geometry . . . . .	327
17.3.5	Render Groups . . . . .	328
17.3.6	Using the 3D Objects Data Structures . . . . .	330
17.4	Material and Textures . . . . .	331
17.5	References to Geometry Information . . . . .	335
17.6	References to Name Building Block . . . . .	336
17.7	3D Objects and Building Footprints . . . . .	336
17.8	References to 3D Objects . . . . .	336
17.9	Landmark Icons . . . . .	340
<b>18</b>	<b>Junction View Building Block . . . . .</b>	<b>341</b>
18.1	Building Block Structure and Content . . . . .	342
<b>19</b>	<b>Full-text Search Building Block . . . . .</b>	<b>345</b>
19.1	Building Block Structure and Content . . . . .	346
19.2	Examples for Search Documents . . . . .	348
19.3	Use of the Full-text Search Building Block . . . . .	352
<b>20</b>	<b>Icons . . . . .</b>	<b>355</b>
20.1	Icon Sets . . . . .	356
20.2	Icon Collections . . . . .	356
20.3	Default Icons and Application-specific Icons . . . . .	357
20.4	Icon Sprites . . . . .	357
20.5	Additional Icons for Banned Routes . . . . .	357
20.6	3D Icons . . . . .	358
<b>A</b>	<b>Glossary . . . . .</b>	<b>359</b>
<b>B</b>	<b>NDS Database Supplier IDs . . . . .</b>	<b>369</b>
<b>C</b>	<b>Overview of Database Tables . . . . .</b>	<b>371</b>
<b>D</b>	<b>List of Flexible Attributes . . . . .</b>	<b>381</b>

---

## List of Figures

3-1	NDS-compliant database and its interfaces . . . . .	44
3-2	Building blocks in NDS and their data . . . . .	47
3-3	Database structure, levels. . . . .	48
3-4	Examples of feature classes . . . . .	50
4-1	Entry point to an NDS database. . . . .	59
4-2	Table structure for a product database, example. . . . .	63
6-1	Line features and associated shape points . . . . .	85
7-1	Tiling scheme for partitioning the Earth's surface: Level 0. . . . .	94
7-2	Tiling scheme for partitioning the Earth's surface: Level 1. . . . .	94
7-3	Tiling scheme for partitioning the Earth's surface: Level 2. . . . .	95
7-4	Tile anchor point definition . . . . .	96
7-5	Features on one tile. . . . .	97
7-6	Features extending over more than one tile . . . . .	98
9-1	Routing building block structure . . . . .	108
9-2	Logical diagram of routing features . . . . .	113
9-3	Link between two intersections. . . . .	113
9-4	Links located on one tile only and links extending over more than one tile . . . . .	115
9-5	Route links on level 13 of the Routing building block . . . . .	116
9-6	Examples of plural junctions and intersections . . . . .	119
9-7	References of routing and basic map display features . . . . .	131
9-8	Upward and downward references between links on different levels . . . . .	132
9-9	Different kinds of interpolation for profiles (profile view) . . . . .	135
9-10	Road elevation profile (grade line) given by a piecewise linear grade profile (profile view) . . . . .	136
9-11	Piecewise linear profile corresponding to the grade line in Figure 6-8 (profile view) . . . . .	137
9-12	Two road center lines with marked chainages (plan view). . . . .	138
9-13	Clothoids and shape points with differing geometry (plan view). . . . .	139
9-14	Example of shape points and related clothoid in different tiles (plan view) . . . . .	142
9-15	Part of a grade line with a sag vertical curve (profile view) . . . . .	143

---

9-16	Grade line based on values from Table 6-10 (profile view) . . . . .	144
9-17	Enhanced geometry information for intersection features . . . . .	145
9-18	Speed profile, example . . . . .	148
9-19	Traffic patterns in array representations vs. DCT reconstructions, example 1 .	150
9-20	Traffic patterns in array representations vs. DCT reconstructions, example 2 .	150
9-21	Fuel/energy consumption effects - Overview . . . . .	153
9-22	Typical CSC representing a plurality of vehicles. . . . .	154
9-23	Statistical CSC and variances . . . . .	156
9-24	Simplified curve for dependency between slope and consumption for combustion engine vehicles . . . . .	160
10-1	Named object feature classes. . . . .	168
10-2	Contained-in relation between named object features . . . . .	175
10-3	Selection graph, example . . . . .	183
10-4	Predefined selection criteria, overview . . . . .	184
10-5	Example: Mapping between selection criteria and named object feature classes	186
10-6	Next-valid-character tree, example . . . . .	187
10-7	Selection graphs, example . . . . .	188
10-8	Selection graphs and NVC trees, example . . . . .	188
10-9	Next-valid-character tree, terms. . . . .	189
10-10	Partitioning of NVC trees, example . . . . .	190
10-11	Disambiguation, example 1 . . . . .	192
10-12	Disambiguation, example 2 . . . . .	192
10-13	Disambiguation, example 3 . . . . .	193
10-14	Name hierarchy for address retrieval, roads and grouping . . . . .	197
10-15	Name hierarchy for address retrieval, roads only . . . . .	198
10-16	Types of address points. . . . .	200
11-1	Shared level between Basic Map Display and Routing building blocks . . . .	202
11-2	Element display in scale sublevels . . . . .	205
11-3	Overview of basic map display features . . . . .	207
11-4	Point feature classes, example . . . . .	208
11-5	Line feature classes, example. . . . .	210
11-6	Area feature classes, example . . . . .	211

---

---

11-7	Polygon types . . . . .	212
11-8	Tile pattern, example . . . . .	215
11-9	Drawing order, example 1. . . . .	216
11-10	Drawing order, example 2. . . . .	216
11-11	Map rendering without Z levels. . . . .	217
11-12	Map rendering with Z levels . . . . .	217
11-13	Example 1: Link without Z level attribute . . . . .	218
11-14	Example 2: Link with Z level = 1 for the complete range . . . . .	219
11-15	Example 3a: Link with constant Z level . . . . .	219
11-16	Example 3b: Link with constant Z level . . . . .	220
11-17	Example 4: Complete link is a ramp . . . . .	220
11-18	Example 5: Link has a start and an end Z level range. . . . .	221
11-19	Example 6: Link ends with a ramp . . . . .	221
11-20	Example 7: Link with defined start and end Z level range and ramp in between . . . . .	222
11-21	Road across a river rendered by means of detached from terrain attribute . . . . .	223
11-22	2.5D city model, example . . . . .	224
11-23	Composed 2.5D representation of a building . . . . .	226
11-24	Labeling area features with horizontal text . . . . .	228
12-1	Sequence of point locations at intersections . . . . .	234
12-2	Location codes along roads with shared road sections . . . . .	234
12-3	Changes in the location code sequence . . . . .	235
12-4	Direction and extent of traffic events, example . . . . .	236
12-5	TMC location hierarchy . . . . .	237
12-6	Schema for referencing TMC locations to links in an NDS database. . . . .	238
12-7	Relational database model for TMC event information . . . . .	240
12-8	Relational database model for TMC location information . . . . .	242
12-9	Identifying direction information for links . . . . .	247
12-10	Example of locations in different tiles. . . . .	248
12-11	Lookup table for TEC event information in NDS . . . . .	250
13-1	Integrated and non-integrated POI building block . . . . .	255
13-2	Relations of POIs at an airport . . . . .	264
13-3	Logical access points and POIs on a freeway . . . . .	266

---

---

13-4	Directed acyclic graph of categories and POIs . . . . .	268
13-5	Example for overlapping icons and an icon representing an icon stack. . . . .	272
13-6	Relations between administrative and named areas . . . . .	274
13-7	Administrative hierarchy for the example in Figure 11-5 . . . . .	274
13-8	Vanity POI . . . . .	277
15-1	Height map as 3D diagram in Excel, example . . . . .	293
15-2	BDAM patch surface covered by a TIN mesh . . . . .	295
15-3	Splitting rules for BDAM patches . . . . .	296
15-4	Continuous and discrete level of detail (BDAM) . . . . .	298
15-5	NDS levels and BDAM levels of detail . . . . .	299
15-6	Recursive splitting of patches in different BDAM levels . . . . .	300
15-7	Mixing BDAM levels of detail . . . . .	301
15-8	Mixing BDAM level of details, error levels . . . . .	301
16-1	Orthoimages, examples. . . . .	305
16-2	Background bitmap with varying ground resolution . . . . .	306
17-1	Structure of the 3D Objects building block . . . . .	310
17-2	Spatial tree with inner nodes and leaf nodes . . . . .	313
17-3	Bounding Box . . . . .	316
17-4	Bounding box for 3D objects . . . . .	317
17-5	Rotated bounding boxes for 3D objects . . . . .	318
17-6	Use cases for rotated bounding boxes. . . . .	318
17-7	Use of object hierarchy level . . . . .	319
17-8	Examples for 3D object feature classes . . . . .	320
17-9	Example for Identifying 3D Objects . . . . .	321
17-10	Normal sphere. . . . .	324
17-11	Elevation and scaling factor – Example . . . . .	325
17-12	Render groups . . . . .	328
17-13	Connecting render groups by adding degenerated triangles . . . . .	329
17-14	Architecture of 3D Objects building block . . . . .	330
17-15	Texture types . . . . .	332
17-16	Occlusion . . . . .	333
17-17	Texture layers . . . . .	334

---

---

17-18	Example: Inner node with four LOD references . . . . .	335
17-19	Spatial node without LOD 0 reference . . . . .	336
17-20	3D object references by means of coordinates and object hierarchy level information . . . . .	338
17-21	3D object references by means of coordinate information only . . . . .	338
17-22	3D object references by means of coordinate and distance to center point . . . . .	339
17-23	3D object references – delta vector . . . . .	340
18-1	Junction view image. . . . .	341
18-2	3D junction view . . . . .	341
18-3	Day view (left) and night view (right), example . . . . .	343
19-1	Structure of the Full Text Search building block . . . . .	346
C-1	Product data . . . . .	371
C-2	Overall data . . . . .	371
C-3	3D Objects building block . . . . .	372
C-4	Basic Map Display building block . . . . .	372
C-5	Digital Terrain Model building block . . . . .	373
C-6	Full-text Search building block . . . . .	374
C-7	Junction View building block. . . . .	375
C-8	Name building block . . . . .	375
C-9	Orthoimages building block . . . . .	376
C-10	POI building block . . . . .	376
C-11	Routing building block . . . . .	377
C-12	Speech building block . . . . .	378
C-13	Traffic Information building block . . . . .	379
D-1	Flexible attributes in NDS, part 1 . . . . .	382
D-2	Flexible attributes in NDS, part 2 . . . . .	383
D-3	Flexible attributes in NDS, part 3 . . . . .	384
D-4	Flexible attributes in NDS, part 4 . . . . .	385
D-5	Flexible attributes in NDS, part 5 . . . . .	386
D-6	Flexible attributes in NDS, part 6 . . . . .	387
D-7	Flexible attributes in NDS, part 7 . . . . .	388
D-8	Flexible attributes in NDS, part 8 . . . . .	389

---

## List of Tables

1-1	Rules for use of modal verbs . . . . .	39
3-1	Components of the feature ID . . . . .	51
4-1	Fields of the product database table . . . . .	60
4-2	Fields of the NDS database supplier table . . . . .	61
4-3	Tables containing metadata for a product database . . . . .	63
4-4	Fields of the update region table . . . . .	65
4-5	Fields of the building block table . . . . .	66
4-6	Structure of the update region building block version table . . . . .	66
4-7	Fields of the version table . . . . .	67
4-8	Fields of the building block component version table . . . . .	69
4-9	Language definition table, example. . . . .	70
5-1	Level metadata . . . . .	73
5-2	Region metadata . . . . .	74
5-3	Data in the RegionSpecificDescriptiveNameTable. . . . .	76
5-4	Data in the RoadNumberClassPrefixTable . . . . .	77
5-5	Road number class prefix table, example. . . . .	78
5-6	Data for additional icons . . . . .	78
5-7	Time zone data . . . . .	80
6-1	Attributes for data quality . . . . .	86
6-2	Flexible attributes specifying time periods . . . . .	87
6-3	Flexible attributes shared between Routing and Basic Map Display building blocks. . . . .	87
7-1	Integer coding of WGS 84 coordinates given in floating point values, examples. .	90
7-2	Morton codes derived from the values given in part 1 of the example . . . . .	92
7-3	Tile numbers in level 10 and level 13 . . . . .	96
8-1	Encryption indicator in the building block version table . . . . .	102
8-2	Fields of the cipher key info structure . . . . .	104
8-3	Fields of the encryptable item code table . . . . .	105
9-1	Data in Routing building block . . . . .	109
9-2	Fixed attributes of link features . . . . .	117

---

9-3	Differences between plural junction and intersection . . . . .	119
9-4	References from links to named objects . . . . .	121
9-5	Fixed attributes of road geometry line features . . . . .	123
9-6	References from road geometry lines to named objects . . . . .	126
9-7	Inter-tile references for routing features. . . . .	129
9-8	References from routing features to named object features . . . . .	129
9-9	Logical structure of the clothoid data (from Figure 6-18) . . . . .	140
9-10	Set of data obtained by combining clothoid data and shape point geometry (from Figure 6-11) . . . . .	141
9-11	Example of mapped values (elevation profiles) . . . . .	144
9-12	Eco routing metadata . . . . .	153
10-1	Data in Name building block . . . . .	164
10-2	Metadata of the Name building block. . . . .	165
10-3	Properties of named object feature classes. . . . .	170
10-4	Relations between named object features . . . . .	173
10-5	Use types for name strings . . . . .	178
10-6	Name formats. . . . .	179
10-7	Relation type attribute for name strings . . . . .	181
10-8	Properties of selection criteria . . . . .	185
10-9	Data structures for NVC tree node types. . . . .	190
10-10	Language information in NVC trees . . . . .	193
10-11	References from the Basic Map Display to the Name building block . . . . .	198
10-12	References from the Routing to the Name building block . . . . .	199
11-1	Data in Basic Map Display building block. . . . .	202
11-2	Example: Assignment of max. map scale denominators to building block levels .	203
11-3	Metadata of the Basic Map Display building block . . . . .	205
11-4	Fixed attributes of point features . . . . .	209
11-5	Fixed attributes of line features. . . . .	210
11-6	Fixed attributes of area features . . . . .	212
11-7	Flexible attributes for 2.5D city models . . . . .	224
11-8	Attribute values for the buildings in Figure 8-26 . . . . .	226
12-1	Data in Traffic Information building block . . . . .	230

---

---

12-2	Lookup tables for event information, overview . . . . .	240
12-3	Lookup tables for location information, overview . . . . .	243
12-4	Elements of the flexible attribute <b>LocalTmcLocation</b> . . . . .	246
12-5	Local attribute list of TMC locations . . . . .	248
12-6	Lookup table for event information, overview . . . . .	251
13-1	Data in POI building block . . . . .	255
13-2	Metadata of the POI building block. . . . .	256
13-3	Primary attributes of POIs . . . . .	259
13-4	Examples for predefined secondary POI attributes . . . . .	261
13-5	Examples for multimedia secondary POI attributes . . . . .	262
13-6	<b>PoiCatRelationTable</b> table . . . . .	268
13-7	Attributes for POI categories . . . . .	269
13-8	Data structures for POI NVC trees . . . . .	278
13-9	Versioning information for POIs . . . . .	279
14-1	Data in Speech building block. . . . .	283
15-1	DTM image data . . . . .	290
15-2	Height map data . . . . .	290
15-3	Examples for height resolutions in NDS tiling scheme . . . . .	291
15-4	Indexing for black and white tiles . . . . .	296
15-5	NDS levels and BDAM levels of detail . . . . .	299
15-6	Metadata for BDAM . . . . .	302
17-1	Data in 3D Objects building block . . . . .	310
17-2	Metadata of the 3D Objects building block . . . . .	311
17-3	Properties of 3D objects stored in spatial nodes . . . . .	313
17-4	Object hierarchy level – Examples . . . . .	319
17-5	Data stored in the body geometry BLOB . . . . .	322
17-6	Body geometry data . . . . .	323
17-7	Level of detail for 3D objects . . . . .	325
17-8	Properties of the texture map table . . . . .	333
18-1	Attributes for junction view images. . . . .	343
19-1	Columns of the full-text search virtual table for POI features . . . . .	349
19-2	Columns of the full-text search virtual table for road named objects . . . . .	350

---





# 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
0.1	2008-06-06	
0.8	2008-06-09	
0.9	2008-06-24	<p>Concepts:</p> <ul style="list-style-type: none"> <li>- Numbering of tiles corrected</li> </ul> <p>Routing:</p> <ul style="list-style-type: none"> <li>- Concept of clusters removed</li> </ul> <p>Names:</p> <ul style="list-style-type: none"> <li>- Named object feature class definition changed</li> <li>- Selection graph definition changed</li> <li>- Metadata description for named object feature classes inserted</li> </ul> <p>Basic Map Display</p> <ul style="list-style-type: none"> <li>- Convex polygon for representing areas removed</li> <li>- New metadata to define hierarchy of Map features</li> <li>- Support of tile patterns introduced</li> <li>- Number of inhabitants and administrative level now flexible attributes for point features instead of fixed attributes</li> </ul>
1.0	2008-07-21	Review comments included after review cycle within NDS Working Group
1.1	2009-01-21	<p>TMC added as building block</p> <p>Encryption added as building block</p> <p>Information on global metadata added</p> <p>Changes to Name building block</p> <p>Smaller changes added to POI building block</p>

Version number	Published	Changes
1.2	2009-04-24	<p>Chapter <i>Extended Datascript Grammar</i> added</p> <p><i>Routing Building Block</i> chapter: Admin road classes and upward/downward references changed</p> <p>Several changes/enhancements made in the <i>Name Building Block</i> chapter:</p> <ul style="list-style-type: none"> <li>- Translation -&gt; Transliteration</li> <li>- Mixed case characters</li> <li>- Character code lists</li> </ul> <p>Added Section <i>Introduction to DataScript</i></p> <p>Added Section <i>Extending the Navigation Data Standard</i></p> <p>Added Chapter <i>Phonetic Representation Building Block</i></p> <p>Added Chapter <i>POI Building Block</i></p> <p>Changes in <i>Encryption</i> chapter</p> <p>Added Section on Z levels to Chapter <i>Basic Map Display Building Block</i></p>

Version number	Published	Changes
1.3	2009-07-10	<p>New: Section 7.9 <i>Metadata</i> (Routing building block)</p> <p>New: Chapter 14 <i>Icons</i></p> <p>New: Section <i>Metadata</i> in <i>Basic Map Display Building Block</i> chapter</p> <p>Renamed DataScript references from <code>psi</code> to <code>nds</code> according to changes in DataScript</p> <p>Moved several sections from <i>Name Building Block</i> chapter to Section 5.2 <i>Global Metadata</i> on page 46 (incl. necessary changes to <i>Name Building Block</i> chapter):</p> <ul style="list-style-type: none"> <li>- Language definition</li> <li>- Rotation Metadata</li> <li>- Max. length for name strings</li> </ul> <p>Added references to Chapter 11 <i>Speech Building Block</i> to Routing, POI, Name, and Traffic Information building block chapters</p> <p>[Bug 379] Added DataScript location and reference to <i>NDS – Compiler Interoperability Specification</i>, Section <i>Levels</i></p> <p>Added small changes to <i>POI Building Block</i> chapter</p> <p>Replaced “update region” with “product” in <i>Encryption</i> chapter</p> <p>Replaced “House Number” with “Named object” in Figure 10-1</p> <p>Added new chapter <i>Profile Attributes</i></p> <p>Added new section <i>Time Zone Information</i></p> <p>Added sentence “For valid continuations, the number of remaining matches is given” in NVC section of the Name Building Block chapter</p> <p>Added new section <i>Languages in NVC Trees</i> to NVC tree section in Name Building Block chapter</p> <p>Added new Chapter 16 <i>Orthoimages Building Block</i> on page 305</p> <p>Added examples for Morton Codes to Chapter 4 <i>Partitioning of Geographic Data</i></p>

Version number	Published	Changes
1.4	2009-10-19	<p>[Bug 418] Fixed wrong arrow in Figure 11-3</p> <p>Corrected wrong numbers in Figure 7-1</p> <p>[Bug432] Updated metadata information (global &amp; BB-specific)</p> <p>[Bug 412] Introduced product DB concept (new section 3.2.1 and updated section 3.2.2)</p> <p>[Bug 417] Changed example in Profile Attributes chapter according to review comments</p> <p>[Bug 422] Changed update region definition</p> <p>[Restructuring] Split <i>Signpost and Lanes</i> section into two separate sections: Lane Attributes and Signpost Attributes</p> <p>[Restructuring] Incorporated <i>Profile Attributes</i> chapter in Routing Building Block chapter</p> <p>[New] Added new sections <i>Warning Sign Attributes</i> and <i>ADAS Attributes</i> to Routing Building Block chapter</p> <p>[New] Added new chapter <i>Full-text Search Building Block</i></p> <p>[New] Added new section <i>2.5D City Models</i> to Basic Map Display Building Block chapter</p> <p>[Bug 410] Added information on rotation separation string</p> <p>[Bug 411] Set escape byte to standard value</p> <p>[Bug 406] Added information on min scale denominator for POIs</p> <p>Enhanced section 6.7.1 <i>Filtering of Links by Road Classes</i></p> <p>[Bug 384] Renamed psi/psf to nds</p> <p>[Bug 415] Added information on storing icons as PNG bitmaps in several resolutions</p> <p>[New] Added section <i>Typographical Conventions</i></p> <p>[Restructuring] Added new chapter <i>Common Data</i> for global metadata, common attributes, and icons; moved content of <i>Content of Databases</i> chapter to <i>Architecture</i> and <i>Common Data</i> chapters</p> <p>[Restructuring] Replaced mandatory and optional attributes with concept of fixed and flexible attributes</p> <p>[Docu] Cross-references added to Update and Compiler Spec documents, some standardization issues (writing, terminology, standard formulations)</p> <p>[New] Added concept for attribute layers and attribute points to <i>Architecture</i> and <i>Routing Building Block</i> chapters</p> <p>[New] Added ADAS information to <i>Routing Building Block</i> chapter</p> <p>[Changed] Added some clarifications and enhancements to <i>POI Building Block</i> chapter</p> <p>[Bug 415] Added <i>UsageTypeMask</i> in <i>Icon</i> section</p> <p>[Bug 397] Added <i>PoiCatToAttrTable</i> to Table 10-2</p>

Version number	Published	Changes
1.4 (continued)		<p>[Bug 431] Added info on icon collection, 2D, and 3D to <i>Icon</i> section</p> <p>[Bug 493] Added info that the assignment of point location to links is not stored</p> <p>[Bug 399] Added info about link direction</p> <p>[Bug 465] Added flexible attribute UNDER_CONSTRUCTION to Table 16-2 <i>Flexible attributes shared between BMD and Routing</i></p> <p>[Bug 472] Added new section <i>Currency</i> to <i>Common Attributes</i> section</p> <p>[Bug 507] Added sentence “References to feature IDs refer to the position of the respective feature in the feature list.” to <i>Referencing Features</i> section</p> <p>[Bug 526] Added access point as examples of POI standard categories</p> <p>[Bug 386] Moved info about signpost names to <i>Name Building Block</i> chapter</p> <p>Added info about references from junction view images and TMC to route links and base links in <i>Routing Building Block</i> chapter</p> <p>Added info about dependencies between junction views and routing features to <i>Routing Building Block</i> chapter</p> <p>[Bug 479] Added info to <i>Area Location Data</i> section in <i>Traffic Information Building Block</i> chapter</p> <p>Changed section about references from routing features to named object features</p> <p>[Bug 542] Changed info about use of ISO country codes in <i>Language Definition</i> section</p> <p>[Bug 522] Changed <i>References to Next-Valid-Character Trees</i> section</p> <p>Made several changes based on review comments in the second document review</p>

Version number	Published	Changes
1.5 (for NDS 2.1)	2010-01-20	<p>Added documentation changes from task force modifications to <i>Name Building Block</i> chapter</p> <p>Added docu changes from task force modifications to <i>Routing Building Block</i> chapter</p> <p>[Bug 501] Added info about defining methods to relate exonyms and alternate spellings with their original names to Table <i>Relation type attribute for name strings</i> ( EXONYM and ALTERNATE SPELLING )</p> <p>[Bug 535] Added info about flexible attribute KM_MI_INIDICATOR to Table 16-7</p> <p>[Bug 534] Added info about use of attribute UNDER_CONSTRUCTIONto Table 16-3</p> <p>[Bug 601] Moved <i>Optimization</i> section of <i>Digital Terrain Model Building Block</i> chapter to Compiler Interoperability Specification</p> <p>[Bug 619] Changed <i>Drawing Order</i> section: „Moreover, features of the same class can be sorted by decreasing importance...“ changed to „... increasing importance“</p> <p>Added postal code to Figure 7-2 <i>Contained-in relation</i></p> <p>[Bug 604]Added <i>Point Addresses</i> section to <i>Routing Building Block</i> chapter</p> <p>[Bug 543] Deleted value STUB_ROAD from Table 16-1 <i>Values for data quality</i></p> <p>[Bug 605] Corrected quotation marks</p> <p>[Bug 615] Added <i>Use of Modal Verbs</i> section to <i>About this Document</i> chapter</p> <p>[Bug 619] Changed <i>Drawing Order</i> section according to proposal in Bug 619</p> <p>Use of online/compiletime/runtime harmonized</p> <p>[Bug 592] Corrected error in Figure 6-6</p> <p>[Bug 614] Added <i>List of NDS Database Supplier IDs</i> appendix and cross-references to it in Format Spec, Table 3-1, Compiler Interop Spec, Table 2-2, and Update Spec, Table 4-1</p> <p>[Bug508] Renamed IS_CAPITAL to IS_COUNTRY_CAPITAL</p> <p>[Bug533] Changed description of YEAR</p> <p>[Bug 636] Updated copyright page and related changes to docs due to transition to NDS e.V.</p> <p>[Bug530] Changed attribute ROAD_WIDTH to PHYSICAL_WIDTH_METRICAL and ROAD_WIDTH_IMPERIAL in Section 6.9 and Table 16-3.</p> <p>[Bug 598] Added info about reversed terms to Table 15-1 in the <i>Full-text Search Building Block</i> chapter</p> <p>[Bug 596] Added relation type PREFERRED to Section 10-3</p>

Version number	Published	Changes
1.5 (ctd.)		<p>[Bug 628] Added info about new implementation for references to NVC trees from named objects and NVC partitions to Section 7.2.3</p> <p>[Bug 588] Added info that attribute groups are defined per layer and not valid across different layers to <i>Attribute Groups</i> section</p> <p>[Bug 630] Changed section about non-printable characters in <i>Name Building Block</i> chapter</p> <p>[Bug 660] Added section about references to <i>RoadFixedAttribute</i> sets to Section 6.3.2, and an explanatory paragraph to <i>Road Geometry Line Feature Class</i> section</p> <p>Added section <i>Language Information in NVC Trees</i> to Section 7.4.3</p> <p>[Bug 571] Adapted tile pattern example adapted to Morton Code</p> <p>[Bug 670] Added info on flexible attribute indicating availability of VICS information to <i>Traffic Information Building Block</i> chapter</p> <p>[Bug 622] Added info about signpost icon reference to Signpost Section in Appendix</p> <p>[Bug 839] Added figure and explanatory text to <i>Inter-Building Block References</i> section</p> <p>[Bug 632] Changed info about structure of time zone definitions and daylight saving time</p>

Version number	Published	Changes
1.6 (for NDS 2.1.1 and NDS 2.1.1.1)	2010-06-03	<p>[Bug 571] Added info about differentiation between unidirectional and bidirectional references to <i>Architecture</i> chapter</p> <p>[Bug 486] Added description of POI NVC trees to <i>POIs in Location Input</i> section</p> <p>[Bug 728] Removed section describing that alternate spellings are not stored</p> <p>[Bugs 616 and 677] Added information on 3D junction views and sequence of junction view images to <i>Junction View Building Block</i> chapter</p> <p>[Bug 424] Added info about max. number of intersected tiles in geographical extent of named objects</p> <p>[Bug 671] Corrected language codes in <i>Language Definition</i> section</p> <p>[Bug 471] Added metadata item <code>groupedAttributeList</code> for storing attribute groups to metadata table (Table 18-6)</p> <p>[Restructuring] Moved sections on lane, signposts and warning sign attributes to <i>NDS – Compiler Interoperability Specification, 17 Using Flexible Attributes and Attribute Groups</i> on page 213</p> <p>[New] Added <i>Labeling Hints</i> section to <i>Basic Map Display Building Block</i> chapter</p> <p>[Bug 746] Added <code>MOTORWAY_INTERSECTION_NAME</code> to Table 7-6 <i>Name Formats</i>.</p> <p>[Bug 438] Added <code>complexIntersection</code>, <code>pluralJunction</code>, <code>startAngle</code>, and <code>endAngle</code> to <i>Fixed Attributes for link features</i> table</p> <p>[Bug 538] Changed DataScript names in Digital Terrain Model Building Block and Orthoimages Building Block chapters according to DataScript changes in <code>nds.amd</code></p> <p>[Bug 494] Added info on admin hierarchy grouping to <i>Address Retrieval</i> section</p> <p>[Bug 763] Added named object class „Hamlet“ to Figure 10-1</p> <p>[Bug 784] Added info that references to map lines are stored in routing geo tile</p> <p>[Bug 679] Added POI relation type for guidance POIs</p> <p>[Bug 700] Added info about rotated BMD icons</p> <p>[New] Added info about full-text search for names and POIs to <i>Full-text Search Building Block</i> chapter</p> <p>[Bug 495] Changed AES-256 to AES 128</p> <p>[Bug 562] Changed <code>offset</code> to <code>heightOffset</code> in Table 12-2</p> <p>[Bug 806] Added info that XDG data is mandatory for NDS</p> <p>[Bug 748] Changed description for <code>OFFICIAL_ABBREVIATION</code> in Table 7-6</p>

Version number	Published	Changes
1.6 (continued)		<p>[Bug 745] Added info about name assignments in routing aux table to Section 6.5.2</p> <p>[Bug 743] Changed info about direction of phonetic representation references</p> <p>Made several changes due to new global feature ID (flexible attribute), which may be assigned to links and used for referencing</p> <p>[Bug 646] Added flexible attribute FERRY_TRAVEL_TIME to Table 17-3 <i>Flexible attributes specifying time periods</i></p> <p>[Bug 651] Added named object class LICENSE_PLATE_ZONE to Figure 7-1 <i>Named object feature classes</i></p> <p>[Bug 680] Added the info to chapter 15-2</p> <p>[Bug 518] Changed description of attribute groups in Table 3-3 <i>Attribute Grouping</i></p> <p>Removed paragraph about levels used for generalization in Name building block from <i>Level</i> section; removed bullet „For searching names...“</p> <p>[Bug 606] Moved <i>Address Points</i> section from <i>Routing Building Block</i> chapter to <i>Name Building Block</i> chapter</p> <p>[Bug 820] Added info about reference points</p> <p>[Bug 876] and others: Integration of prerecorded voice, renaming of Phonetic Representation building block to Speech building block, see chapter 14 <i>Speech Building Block</i> on page 281 and sections describing dependencies to Speech building block in Routing, Names, TMC, and POI</p>

Version number	Published	Changes
1.7 (for NDS 2.1.1.2, 2.1.1.3)	August 2010	<p>[Bugs 855 and 856] Added info about extensions of icon data for rendering road icons to Chapter 20 <i>Common Data</i> on page 349 and Section 13.5 <i>Representing POIs as Icons</i> on page 270</p> <p>[Bug 872] Changed definition of <code>TIME_RANGE_OF_DAY</code> attribute in Table 20-2</p> <p>[Bugs 874 and 1052] Added new structures for Speech building block, see Section 14.2 <i>Grammatical Structures for Phonetic Transcriptions</i> on page 286</p> <p>[Bug 940] Added clarification regarding feature classes</p> <p>[Bug 941] Added subdivision of table Table 10-2</p> <p>[Bug 944] Added attributes for junction view images to Table 18-1</p> <p>[Bug 957] Changed <code>dtmLevelDefinition</code> to <code>dtmLevelAvailability</code> in Table 12-1</p> <p>[Bug 1023] Added fixed attribute <code>insideCityLimitStatus</code> to Table 6-2 and Table 6-4 in the <i>Routing Building Block</i> chapter</p> <p>[Bug 1080] Added following common metadata to Table 17-8 <i>Common Metadata</i>: <code>mapDataReleaseDate</code>, <code>releaseYear</code>, <code>releaseMonth</code></p> <p>[Bug 900] Changed DataScript locations of fixed attributes <code>startAngle</code>, <code>endAngle</code> and <code>averageSpeed</code> to <code>nds.routing.link</code> in Table 6-2 and for <code>averageSpeed</code> in Table 6-4</p> <p>[Bug 1034] Added info about storing tile IDs for referenced map lines to Section 6.2.5, subsection <i>A Route Link can Reference One or Several Road Map Lines in the Basic Map Display Building Block</i>.</p> <p>[Bug 792] Added glossary entry for „profile“</p>
1.8 (for NDS 2.1.1.4, 2.1.1.5)	October 2010	<p>[Bug 861] Added definition of south-west rule in Glossary</p> <p>[Bug 1067] Added <code>DAYTIME_RUNNING_LIGHT</code> to Table 17-6, <code>row groupedAttributeList</code></p> <p>[Bug 1105] Size reduction: Reworked <i>Profile Attributes</i> section in <i>Routing Building Block</i> chapter</p> <p>[Bug 859] Added Section <i>Vanity POIs</i> to Section 10.6 <i>POI Regions</i></p> <p>[Bug 881] Added information about relation between POI categories and administrative areas to Chapter 10 <i>POI Building Block</i></p> <p>Added a note about the Levenshtein distance to Chapter 16 <i>Full-text Search Building Block</i></p> <p>Removed XDG section and all references to XDG according to TC decision</p>

Version number	Published	Changes
1.9 (for NDS 2.1.1.6)	2011-02-07	<p>Reworked <i>3D Objects Building Block</i> chapter</p> <p>Reworked <i>Traffic Information Building Block</i> chapter: New relational model, data structures for VICS</p> <p>[Bug 620] Explained drawing order in more detail; added information about drawing order of road geometry lines</p> <p>[Bug 1216] Added info that routing geo tile can also be used for data stored on higher levels; changed Section <i>Routing Tiles and Routing Geo Tiles</i> accordingly</p> <p>[Bug 1329] Added new Section 10.5.2 to <i>POI Building Block</i> chapter: New parameters in POI icon set table for drawing overlapping icons</p> <p>[Bug 593] Added subsection for default and application-specific icon handling to Section 16.3 <i>Icons</i></p> <p>[Bug 1217] Added information to Section 6.2.2 <i>Route Links</i> and Chapter 8 <i>Basic Map Display Building Block</i> that a link (higher level 13) in routing data which is visible in the respective BMD levels must have at least one reference to a corresponding road map line</p> <p>[Bug 1321] Added information to Subsection <i>Contained-In Relation</i> that contained-in relations are mandatory</p> <p>[Bug 1216] Deleted „level 13“ from routing geo tile definition in Subsection <i>Routing Tiles and Routing Geo Tiles</i></p> <p>[Bug 593] Added information about custom and default icons to Section 17.3 <i>Icons</i>.</p> <p>[Bug 1362] Changes in <i>Digital Terrain Model Building Block</i> chapter and in Table 17-5 <i>Global Metadata referring to an Update Region</i> according to changes for level metadata in DataScript.</p> <p>[Bug 1286] Added <i>length</i> to Table 6-2 <i>Fixed attributes of link features</i></p> <p>[Bug 1352] Removed Section on labeling line features in Chapter <i>Basic Map Display Building Block</i>.</p>

Version number	Published	Changes
2.0 (for NDS 2.1.1.7)	2011-06-02	<p>[Bug 936] Added PARENT_NVC relation in <i>Name Building Block</i> chapter, Table 7-4 and Section <i>Parent NVC relation</i>.</p> <p>[Bug 1341] Changed attribute names BY_PASS and SHORT_CUT to BYPASS and SHORTCUT in Appendix D</p> <p>[Bug 1439] Replaced phonetic <i>representation</i> with phonetic <i>transcription</i></p> <p>[Bug 1449] Three new named object classes (zone, country set, aggregated named object class) added to Figure 7-1.</p> <p>[Bug 1450] Replaced coords with coord in Table 8-4 <i>Fixed attributes of point features</i></p> <p>[Bug 1470] Renamed MONTH_OF_YEAR to MONTHS_OF_YEAR</p> <p>[Bug 1443] Changed description of extended postal code to relative geographic position in Section 7.4.7 <i>Extended Postal Codes in NVC Trees</i></p> <p>[Bug 1565] Added row for isoCountryCode in Table 17-6 <i>Region Metadata</i></p> <p>[Bug 1611] Made required changes to Name BB chapter, subsection <i>Dependencies to Speech Building Block</i> and Speech BB chapter, subsection <i>Tables of the Speech Building Block</i></p> <p>[Bug 1658] Added cross reference to name metadata to Section 7.4.6 <i>Mixed case Characters in NVC Trees</i></p> <p>[Bug 1440] TTS removed as prerecorded voice format in Chapter 11 <i>Speech Building Block</i>, subsection <i>Prerecorded Voice Data</i>.</p> <p>[Bug 1435] Clipping of house number ranges added to Table 7-6 <i>Name formats</i> and to Section <i>Dependencies to Name Building Block</i> in <i>Basic Map Display Building Block</i> chapter.</p> <p>[Bug 1437] Added information about attribute type availability for NVCs and named objects to Table 7-2 <i>Metadata of the Name Building Block</i>.</p> <p>[Bug 1444] Changed description for DEFAULT_OFFICAL_NAME and PREFERRED_ALTERNATE_NAME in Table 7-5.</p> <p>[Bug 1484] Added on route/off route to list of usage type mask values in Section 17.3 <i>Icons</i></p> <p>[Bug 1353] Added information that language code is stored with descriptive names in description for nameList in Table 7-8.</p> <p>[Bug 1542] Added new cross-references to Appendix <i>List of Flexible Attributes</i> and replaced cross-references to old Appendix <i>List of Flexible Attributes ...</i> with cross-references to new Appendix <i>List of Flexible Attributes</i></p> <p>[Bug 1417] Changed figure and explanation for DETACHED_FROM_TERRAIN attribute in <i>Basic Map Display Building Block</i> chapter.</p>

Version number	Published	Changes
2.0 (ctd.)		<p>[Bug 1464] Added new subsection in Appendix B for lane connectivity use case where no attribute is needed.</p> <p>[Bug 1235] Removed row for <code>defaultFeatureClass</code> from Table 10-9 <i>Data Structures for POI NVC Trees</i> and changed the paragraph accordingly.</p> <p>[Bug 1353] Changed explanation for <code>nameList</code> in Table 7-8 <i>Properties of Selection Criteria</i>.</p> <p>[Bug 1369] New model for icon sets – changes in all relevant sections</p> <p>[Bug 1405] Added section <i>Big Cities in NVC Trees</i> to <i>Name Building Block</i> chapter</p> <p>[Bug 705] Added new sections 8.12.3D <i>Objects Representing Building Footprints</i> and 14.7 <i>Avoiding Overlap of 3D Objects and Building Footprints</i></p> <p>[Bug 1334] Added information about time zone names to Section 17.2.8 <i>Time Zone Information</i></p> <p>[Bug 1122] Added subsections about Multipolygons and Pseudo Edges to Section 7.4 <i>Area Features in Basic Map Display Building Block</i> chapter; moved two paragraphs from <code>subsectionArea Features</code> to Section 7.4</p> <p>[Bug 1445] Added information about ISO country code XXX for international waters to table 7-6, row <code>OFFICIAL_ABBREVIATION</code>, and to Section 17.2.5 <i>Lnguae Definition</i></p> <p>[Bug 1446] Changed subsection <i>Flexible Attribute for Defining Currency Values</i> in <i>Common Data</i> chapter</p> <p>[Bug 1400] Added additional explanations for <code>tileID</code>, <code>classID</code>, <code>localIndex</code> in Table 3-2 <i>Components of the feature ID</i></p> <p>[Bug 1423] Added information about leading coefficients to Section 6.7.3 <i>Discrete Cosine Transformation</i></p> <p>[Bug 1457] Added information about <code>accessLocaionList</code> to Section A <i>Named Object Feature Has Access Location(s)</i></p> <p>[Bug 1493] Added note with reference to <i>Big Tiles</i> article in NDS Wiki to Section 4.3 <i>Tiling Scheme</i></p> <p>[Bug 1403] Changed Table 7-9 with regard to size optimization changes for NVCs</p> <p>[Bug 1497] Added <code>ZIP_CODE_RANGE</code> to Table 7-6 <i>Name formats</i>.</p> <p>[Bug 1460] Changed Section <i>References to Next-valid-character Trees</i> in <i>Name Building Block</i> chapter</p> <p>Added feedback from Garmin</p> <p>[Bug 764] Added information about transliteration of house number ranges to Table 7-6 <i>Name Formats</i></p>

Version number	Published	Changes
2.0 (ctd.)		<p>[Bug 1448] Added information that language table is unique per product database to Section 17.2.5 <i>Language Definition</i></p> <p>[Bug 1867] Changed information about routing levels to be used for generalization in subsection <i>Levels</i> in <i>Routing BB</i> chapter</p>
2.1 (for NDS 2.1.1.8)		<p>[Bug 1677] Replaced <code>negativeOffsetLocationId</code> and <code>positiveOffsetLocationId</code> with <code>negativeLocId</code> and <code>positiveLocId</code>. Replaced <code>distanceToPosOffsetLoc</code> and <code>distanceToNegOffsetLoc</code> with <code>distanceToPosLoc</code> and <code>distanceToNegLoc</code>.</p> <p>[Bug 1636] Enhanced <i>Routing Metadata</i> section (<i>Routing BB</i> chapter) and <i>Attribute Layer</i> section (<i>Architecture</i> chapter) with information that all attribute layers have an attribute type availability.</p> <p>[Bug 1638] Added name format <code>MOTORWAY_EXIT_NAME</code> to table 7-6 <i>Name Formats</i></p> <p>[Bug 1643] Added <code>TEXT_SEPARATOR</code> in Table 17-8 <i>Common Metadata</i></p> <p>[Bug 1660] Added information about virtual tiles for POI, changed information on tiles and tiling in chapters: <i>Architecture, Partitioning of Geographic Data, POI Building Block, and Glossary</i></p>
2.2 (for NDS 2.1.1.9 and 2.1.1.10.1 - 2.1.1.10.6)	2012-01-02	<p>[Bug 1582] Changed description for row <code>TmcCrossReferenceTable</code> in Table 9-3 <i>Lookup tables for location information, overview</i></p> <p>[Bug 1803] Added <code>iconSetId</code> in Figure 9-10 <i>Lookup table for TEC event information in NDS</i></p> <p>[Bug 1744] Routing BB, Section 7.6.1 <i>Attributes for ADAS</i>: Changed attribute name to <code>CURVATURE</code>; added Glossary definitions for curvature and clothoid</p> <p>Changed tile definitions according to results from documentation taskforce meeting in Hildesheim, August 4, 2011 (see meeting minutes)</p> <p>Changed definition of base links according to results from documentation taskforce meeting in Hildesheim, August 4, 2011 (see meeting minutes)</p> <p>[Bug 1789] Changed description of Name format <code>EXIT_NUMBER</code> in Table 8-6 <i>Name formats</i> accordingly</p> <p>[Bug 1768] Added information about ADAS data availability type to Section 6.6 <i>Data Structures for Advanced Driver Assistance (ADAS)</i></p> <p>[Bug 1969] Changed description of <code>iconDrawingPriority</code>: Higher number has higher priority instead of lower number</p>

Version number	Published	Changes
2.2 (ctd.)		<p>[Bug 1999] Added information about Labeling hints with tangential text in Section 9.13 <i>Labeling Hints</i>: Tangential text can be modeled by means of curved labeling hints</p> <p>[Bug 1880] Changed OrthoImageFormat to ImageFormat in subsection <i>Storage Format</i> of the <i>Orthoimage Building Block</i> chapter</p> <p>[Bug 2228] Changed second bullet in note in Section 4.2 <i>Coding of Coordinates</i> according to proposal (added operation round)</p> <p>[Bug 2135] Changed sentence about references from administrative areas to respective links in Section 7.7 <i>Address Retrieval</i> to make clear that attribute groups are attributes of links.</p> <p>[Bug 1250] Changed attributes names according to changes in DataScript: BUILDING_ELEVATION renamed to BASE_ELEVATION</p> <p>[Bug 1793] Table 12-2 <i>Height Map Data</i>: Row compressionType removed and changed <i>Compression for Height Maps</i> section accordingly</p> <p>[Bug 1797] Table 17-5 <i>Global metadata referring to an update region</i>: Removed row urBoundingBox</p> <p>[Bug 1820] Added note with information about ENHANCED_GEOMETRY attribute in sections 6.3 <i>Road Geometry Line Feature Class</i> and 6.6 <i>Data Structures for Advanced Driver Assistance (ADAS)</i></p> <p>[Bug 2047] Changed DataScript location for TextureMapTable in subsection <i>Textures in 3D Objects BB</i> chapter</p> <p>[Bug 2050] Changes subsection <i>Orthoimage Metadata</i> in <i>Orthoimage BB</i> chapter according to DataScript (OrthoImageMetadata and OrthoImageMetadataTable)</p> <p>[Bug 2106] Removed leading zeros in hexadecimal codes in tables 4-1 to 4-3 in Chapter <i>Partitioning of Geographic Data</i>; added 0x prefix where missing</p> <p>[Bug 2201] Changed sentence „This means for tile 0 ...“ below Figure 4-1 <i>Tiling scheme for partitioning the Earth's surface: Level 0</i> according to proposal in Bug 2201</p> <p>[Bug 1781] Added subsection <i>Icon Sprites</i> to Section 17.3 <i>Icons</i></p> <p>[Bug 1759] Added subsection <i>3D Icons</i> to Section 17.3 <i>Icons</i></p> <p>[Bug 1786] Added subsection <i>Bannered Routes</i> to Section 17.3 <i>Icons</i></p> <p>[Bug 1833] Added Section 14.9 <i>Landmark Icons</i> to <i>3D Objects Building Block</i> chapter</p> <p>[Bug 1990] Changed “free flow speed” to “baseSpeed” in Section 7.7.1 <i>Flexible Attributes for Speed Profiles</i>.</p> <p>[Bug 2021] Changed maximum value of heightOffset to 32767 in section 15.1.1 <i>Structure and Content of Height Maps</i></p>

Version number	Published	Changes
2.2 (ctd.)		<p>[Bug 2246] Changed figure <i>Selection graph and NVCs, example</i> in Section 8.6.1 <i>Selection Graphs and NVC Trees</i></p> <p>[Bug 1695] Changed second sentence in Section 14.5 <i>References to Geometry Information</i> to „at least one LOD“</p> <p>[Bug 2202] Changed paragraphs below Figure 5-6 <i>Features extending over more than one tile</i> in Chapter <i>Partitioning of Geographic Data</i></p> <p>[Feedback from partners] Added paragraph on using multiple FTS in Chapter 17 <i>Full-text Search Building Block</i></p> <p>[Feedback from partners] Added explanation of constants for special control characters in Section <i>Non-printable Control and Separation Characters</i></p> <p>[Bug 1333] Several changes in Format and Compiler Interoperability Specification to improve the description of base links and route links</p> <p>[Bug 2333] Improved description of fixed attributes URBAN and INSIDE_CITY_LIMIT_STATUS</p> <p>[Bug 2334] Added line on transitions and intersections in Section <i>Intersection Feature Class</i> in the <i>Routing Building Block</i> chapter</p> <p>[Bug 2246] Added figure <i>Selection graphs, example</i> and added sentence to specify difference between selection graphs and physical representation of NVCs</p> <p>[Bug 2297] Changed definition of SpatialTreeInnerNode from 4-bit to 6-bit value in section <i>3D Object Feature</i> in <i>3D Objects Building Block</i> chapter</p> <p>[Bug 1911] Deleted START_OR_END_HOUSE_NUMBER in Table <i>Name Formats</i> in Name BB chapter and changed description of HOUSE_NUMBER_RANGE: Must be stored with base links and road geometry lines, can additionally be stored with route links</p> <p>[Bug 2011] Changed colorId to wallColorId and roofColorId in Table <i>Flexible attributes for 2.5D city models</i> in Basic Map Display BB chapter</p> <p>[Bug 2157] Added information about transparent extensions in Section <i>Extending the Navigation Data Standard</i> and added the building block type EXTENSION to the list of building blocks types in subsection <i>Building Block</i> (both in <i>Architecture</i> chapter)</p> <p>[Bug 2311] NvcTreeAvailability changed to NvcTreeAvailabilityType; LOAD_OR_GENERATE changed to USE_ALTERNATE and adapted description accordingly in Section <i>References to Next-valid-Character Trees</i> in Name BB chapter</p> <p>[Bug 1793] Updated appendix with NDS supplier IDs.</p> <p>[Bug 2331] Sentence added in About this Document chapter that examples used in the Specification documents are not binding.</p>

Version number	Published	Changes
2.2 (ctd.)		<p>[Bug 1459] Reworked paragraph on downlink optimization in section 8.5.3 <i>Inter-Level References</i></p> <p>[Bug 1829] Changed sentence on predefined POI categories in section 12.4.1 <i>Attributes of POI Categories</i> to mention the generic predefined category</p> <p>[Bug 2077] Reworked section 9.6.5 <i>Mixed-case Characters in NVC Trees</i></p> <p>[Bug 1787] Added information about curved and anti-labeling hints for line features to the <i>Labeling Hints</i> section in <i>Basic Map Display</i> chapter</p> <p>[Bug 2043] Added section about DTM metadata in <i>Digital Terrain Model Building Block</i> chapter</p> <p>[Bug 2049] Added subsection for DTM pattern tile in <i>Digital Terrain Model Building Block</i> chapter</p> <p>[Bug 1811] Added new chapter <i>Overall Building Block</i> Restructuring: Deleted chapter Common Data, added Overall Building Block chapter and introduced a chapter for product database related data.</p> <p>[Bug 2408] Added information about <code>isTransparent</code> in <i>3D Objects Building Block</i> chapter in section <i>Material and Textures</i></p> <p>[Bug 2342] Added a reference to DataScript <code>nds.bmd.common</code> for the complete list of point features implemented in NDS; this also covers future enhancements of the list</p> <p>[Bug 2423] Added named object class <code>NEIGHBORHOOD</code> in <i>Name Building Block</i> chapter, Figure 10-2 <i>Named Object Feature Class</i></p> <p>[Bug 1825] Changed section about Road Number Class Prefix table</p> <p>[Bug 2430] Added <code>position</code> in <i>Junction View BB</i> chapter in <i>Attributes for junction view images</i> table</p> <p>[Bug 2425] Added information about URIs provided as relative address in <i>Database Structure</i> chapter in <i>Fields of the building block component version table</i> table</p> <p>[Bug 2313] Composed POI region ID: Changed description in Relation between <i>POI regions and Named Objects</i> section in <i>POI Building Block</i> chapter</p> <p>[Bug 2313] Added information about composed POI region ID in Section <i>Relation between POI Regions and Named Objects</i></p> <p>[Bug 2223] Added information about numbers in NDS to <i>Introduction to DataScript</i> section in <i>Architecture</i> chapter</p> <p>[Bug 1565] ISO country code now in region metadata, changed section on name formats</p> <p>[Bug 2033] Reworked sections on BMD feature generalization in Format and Compiler Interoperability Specifications</p> <p>[Bug 2150] Added information that POI virtual tile table is mandatory and shall contain only one level</p>

Version number	Published	Changes
2.2 (ctd.)		<p>[Bug 2260] Added Icon and Overall building blocks to architecture overview image</p> <p>[Bug 2314] Added information on bit-mask approach in filtering of POI secondary attributes</p> <p>[Bug 1973] Updated <i>Appendix NDS Database Supplier IDs</i></p> <p>[Bug 1988] Added clarification on definition of intersections in <i>Routing Building Block</i> chapter</p> <p>[Bug 2283] Changed Copyright statement, as NDS e.V. is no longer in foundation</p>

## 1.2 Purpose and Subjects

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

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

The document deals with the following subjects:

- Chapter 2 *Introduction to Navigation Data Standard* on page 41. This chapter introduces Navigation Data Standard (NDS).
- Chapter 3 *Architecture* on page 43. This chapter introduces the architecture of databases that comply with Navigation Data Standard and explains the most important terms and types of data structures used. The chapter also gives a short introduction to Relational DataScript (RDS), and describes possibilities to extend the standard.
- Chapter 4 *Database Structure* on page 59. This chapter introduces the structure of an NDS database and the product-specific metadata stored in the Product Database building block.
- Chapter 5 *Overall Building Block: Update Region Metadata* on page 73. This chapter introduces the overall metadata for all building blocks stored in the Overall building block.
- Chapter 6 *Attributes* on page 81. This chapter explains how attributes are used in NDS databases.
- Chapter 7 *Partitioning of Geographic Data* on page 89. This chapter makes you familiar with the concepts Navigation Data Standard uses to map geographic data to a database. Based on the underlying coordinate system, the tiling scheme used for partitioning geographic data is introduced.
- Chapter 8 *Encryption* on page 101. This chapter introduces the concept of encryption for the content of NDS databases.

- Chapter 9 *Routing Building Block* on page 107. This chapter introduces the basic concepts relevant for the Routing building block. It describes the building block structure and content and makes you familiar with the routing features. The individual feature classes used to represent the routing features in the database are explained. The chapter also describes attributes that cover specific use cases, such as Advanced Driver Assistance.
- Chapter 10 *Name Building Block* on page 163. This chapter introduces the basic concepts relevant for the Name building block. It describes the building block structure and content and makes you familiar with the name features. The individual feature classes used to represent the name features in the database are explained as well as the conceptual approach of Navigation Data Standard to location input and address retrieval.
- Chapter 11 *Basic Map Display Building Block* on page 201. This chapter introduces the basic concepts relevant for the Basic Map Display building block. It describes the building block structure and content and makes you familiar with the basic map display features. The individual feature classes used to represent the basic map display features in the database are explained.
- Chapter 12 *Traffic Information Building Block* on page 229. This chapter introduces the concept of traffic information. It describes how applications can be enabled to decode Traffic Message Channel (TMC) messages for displaying them in maps, for their automatic consideration in dynamic route calculation, and for translation to speech and/or text output.
- Chapter 13 *POI Building Block* on page 253. This chapter introduces the concept for integrating Points of Interest (POIs) in Navigation Data Standard. It describes the building block structure and content and introduces concepts for using third-party POIs in NDS.
- Chapter 14 *Speech Building Block* on page 281. This chapter introduces the concept for integrating phonetic transcriptions and prerecorded voice data in Navigation Data Standard.
- Chapter 15 *Digital Terrain Model Building Block* on page 289. This chapter introduces the modelling of terrain in Navigation Data Standard. It introduces the two possible representations of the digital terrain model (DTM): Height maps and Batched Dynamic Adaptive Meshes (BDAM). Structure and content of the DTM building block are explained.
- Chapter 16 *Orthoimages Building Block* on page 305. This chapter introduces the use of orthoimages in Navigation Data Standard. It describes the metadata for orthoimages and how the data is stored in the Orthoimages building block.
- Chapter 17 *3D Objects Building Block* on page 309. This chapter introduces the concepts of modelling three-dimensional objects in Navigation Data Standard.
- Chapter 18 *Junction View Building Block* on page 341. This chapter introduces the concept of junction views, which give the driver realistic visual information of a junction and the required maneuver.
- Chapter 19 *Full-text Search Building Block* on page 345. This chapter introduces the concept for a string-based search for objects in an NDS database. It describes the building block structure and content and how full-text search can be used in an NDS database.
- Chapter 20 *Icons* on page 355. This chapter explains, how Icons are used in NDS databases.

- Appendix A *Glossary* on page 359. This Appendix gives definitions and concepts for terms used in the *NDS – Format Specification*.
- Appendix B *NDS Database Supplier IDs* on page 369. This Appendix lists the current NDS database suppliers and their IDs.
- Appendix C *Overview of Database Tables* on page 371. This Appendix gives an overview of the tables of an NDS database.
- Appendix D *List of Flexible Attributes* on page 381. This Appendix lists the flexible attributes, which are available for NDS.

## 1.3 Target Audience

This document is provided for IT professionals who want to get familiar with Navigation Data Standard. This can be, for example, software development managers, product managers specifying automotive 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

## 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
<b>Requirement</b> 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
<b>Recommendation</b> 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
<b>Permission</b> 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
<b>Possibility and capability</b> These verbal forms are used for stating possibilities or capabilities, being technical, material, physical, or causal.	can	be able to, there is a possibility of, it is possible to
	cannot	be unable to, there is no possibility of, it is not possible to

## 1.6 Typographical Conventions

This documentation uses the following typographical conventions:

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

The following types of note are used:

---

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

---

---

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

---

---

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

---

---

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

---

## 2 Introduction to Navigation Data Standard

The NDS e.V. is a registered society of car manufacturers, system suppliers and map suppliers. The objective of the initiative is the standardization of a physical storage format for car navigation systems.

One of the drawbacks of today's navigation systems is that each system needs its own physical format for navigational data. A direct exchange of map data between different navigation systems is not possible.

A standardized binary format, which also provides new features, opens many new opportunities.

The partners of the initiative have agreed upon a set of requirements which Navigation Data Standard shall meet. Based on these requirements, the following objectives have been defined:

### **Compatibility and Interoperability**

The key objective is compatibility and interoperability allowing the exchange of navigational data between systems that comply with Navigation Data Standard.

All information relevant for interoperability is contained in the *NDS – Compiler Interoperability Specification*.

### **Separation of Application Software and Navigational Data**

Today's navigation systems combine application software and navigational data. Separating software from data will create flexibility between systems.

### **Updates**

At present, a variety of update media and update mechanisms are available for in-car systems and mobile clients. With Navigation Data Standard, a solution is developed that allows for a harmonized format for different systems as well as for flexibility regarding content update, including incremental and partial updates. Moreover, database content can be enriched without changing basic structures.

The update scenarios and update mechanisms supported by NDS are described in the *NDS – Update Specification*.

### **Further Objectives**

Besides the objectives listed above, the following aims are set:

- World-wide use
- Format compactness and efficiency
- Support of different distribution and work media
- Prevention of illegal use



## 3     Architecture

This chapter introduces the architecture of databases that comply with Navigation Data Standard (so-called *NDS databases*). It explains the most important terms and key concepts used.

### 3.1    NDS Database and Its Interfaces

Navigation system suppliers often provide physically or logically separate data collections, for example, for routing, map display and speech output, and refer to these collections as databases. In other contexts, the entire set of such data collections delivered on one data carrier is also called a database.

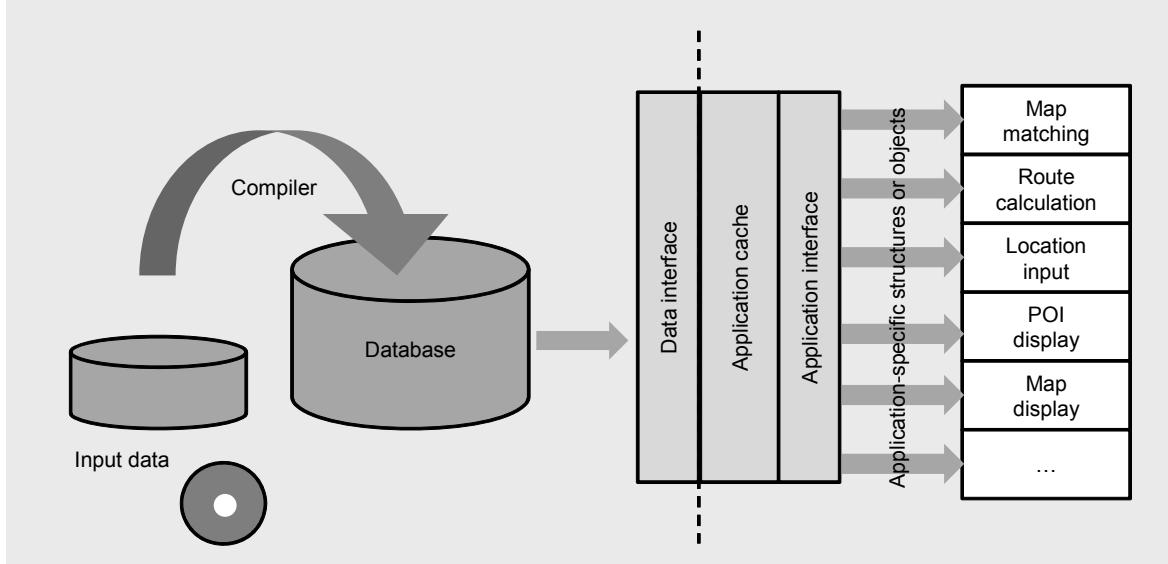
To avoid this ambiguity, Navigation Data Standard defines a database as the totality of navigational data that a navigation system operates on at a given time. The content and structure of this database are standardized by this specification. Standardization aims at enabling any standard-compliant application to work with any standard-compliant database.

The applications can work with database content from more than one location simultaneously, for instance, when a full database on the hard disk drive is supplemented by additional POI content from a flash card. In this case, the two physical entities still form a single logical database.

The database files must be stored in a format that can be processed by SQLite 3.7.4-nds3. For more information, refer to <http://www.sqlite.org/fileformat.html>.

Figure 3-1 illustrates a navigation system in the context of Navigation Data Standard. The dashed line in the middle marks the border between the database that fully complies with Navigation Data Standard as described in the current format specification and the application-specific components depicted in the right part of the figure.

Figure 3-1 NDS-compliant database and its interfaces



A compiler creates standard-compliant data from input data. The compiled NDS-compliant data can be used by different applications for navigation, map display or other map-based services. Typical applications supported by Navigation Data Standard are in-car or hand-held navigation systems.

## 3.2 NDS Database Structure

An NDS database may consist of several *product databases*, and each product database may be divided further into *update regions*. This concept supports a flexible and consistent versioning concept for NDS databases and makes it possible to integrate databases from different database suppliers into one NDS database. The inner structure of databases complying with Navigation Data Standard is further characterized by *building blocks*, *levels* and the content itself.

### 3.2.1 Product Database

Each product database is delivered by one database supplier, has its own version control and can therefore be updated independently from other product databases. Product databases can contain one or more building blocks. Product databases cover a geographic area, which can be further divided into several update regions.

---

**Example** An NDS database comprises the following two product databases:

- Europe basic navigation (including the Basic Map Display, Routing and Name building blocks) produced by Harman Int.
  - Europe POIs produced by Michelin
- 

All information related to product databases is stored in the product database table (DataScript location: `nds.overall.version > ProductDbTable`).

**Related Topics:**

- *4 Database Structure* on page 59
- *3.2.2 Update Region* on page 45
- *3.2.3 Building Block* on page 46

### 3.2.2 Update Region

Update regions enable incremental and partial updating of defined geographic regions within an NDS database.

An 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. The extent is determined by the requirements of the map database products defined by map suppliers, system vendors, or car manufacturers.

The *NDS – Update Specification* provides a detailed conceptual description of update regions in connection with a detailed description of the versioning concept for databases that comply with Navigation Data Standard.

**Related Topics:**

- *NDS – Update Specification, 3.2 Update Region* on page 23
- *5 Overall Building Block: Update Region Metadata* on page 73

### 3.2.3 Building Block

All navigational data is regarded as belonging to a specific building block. Each building block addresses specific functional aspects of Navigation Data Standard.

---

<b>Note</b>	Building block boundaries are sometimes motivated by access efficiency rather than by logical coherence and may therefore not directly reflect the needs of a specific application. In this case application programming interfaces (APIs) accessing the database can create a more suitable logical view on the database by filtering and aggregating data from different building blocks (see Figure 3-1).
-------------	--

---

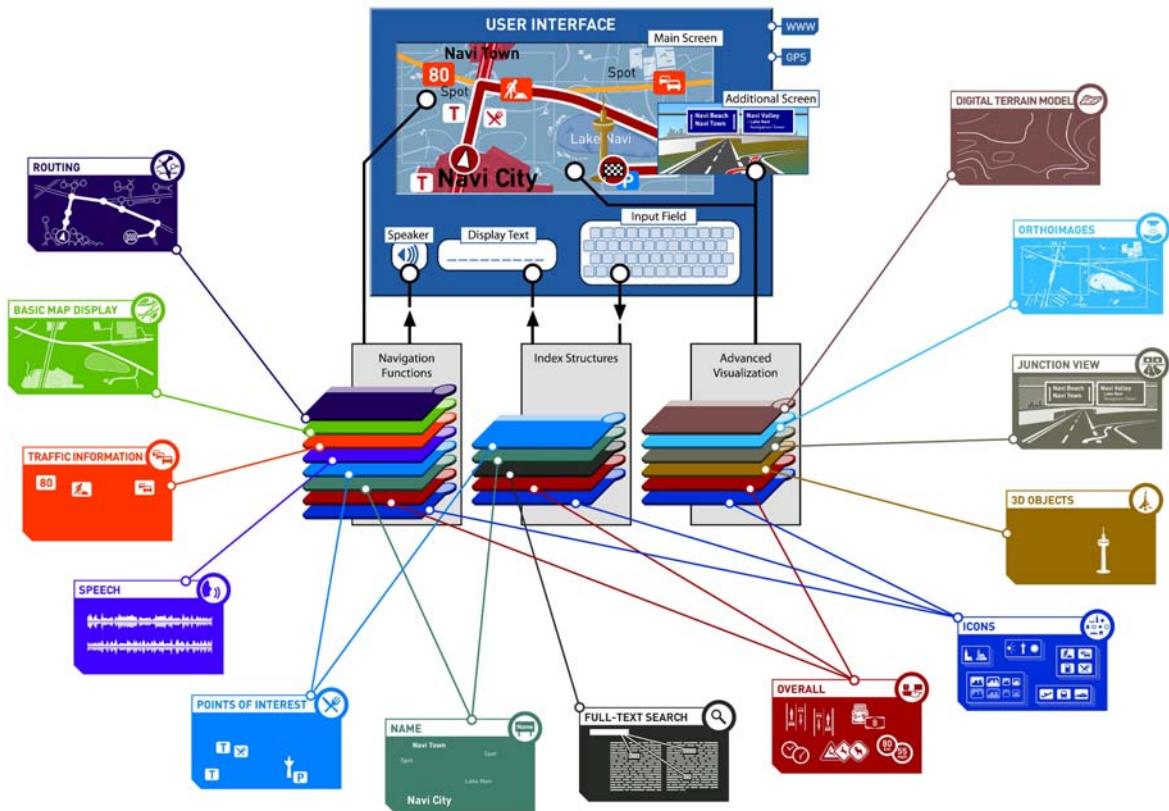
The following are typical building blocks in an NDS database (DataScript location: `nds.overall.productdbversion > buildingBlockType`):

- Overall building block (**OVERALL**)
- Basic Map Display building block (**BASIC\_MAP\_DISPLAY**)
- Routing building block (**ROUTING**)
- Name building block (**NAME**)
- POI building block (**POI**)
- 3D Objects building block (**OBJECTS\_3D**)
- Orthoimages building block (**ORTHO\_IMAGE**)
- Digital Terrain Model building block (**DTM**)
- Speech building block (**SPEECH**)
- Junction View building block (**JUNCTION\_VIEW**)
- Traffic Information building block (**TRAFFIC\_INFORMATION**)
- Full-text Search building block (**FTS**)
- Icon building block (**ICON**)
- Extension building block (**EXTENSION**)

The Routing and Name building blocks are mandatory building blocks. They are necessary, for example, for location input, route calculation, route guidance, and map matching. All other building blocks are optional. They are only required if the navigation applications use this kind of data.

Figure 3-2 depicts the building blocks of an NDS database and the data that is stored in the respective building blocks.

Figure 3-2 Building blocks in NDS and their data

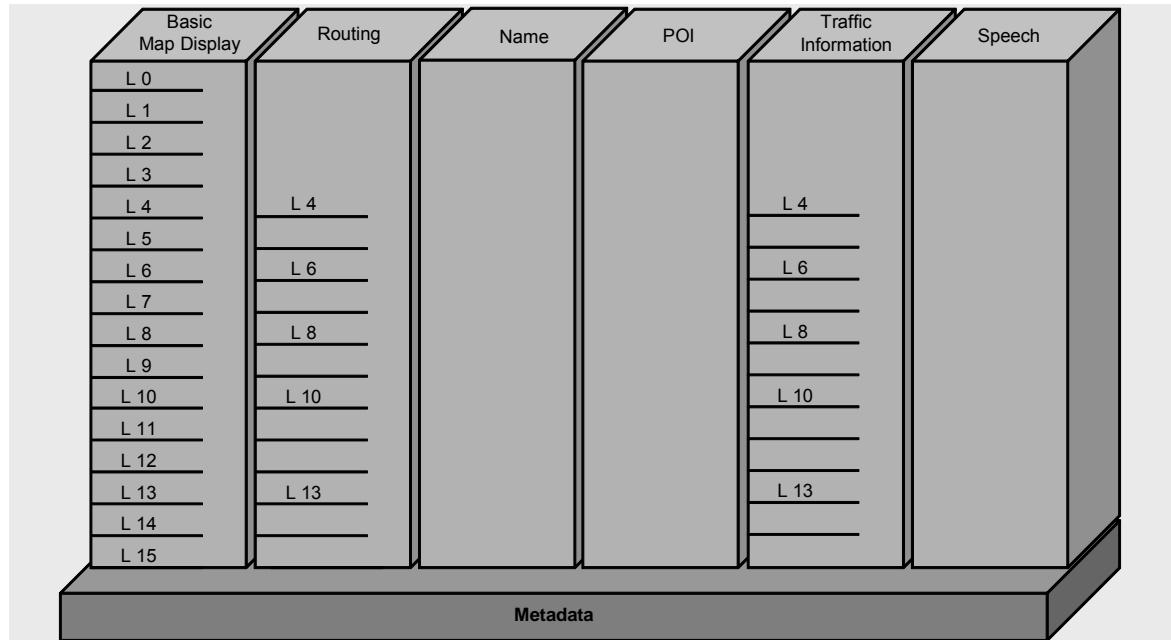


### 3.2.4 Levels

In some building blocks, the data belonging to this building block is partitioned into different levels. The higher the level, the larger are the partitions in terms of spatial size, which is at least partly compensated by lower spatial density and/or reduced content.

Figure 3-3 shows the levels in an example database.

Figure 3-3 Database structure, levels



For routing and basic map display, data is stored partly redundant on different levels with decreasing spatial density of content on higher levels, that is on levels with lower numbers. The reduction of density is achieved through generalization processes during compilation. These processes filter out less important features and reduce details of the remaining features.

By introducing levels, the following advantages are achieved for the different types of applications:

- For map display, higher levels allow a fast display of small-scale levels covering large areas with fewer details.
- Long distance routes can be calculated very quickly taking into account the more important road classes only.

Depending on its type, a building block can have up to 16 levels (see Figure 3-3). Level 13 is mandatory for the Routing building block.

Each level is subdivided into *tiles*, which form a regular grid covering the complete surface of the earth. Tiles partition the data that is available; they are used for data access and referencing data content according to its geographic position.

#### Related Topics:

- Tiles, see [7 Partitioning of Geographic Data](#) on page 89
- Glossary: [Tiling scheme](#) on page 367
- For more information on levels, refer to the individual building block chapters.

### 3.3 Database Content

The following types of data are distinguished in the database:

- *Features*: All real-world objects relevant for a navigation system are represented by one or more features and can be represented on one or more levels.
- *Attributes*: Attributes describe the specifics of the different features.
- *Metadata*: Metadata contains information on variable database content and database properties. It can refer to data of a specific product database, a building block, or the complete database.

Database content is stored as BLOB (for example, in the Routing building block) and/or relationally (for example, in the POI and Names building blocks). For detailed information on the storage method, refer to the individual building block chapters.

#### Related Topics:

- Features: Section 3.3.1 *Features* on page 49
- Attributes: Section 3.3.2 *Attributes* on page 52 and Chapter 6 *Attributes* on page 81
- Metadata: Section 3.3.3 *Metadata* on page 53 and Chapter 5 *Overall Building Block: Update Region Metadata* on page 73

#### 3.3.1 Features

This section introduces the concept of features. It explains *feature classes* and shows the logical structures in which a feature is embedded.

All real-world objects relevant for a navigation system are represented by one or more features and can be represented on one or more levels. For example, road segments between two intersections are represented by link features in the Routing building block, mountain peaks are represented by point features in the Basic Map Display building block, cities are represented by administrative area name features in the Name building block, restaurants are represented by POI features in the POI building block.

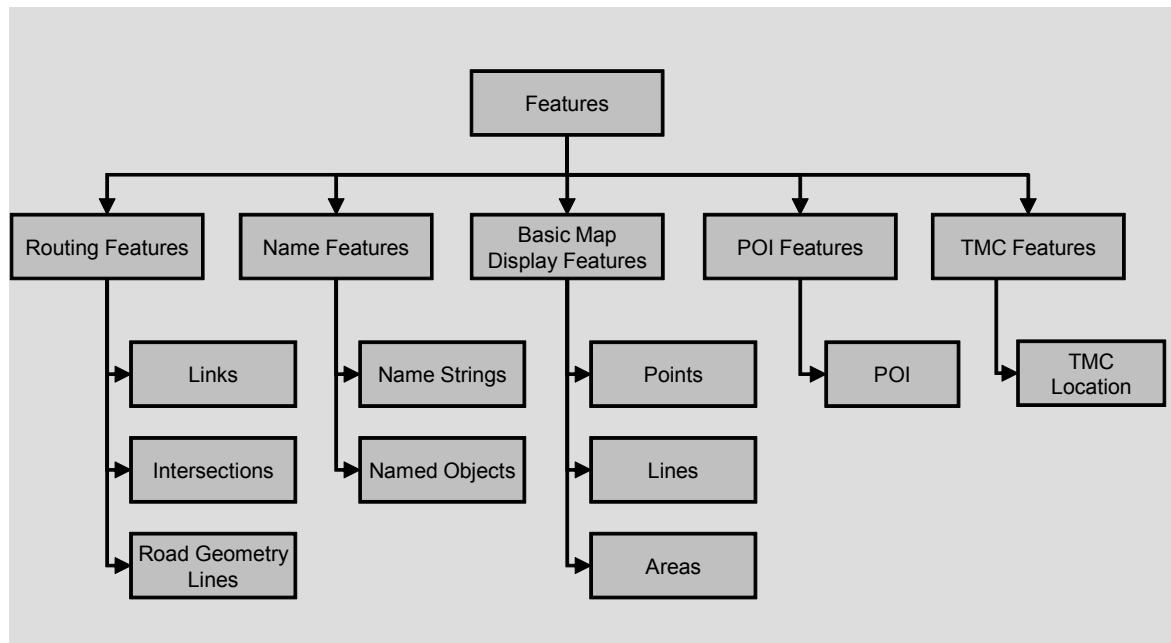
#### Feature Classes

A feature represents a real-world object. All real-world objects relevant for a navigation system have a representation in the NDS database and are mapped to instances of one or more feature classes. For example, a road has a name and connects several intersections. The road name is represented by a name feature. The road is represented by link features.

Navigation Data Standard organizes the classes in a multi-level hierarchy with abstract classes that cannot be instantiated and concrete classes on the lowest level of the class hierarchy that can be instantiated. Figure 3-4 gives examples for feature classes for some building blocks.

For a complete overview down to the lowest level in the hierarchy and for details on the classes and their use, refer to the feature class descriptions in the chapters describing the building blocks.

Figure 3-4 Examples of feature classes



## Feature Structure

The structure of a feature comprises the following:

- Attributes stored with the feature. An attribute is used to describe a property or a characteristic of a feature.
- References to other features. A reference is used to map dependencies between features. A reference can be defined to a feature in the same or a different feature class. The referenced feature can be stored in the same or a different building block either on the same or on a different level.

For more information on attributes, refer to Section 3.3.2 *Attributes* on page 52. For more information on references, refer to Section *Referencing Features* on page 51.

## Identifying Features

Features must be identifiable. This is required

- For applications to access features
- For defining references between features

Table 3-1 shows the components of the feature ID which is used to uniquely identify a feature in Navigation Data Standard. Usually, the full identifier composed of all components is used for referencing. However, depending on the referencing context and to save space, part of the identifier may be omitted.

This identification scheme is only used for the feature classes of the Routing and Basic Map Display building blocks. The other building blocks use different schemes. For more information, refer to Chapter 13 *POI Building Block* on page 253 and Section 12.2.5 *Linking TMC Information to NDS Features* on page 237.

*Table 3-1 Components of the feature ID*

Identifier	Description
tileId	<p>Uniquely identifies the tile on which a feature is located</p> <p>The <code>tileId</code> is composed of the <code>levelNumber</code> and the <code>tileNumber</code>.</p> <p>When referencing a feature, the <code>tileId</code> can be omitted if the tile of the referenced feature can be determined by the context.</p> <p><code>tileNumber</code> and <code>levelNumber</code> are implicitly stored in the <code>tileId</code> (and thus have no equivalents in DataScript), see <i>NDS – Compiler Interoperability Specification</i>, 3.1 <i>Generating Tile IDs</i> on page 53.</p>
ClassID	<p>Uniquely identifies the class a feature belongs to</p> <p>The <code>classID</code> is a logical construct and does not have a representation in DataScript. It indicates that the application needs to distinguish between different feature types, for example, between base and route links. The format usually distinguishes the feature types by feature attributes, such as the <code>ShapePointType</code> attribute, or describes the respective feature type in the name of the list containing the features, for example the <code>RoadGeoLineList</code>.</p> <p>When referencing a feature, this identifier can be omitted if the referencing and the referenced feature belong to the same class or if the reference unambiguously implies the target feature class.</p>
LocalIndex	<p>Uniquely identifies a feature in the list of features of the given class in the given tile</p> <p>The local index describes the general concept of referencing list elements by means of an index. The index is normally not stored with the list, but just used in case a reference to the feature is stored. References to links, for example, are stored as link IDs (<code>LinkId</code>), which actually are indices to the <code>LinkList</code>.</p>

To facilitate loose coupling of building blocks, NDS provides a flexible attribute for assigning a *global feature ID* to features, for example, to links of the Routing building block (DataScript location: `nds.common.flexibleattributes > GLOBAL_FEATURE_ID`). This global ID may be used for references from the Junction View or POI building blocks to the Routing building block, thus making it possible to use junction view and POI databases for more than one NDS product. However, global feature IDs should not be used extensively, as data size increases considerably when this flexible attribute is assigned to many links of the Routing building block.

## Referencing Features

Navigation Data Standard defines the following referencing patterns:

- Inter-building block references, for example, from a link in the Routing building block to a named object in the Name building block. This reference provides the name information necessary for generating route lists.
- Inter-level references, for example, from a link on a lower level to a link on a higher level of the Routing building block. This reference is needed for route calculation purposes.

- Inter-tile references, for example, from a road geometry line to a link. This reference is needed to describe the road geometry of a link that extends over more than one tile.
- Intra-tile references, for example, from an intersection to a link that is connected to it.

The way referencing is done depends on the building blocks and on the feature classes involved. In most cases, features reference each other indirectly. This makes it possible to

- Compile data for different tiles independently of each other
- Update data of single tiles

In Navigation Data Standard, a reference between two features is regarded as a logical and not necessarily as a physical entity. The application creates any physical references in its application cache on demand.

References to feature IDs refer to the position of the respective feature in the feature list.

If the global feature ID is to be used for a reference to a link, the flexible attribute `GLOBAL_FEATURE_ID` must be assigned to the referenced link. See also Section *Identifying Features* on page 50.

### 3.3.2 Attributes

Attributes describe the specifics of the different features. For each feature class, a set of attribute types is provided.

NDS provides two types of attributes: Fixed and flexible. For fixed attributes, a corresponding value must be provided when the feature is instantiated. Flexible attributes are optional and can be combined to form a so-called attribute group to model, for example, complex traffic regulations.

#### Related Topics:

- More information about attributes: Chapter *6 Attributes* on page 81
- List of flexible attributes: Appendix *D List of Flexible Attributes* on page 381
- Use cases for flexible attributes and attribute groups: Chapter *NDS – Compiler Interoperability Specification, 17 Using Flexible Attributes and Attribute Groups* on page 221

### 3.3.3 Metadata

Metadata contains information on variable database content and database properties. NDS distinguishes the following types of metadata:

- Global metadata: Global metadata refers to several or all building blocks in the database. By using the global metadata, applications can adapt themselves to a given database. Global metadata can be defined per product, per update region, per *administrative area*, or as common metadata applying to several building blocks.

For more information on global metadata, refer to Section 20.2 *Common Data* on page 352.

- Building block-specific metadata: The building block-specific metadata refer to one specific building block.

For more information on building block-specific metadata, refer to the metadata sections in the corresponding building block chapters.

## 3.4 Introduction to NDS DataScript

Navigation Data Standard (NDS) uses *DataScript*, a formal language for modelling binary data types, bit streams, or file formats. Using a formal language for defining binary data types resolves all ambiguities typically found in textual or tabular specifications. It is also possible to automatically generate encoders and decoders for a given binary format from the formal specification, so that application developers do not have to think about serialization and can focus on application logic instead.

The *NDS DataScript* implementation bases on Godmar Back's reference implementation. However, to support the specific NDS requirements, NDS e.V. branched off its own DataScript tools to add language extensions.

NDS e.V. enhanced the DataScript language with relational extensions. This is called *Relational DataScript (RDS)*. RDS 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.

Data types in DataScript are defined recursively. The following DataScript types are available:

- A primitive type, for example, a bit field or an integer
- A set type, which can be either an enumerated type or a bitmask type
- A linear array of a type, which uses integer indices starting with index 0
- A composite type, which can either be a record type or a variant-record type

A constraint can be assigned to composite type fields. The constraint has to be specified as an arbitrary Boolean predicate separated from the field name by a colon. Constant fields are a special form of constraints which can be written like an initializer in C and Java.

A DataScript compiler operates on a set of DataScript files and transforms them into a target language, for example, C++ or Java. The generated target classes usually contain code portions for serialization and deserialization which makes the coding of input and output routines unnecessary.

## Numbers

On the deepest level of abstraction, NDS data is a sequence of bits. One abstraction level above, NDS data consists of a sequence of signed and unsigned binary numbers of different bit width. NDS features are, therefore, represented by a collection of numbers, which are represented by bits.

An  $n$ -bit binary number  $N$  consists of bits  $b_{n-1} \dots b_0$ , where  $b_{n-1}$  is the most significant bit and  $b_0$  is the least significant bit. There are two possibilities to interpret the bits of  $N$ :

- As an unsigned value rendering  $N$  an unsigned number
- As a signed value rendering  $N$  a signed number

The unsigned value of  $N$  is formally defined as the integer:

$$uval(N) = \sum_{i=0}^{n-1} b_i \cdot 2^i$$

For example,  $uval(10000010) = 130$ . By its definition, an  $n$ -bit unsigned number can represent the integer values from 0 to  $2^n - 1$ . It is possible to extend a given  $n$ -bit unsigned number  $N = b_{n-1} \dots b_0$  to an  $m$ -bit unsigned number  $M$  with  $m > n$ , by adding  $m - n$  zero bits on the most significant side. That is,  $M = 0 \dots 0 b_{n-1} \dots b_0$ . The reverse operation, converting to an unsigned number with less bits, is only possible if the unsigned value fits the numeric range of the target number, that is, only leading zeros are discarded.

To represent negative integers, a *two's complement representation* is used. Thus, the signed value of  $N$  is formally defined as the integer:

$$sval(N) = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i$$

For example,  $sval(10000010) = -126$ . Bit  $b_{n-1}$  is often called *sign bit*. While it is true that the value of  $b_{n-1}$  determines the sign of  $sval(N)$ , the two's complement representation is different from a representation using one bit for the sign and  $n-1$  bits for the absolute value. By its definition, an  $n$ -bit signed number can represent the integer values from  $-2^{n-1}$  to  $2^{n-1}-1$ . Converting an  $n$ -bit signed number  $N = b_{n-1} \dots b_0$  to an  $m$ -bit signed number  $M$  with  $m > n$  is done by replicating the sign bit. That is, if  $b_{n-1} = 0$  then  $M = 0 \dots 0 b_{n-1} \dots b_0$ , and if  $b_{n-1} = 1$  then  $M = 1 \dots 1 b_{n-1} \dots b_0$ . The converse operation is only possible, if the signed value fits into the target range.

Programming languages typically offer signed and unsigned numbers of certain bit widths, which are all multiples of eight ( $n = 8, 16, 32, 64$ ). This reflects the capabilities of modern CPUs. NDS, however, may use any bit width for numbers between 1 and 64, mainly to save space. During parsing of the data, an application has to extend the numbers to the next suitable bit width offered by the programming language. This application has to take into account whether the number is signed or unsigned. For example, the 5-bit number 10111 is converted into the byte 00010111 if it is an unsigned number and to the byte 11110111 if it is a signed number.

DataScript stores data as bit stream. Typical programming languages and operating systems, however, offer only byte streams. NDS maps the DataScript bit streams to byte streams by using the big-endian approach, which stores the most significant bits first (also refer to <http://datascript.berlios.de/DataScriptLanguageOverview.pdf> > 3 Base Types > 3.1 Integer Base Types). For example, the 16-bit number 1111111100000000 is stored in the two bytes  $b_0 = 11111111$  and  $b_1 = 00000000$ .

It might happen that less than eight bits must be transferred to the subsequent byte, either because the number has a bit width that is not divisible by eight, or because it starts at a bit position that is not at a byte border. In any case, the least significant bits that do not fit into the first byte are stored as most significant bits in the subsequent byte. Take, for example, the sequence of the 10-bit number 1011111111 and the 6-bit number 000000. They are concatenated in the DataScript bit stream and, thus, stored as bytes  $b_0 = 10111111$  and  $b_1 = 11000000$  in a byte stream.

#### Related Topics:

- <http://people.cs.vt.edu/~gback/papers/gback-datascript-gpce2002.pdf>
- <http://datascript.berlios.de/DataScriptLanguageOverview.pdf>
- <http://datascript.sourceforge.net/>
- <http://datascript.berlios.de/>
- <http://en.wikipedia.org/wiki/Endianness>
- [http://en.wikipedia.org/wiki/Two%27s\\_complement](http://en.wikipedia.org/wiki/Two%27s_complement)

### 3.5 Extending the Navigation Data Standard

Navigation Data Standard (NDS) defines a set of features that are needed for all maps and provide a stable basis for navigation systems with optional features to meet additional market requirements. The navigation system market, however, is diverse and driven by different traditions of the car manufacturers, market segments, region-specific requirements, as well as technologies and innovation.

To ensure flexibility to meet diverse requirements coming from the market and the database suppliers and to allow for the implementation of future developments and features for navigation systems, NDS supports the following extensions to the standard:

- *Enhancement*: NDS extension which has been submitted to NDS e.V. for integration into the standard. For the transition period (before integration is complete), the enhancement may be added to NDS databases by means of the *NDS transparent enhancement approach*.

See also *Data Structures for Enhancements of the Standard* on page 57.

- *Proprietary add-ons* for features and technologies outside the standard, for example, precompiled data structures for speech recognition

- *Adaptation*: Variants of NDS facilitating specific project requirements during the grace period.

The rules for adaptations are defined in *NDS – Certification Specification*, 1.2.1 *Extensions to the NDS Standard* on page 7.

## Rules for Extensions

To protect the standard and to ensure interoperability, extensions to NDS need to comply with the following rules:

- Respecting the standard: Extensions may enhance, but not replace standard functionality or data structures as specified in the *NDS – Format Specification*.
- Supporting updates: Extensions shall support update processes for navigation data as specified in the *NDS – Update Specification*.
- Ensuring interoperability: Extensions shall comply with the interoperability concepts specified in the *NDS – Compiler Interoperability Specification*. The impact of extensions on systems not using the extension shall be minimized. Neither functionality nor performance shall be affected by the extensions.

---

### Note

The rules apply to both proprietary add-ons and enhancements of the standard. They may, however, be relaxed for enhancements of the standard upon good cause shown and under the condition that the deviations from the rules are handled with great care. This is ensured by the Technical Committee of NDS e.V., which must approve all deviations from the rules.

---

## Impact of Extensions

Extensions influence the applications in several ways:

- Database size
- Seek times (for example, because of additional head movements)
- Data access times
- RAM requirements
- Processing times
- Production and validation costs

## Integration of Extensions

A close integration of new features has a positive impact on efficiency and reduces the overhead of additional references. The increase of the size, however, must be limited. New features can be stored in additional BLOBs either in existing tables or in new tables.

To keep the production and validation costs low and to ensure interoperability, the following rules for the integration of proprietary add-ons apply:

- Data must be stored in new tables in separate database files.<sup>1)</sup>
- References to standard features must use the existing NDS reference mechanisms.
- References from NDS features to extension features are not allowed, as they would require the extension of existing BLOBs.

It shall be possible for applications to work properly without the proprietary add-ons.

## Data Structures for Enhancements of the Standard

Additional data used for enhancements of the standard is stored separately in the Extension building block (building block type EXTENSION). To indicate the availability of additional data for features, the following attributes are introduced (DataScript location: `nds.common.flexibleattributes`):

- **HAS\_ADDITIONAL\_DATA**: This attribute indicates that additional data is available for the feature, which is referenced by the attribute. The attribute stores the ID of the Extension building block as a value (`buildingBlockId`). The application can use the building block ID to load the respective record from the `BuildingBlockTable`, if required, or ignore the additional data.
- **HAS\_ADDITIONAL\_DATA\_ELEMENT\_ID**: This attribute provides an identifier, which differentiates between data elements in the Extension building block. Thus, the application can query a specific SQL table in the Extension building block.

The attribute contains the `additionalDataElementId`, which identifies the `buildingBlockId` of the Extension building block and the `dataElementId` of the respective data element in the Extension building block.

---

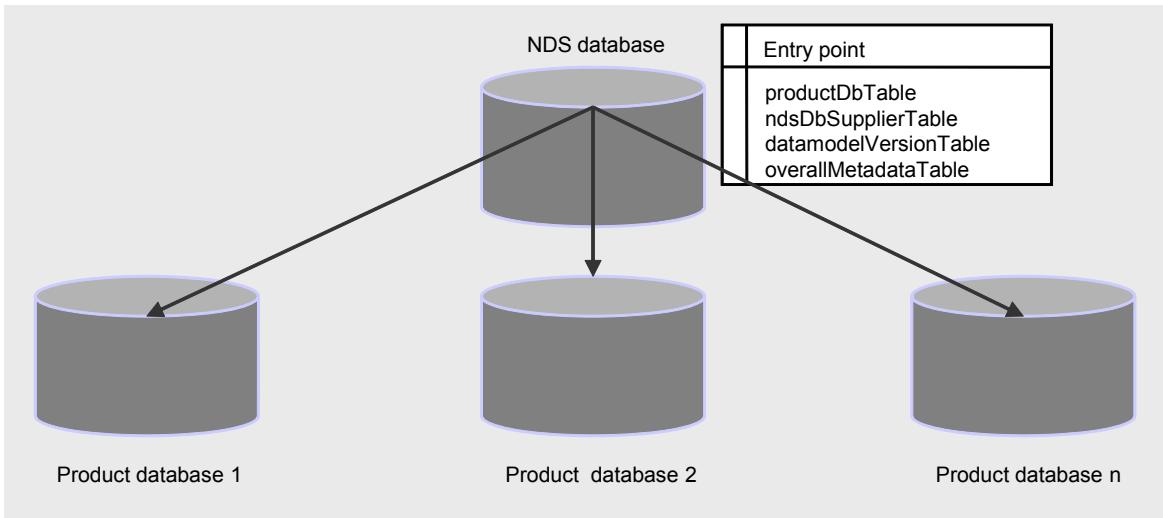
1) This rule may be relaxed on good cause shown if a closer integration is indispensable to achieve reasonable performance results. Upon request, the Technical Committee of NDS e.V. decides about deviations from this rule. Only deviations from this rule that have been approved formally by the Technical Committee may be used in NDS products.



## 4 Database Structure

All databases that are compliant with NDS must use the file name extension NDS. The entry point to an NDS database is the product database table (**ProductDbTable**). This table is stored in the data carrier's root directory in the **ROOT.NDS** file.

*Figure 4-1 Entry point to an NDS database*



### 4.1 Product Database Table

The product database table (**ProductDbTable**) lists all product databases that are currently present in a given NDS database.

The version entry in the **ProductDbTable** refers to the filled **VersionTable** of the corresponding product. **ROOT.NDS** therefore only contains a filled version table, if a single file database is used with one product only.

- 
- |             |  |
|-------------|--|
| <b>Note</b> | A product database shall be updated only by the provider who originally compiled it. The relational tables of all building blocks that are available in an NDS database must be created and registered in the metadata. This also applies to empty tables. |
|-------------|--|
- 

For each product database, the product database table contains a Uniform Resource Identifier (URI) pointing to the product database's SQLite file which contains detailed information.

The URI notation is compliant with RFC 3986, see <http://rfc-ref.org/RFC-TEXTS/3986/index.html>. It uses the general format:

<schema>://<server>/<path-segment-1>/<...>/<path-segment-x>/<filename>.NDS

The URI shall be compatible with the FAT32 file system convention without using the VFAT extension. Path segments shall use a maximum of eight characters plus three characters for the file name extension NDS. For ensuring compatibility between different operating systems, only a subset of the characters allowed for FAT32 is allowed for the names of directories and files: Upper case ASCII letters (A-Z), digits (0-9), underscores (\_), the at sign (@), and plus (+) and minus (-) shall be used. The following names must not be used, because they are resolved as communication ports in DOS systems: CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9. Also avoid these names followed immediately by an extension; NUL.TXT for example, is not allowed.

The URI specifying the SQLite file of the product includes the name of the SQLite file for the product database metadata.

---

<b>Note</b>	<p>The file name of the entry point (<b>ROOT.NDS</b>) and the file name extensions (*.NDS) are predefined. In case of multiplexed database files, the chunks are numbered by using the file name extension as a three digit counter (.001,.002,...). The file names for the building blocks are not predefined.</p> <p>For describing the locations of NDS database files on the local data carrier, relative URIs are the preferred solution as they allow copying files and directories without the need to update the metadata. The location of remote resources (future use) must be described by absolute URIs.</p> <p>If a relative URI is used, the URI is always relative to the location of the NDS database file in which the URI is stored.</p>
-------------	--

---

Table 4-1 lists the most important fields of the product database table (DataScript location: **nds.overall.version > ProductDbTable**).

Table 4-1 Fields of the product database table

Field	Description
<b>productId</b>	Unique identifier for an NDS product database from a specific NDS database supplier. Set by the NDS database supplier. Together with the <b>ndsDbSupplierId</b> , the <b>productId</b> uniquely identifies an NDS product.
<b>productName</b>	Human-readable name assigned to a product database by the NDS database supplier
<b>ndsDbSupplierId</b>	Unique identifier for a party that compiles and supplies data The identifier is defined and maintained by NDS e.V., see <i>NDS – Format Specification, B NDS Database Supplier IDs</i> on page 369.
<b>baselineMapId</b>	Unique identifier for the single root of a generation of NDS products and updates that has been compiled by an NDS database supplier. This root is called <i>baseline map</i> .
<b>versionId</b>	Uniquely denotes a consistent database status from a particular NDS database supplier

Field	Description
isDirty	Indicates whether the product database has been updated When this indicator is set to <code>true</code> , not all database elements have been updated.
isOverviewMap	Indicates whether the product database is an overview map, see <i>World Overview Map Product Database</i> on page 61
uri	Uniform Resource Identifier pointing to the product database's SQLite file
copyright	Includes a legal copyright statement for this product database; this statement can be presented to the user of the system, for example via the user interface.

### World Overview Map Product Database

The world overview map is delivered as a separate optional product database. It is used for showing the whole world, or large sections of the world map. A flag indicates whether or not a product database represents a world overview map.

The world overview map product database usually contains the Basic Map Display, Name and POI building blocks. Routing data is not allowed for the world overview map product database.

The world overview map product database shall only include levels 0, 1, 2, and 3. Which levels are filled is indicated in the metadata of the corresponding building blocks of the product database. The corresponding `minScaleDenominator` and `maxScaleDenominator` have to be set for the world overview map product database.

For more information, see *NDS – Update Specification*, 3.2.3 *World Overview Map* on page 32.

## 4.2 NDS Database Supplier Table

The NDS database supplier table (`NdsDatabaseSupplierTable`) lists all identifiers as officially assigned by NDS e.V. to parties that produce NDS databases. The table is filled at compile time with all known values. It may be extended via updates if data from new parties is installed in the system.

Table 4-2 lists the fields of the NDS database supplier table (DataScript location: `nds.overall.version > NdsDatabaseSupplierTable`).

Table 4-2 Fields of the NDS database supplier table

Field	Description
<code>ndsDbSupplierId</code>	Unique identifier for a party that compiles and supplies data in NDS format; defined and maintained by NDS e.V.
<code>ndsDbSupplierName</code>	Human-readable name for the party with the given ID This name can be presented to the user, for example, via a user interface.

For the current list of NDS database suppliers, see Appendix B *NDS Database Supplier IDs* on page 369.

### 4.3 Data Model Version Table

The data model version table (`DataModelVersionTable`) lists the available versions of DataScript definitions for the NDS format used in this particular database.

The `DataModelVersionTable` applies to all products on an NDS data carrier and therefore needs to be filled in `ROOT.NDS`.

The flag `hasAdaptations` indicates whether variants of NDS are available in the database to facilitate specific project requirements. For more information about adaptations to the standard, refer to *NDS – Certification Specification*, 1.2.1 *Extensions to the NDS Standard*.

When the flag `hasAdaptations` is not set (=0), an application can rely that the database is fully compliant to the indicated NDS version.

### 4.4 Overall Metadata Table

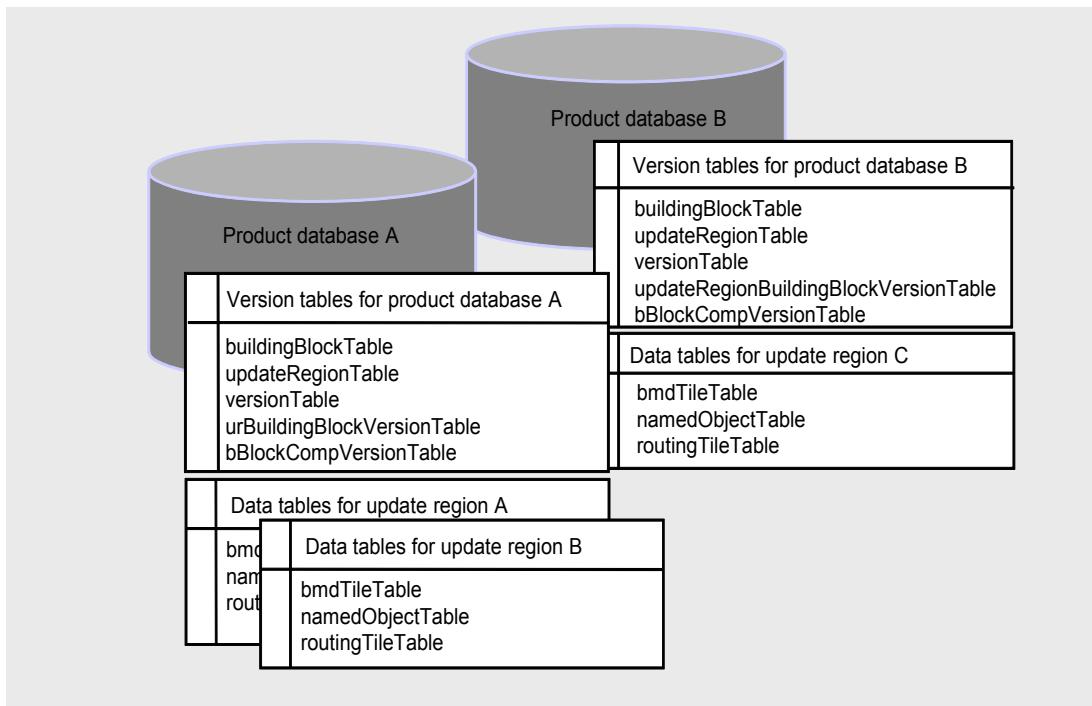
The overall metadata table (`OverallMetadataTable`) defines the DataScript model version for all tables included in the SQLite file. It is based on the list of available versions of DataScript definitions in the `DataModelVersionTable` and contains only one row. An instance of this table must be present in each NDS SQLite file.

### 4.5 Structure, Data, and Metadata of Product Databases

A product database (`sql_database ProductDatabase`) contains several tables. Tables are either metadata tables with information describing the particular product, or data tables with the actual NDS features.

Figure 4-2 illustrates the tables contained in a product database.

Figure 4-2 Table structure for a product database, example



## Data Tables

The data tables contain the NDS features of the individual building blocks. For detailed information, refer to the chapters describing the building blocks.

## Metadata Tables

Table 4-3 gives an overview of the tables, in which product-specific metadata is stored.

Table 4-3 Tables containing metadata for a product database

Metadata	Description
<b>BuildingBlockTable</b>	Specifies, which building blocks are present in a product database DataScript location: <code>nds.overall.productdbversion</code> For more information, refer to Section 4.5.2 <i>Building Block Table</i> on page 65.
<b>UpdateRegionTable</b>	Contains the update region definition for the database DataScript location: <code>nds.overall.productdbversion</code> For more information, refer to Section 4.5.1 <i>Update Region Table</i> on page 65.
<b>TileContentIndexTable</b>	Specifies available tiles for building blocks as a list of content indices per level DataScript location: <code>nds.overall.productdbversion</code> For more information, refer to Section 7.3.3 <i>Content Indices for Update Regions</i> on page 99.

Metadata	Description
<code>UrBuildingBlockVersionTable</code>	<p>Uniquely describes the version of a particular building block instance within a given update region</p> <p>DataScript location: <code>nds.overall.productdbversion</code></p> <p>For more information, refer to Section 4.5.3 <i>Update Region Building Block Version Table</i> on page 66.</p>
<code>VersionTable</code>	<p>Stores all version identifiers that have been or are in use within a given product database</p> <p>DataScript location: <code>nds.overall.productdbversion</code></p> <p>For more information, refer to Section 4.5.4 <i>Version Table</i> on page 67.</p>
<code>BBlockCompVersionTable</code>	<p>Specifies the version of components within a specific building block that is available in an update region</p> <p>DataScript location: <code>nds.overall.productdbversion</code></p> <p>For more information, refer to Section 4.5.5 <i>Building Block Component Version Table</i> on page 68.</p>
<code>CipherKeyInfoTable</code>	<p>Contains the cipher key information that is used for encryption</p> <p>DataScript location: <code>nds.overall.drm</code></p> <p>For more information, refer to Section 8.3 <i>Cipher Key Info Structure</i> on page 104.</p>
<code>EncryptableItemCodeTable</code>	<p>Contains the encryptable item codes for encryption</p> <p>DataScript location: <code>nds.overall.drm</code></p> <p>For more information, refer to Section 8.4 <i>Encryptable Item Code</i> on page 105.</p>
<code>ColorTable</code>	<p>Contains the color definitions for the database, identified by a color ID (<code>colorId</code>); colors are defined with RGB values and alpha channel</p> <p>DataScript location: <code>nds.overall.metadata</code></p>
<code>FontTable</code>	<p>Contains font information that is required to render text, identified by a font ID (<code>fontId</code>); the table contains information about the name, size, and color for each font ID</p> <p>The <code>genericFontId</code> defines the fallback, if an application does not support the specified font.</p> <p>DataScript location: <code>nds.overall.metadata</code></p>
<code>LanguageTable</code>	<p>Defines the languages used in the database</p> <p>DataScript location: <code>nds.overall.metadata</code></p> <p>For more information, refer to Section 4.5.6 <i>Language Table</i> on page 70.</p>

### 4.5.1 Update Region Table

The **UpdateRegionTable** uniquely describes a particular region in a product database that can be subject to an update. For general information about Updates, refer to *NDS – Format Specification, 3.2.2 Update Region* on page 45.

Table 4-4 lists the metadata stored in the **UpdateRegionTable**.

DataScript location: `nds.overall.productdbversion`

*Table 4-4 Fields of the update region table*

Field	Description
<code>updateRegionId</code>	Unique identifier of a particular update region within a product database that remains constant over the lifetime of the corresponding <i>baseline map</i> ; defined and maintained by the NDS database supplier  This implies that the IDs for all update regions are assigned at compile time and that these IDs remain constant. IDs of update regions that are added by map expansion later on shall not violate the uniqueness constraint.
<code>updateRegionNamedObjectRef</code>	Reference to the named object representing a human-readable name of an update region
<code>nvcStringSeparationCharacter</code>	Defines the string that is used to separate an NVC string for NVC view and list view
<code>versionId</code>	Identifier of the latest consistent update executed on the particular update region
<code>isDirty</code>	Indicates whether an update region has been updated  When this indicator is set to TRUE, not all database elements have been updated. For more information, refer to <i>NDS – Update Specification, 5.2 Subversion and Version Control for NDS</i> on page 40.
<code>mapDataReleaseDate</code>	Gives the release date of the update region map data, which can be displayed to the customer, for example, 2010/Q4
<code>releaseYear</code> <code>releaseMonth</code>	Indicates the release year/month of map data in a machine-readable form

### 4.5.2 Building Block Table

The **BuildingBlockTable** describes, which building blocks are available in the given product database. It is filled at compile time with information about all building blocks which are available, and is extended during the update process if additional building blocks are installed.

Table 4-5 lists the metadata stored in the **BuildingBlockTable**.

DataScript location: `nds.overall.productdbversion`

Table 4-5 Fields of the building block table

Field	Description
buildingBlockId	Unique identifier of an instance of a building block type within a product database  There may be multiple building block instances of the same type, for example, of the POI building block.
buildingBlockName	Human-readable name of a building block instance assigned by the NDS database supplier
buildingBlockType	Uniquely defines a building block type
buildingBlockDetailedType	Defines a more detailed type of the building block, for example, the phonetic transcription format for the Speech building block  The field may be empty (default).

### 4.5.3 Update Region Building Block Version Table

The UrBuildingBlockVersionTable uniquely defines the version of a particular building block instance within a given update region.

Table 4-6 lists the metadata stored in the UrBuildingBlockVersionTable.

DataScript location: nds.overall.productdbversion

Table 4-6 Structure of the update region building block version table

Field	Description
updateRegionId	Represents a unique identifier of a particular update region of a product database that remains constant over the lifetime of the corresponding <i>baseline map</i> ; defined and maintained by the NDS database supplier  This implies that the IDs for all update regions are assigned at compilation time, and that these IDs remain constant. IDs of update regions that are added by map expansion later on shall not violate the uniqueness constraint.
buildingBlockId	Unique identifier of an instance of a building block type within an update region; there may be multiple building block instances of the same type, for example, of the POI building block
versionId	Identifier of the most recent consistent update applied to the particular update region
isDirty	Indicates whether an update region has been updated  When this indicator is set to TRUE, not all database elements have been updated.
isPartiallyFilled	Indicates whether the building block within the update region is only partially filled  For more information on partially filled update regions, refer to <i>NDS – Update Specification</i> , 3.2.1 <i>Partially Filled Update Regions</i> on page 27.

Field	Description
uri	<p>Uniform Resource Indicator pointing to the SQLite file which contains detailed data for the given building block in the given update region</p> <p>The URI specifying the SQLite file of the building block is defined as an absolute or relative address, and includes the name of the SQLite file that contains the data.</p>
	<p><b>Note</b> Per default, all tables of a building block are located in the building block's SQLite file(s) specified by the URI. However, individual tables may be located in different files. In such a case, the table's URI shall be specified in the building block component version table, see 4.5.5 <i>Building Block Component Version Table</i> on page 68.</p>
useFileEncryption	Indicates whether file encryption is used for this database file NULL indicates that the file is not encrypted.
useBlobEncryption	Indicates whether BLOB encryption is used NULL indicates that the BLOB is not encrypted.
eic	<p>Encryptable item code used for construction of the initialization vector applied to the encryptable item</p> <p>For more information, refer to Chapter 4 <i>Encryption</i> on page 71 and <i>NDS – Format Specification, 8 Encryption</i> on page 101.</p>
dataModelVersionId	Identifies the DataScript specification version of the content layer definition
buildingBlockMetadata	Stores metadata for the respective building block of this update region

#### 4.5.4 Version Table

The **VersionTable** stores all version identifiers that have been or are in use within the given product database.

Table 4-7 shows the metadata stored in the **VersionTable**.

DataScript location: `nds.overall.productdbversion`

Table 4-7 Fields of the version table

Field	Description
versionId	Uniquely identifies a consistent database status from a particular NDS database supplier of the given product database
versionName	Human-readable name of the version, assigned by the NDS database supplier
compilerVersion	Plain text name of the Compiler used by the NDS database supplier
compilerConfiguration	Plain text description of the Compiler configuration

Field	Description
creationDateTime	<p>Creation date and time in UTC when the version was created The format must be compliant to <a href="http://en.wikipedia.org/wiki/ISO_8601">http://en.wikipedia.org/wiki/ISO_8601</a>, combined date and time representations. A navigation system can use this information to display status information via the HMI. Example:</p> <p>YYYY-MM-DDThh:mm:ssZ 2011-07-18T13:24:19Z</p>
dataModelVersionId	<p>DataScript specification version of the content layer definition The value must correspond to one of the <code>dataModelVersionId</code> values in the <b>DataModelVersionTable</b> (see <i>Data Model Version Table</i> on page 62).</p>

- Single data model version per NDS database: The data model version column of the version table is filled with the same value for all entries.
- Multiple data model versions per NDS database: Versions may be associated with different data model versions.

**Example** In tile-based building blocks, tiles with the same version also have the same data model version.

Tiles with different version numbers, however, may either have the same data model version (several versions are linked to the same data model), or different data model versions (versions are linked to different data models).

#### 4.5.5 Building Block Component Version Table

The **BBlockCompVersionTable** specifies the version of components within a specific building block that is available for an update region. This is usually a particular table or a column within a table.

**Note** The **BBlockCompVersionTable** is mandatory for tile content indices and used for those tables, which are referred in the respective tile content indices.

**Example** The **GlobalGatewayTable** is part of the Routing building block. However, it is global thus only one such table is needed for all route building blocks of all update regions in a particular product. Thus it makes sense to store this table on a place central for the product: uri = product1/product1.nds

The building block component version table (`BBlockCompVersionTable`) uniquely describes the version of a particular component of a building block instance within a given update region. A component is either a relational table, or a column within a relational table. The building block component version table also describes the location of tables via an URI column, if the location deviates from the location of the complete building block as described by the update region building block version table. See also *NDS – Compiler Interoperability Specification*, 2.1 *Distribution of Tables over SQLite Files* on page 39.

Table 4-8 lists the fields of the update region building block version table (DataScript location: `nds.overall.productdbversion > BBlockCompVersionTable`).

*Table 4-8 Fields of the building block component version table*

Field	Description
<code>updateRegionId</code>	Unique identifier of a particular update region within a product database that remains constant over the lifetime of the corresponding baseline map  This implies that the IDs of all update regions are assigned at compilation time and that these IDs remain constant. IDs of update regions that are added by map expansion later on shall not violate the uniqueness constraint.
<code>buildingBlockId</code>	Unique identifier of an instance of a building block type within an update region; there may be multiple building block instances of the same type, for example of the POI building block
<code>tableName</code>	Name of the instance of the table to be versioned, for example, <code>GlobalGatewayTable</code>
<code>columnName</code>	Name of the column to be versioned, in case versioning for columns is used. If the column is not versioned, an empty string shall be given as <code>columnName</code> ; NULL must not be used, as the column name is part of the primary key.
<code>versionId</code>	Identifier of the most recent consistent update executed on the particular table or column
<code>isDirty</code>	Indicates whether the table or column has been updated When this indicator is set to TRUE, not all records have been updated.
<code>uri</code>	Uniform Resource Indicator pointing to the SQLite file which contains the instance of the table. The URI is defined as an absolute or relative address and includes the name of the SQLite file that contains the data.  This field shall be empty when the location of the table is the same as the location of the containing building block. An URI shall be specified if the table is to be found in an alternative SQLite file. An example is the <code>GlobalGatewayTable</code> that may be stored in another SQLite file than the Routing building block.  If the URI is provided as a relative address it must be relative to the location of the SQLite file that contains the <code>BBlockCompVersionTable</code> .

#### 4.5.6 Language Table

All languages supported in a product database are defined in a language table (DataScript location `nds.overall.metadata > LanguageTable`). The table maps the combination of ISO 3166-1 alpha 3 country code (upper case), ISO 639-3 language code (lower case) and ISO 15924 alpha 4 script code (first letter upper case, other three letters lower case) to a small integer value. This value is used to identify the language of name strings, signposts, POI names, and many more names. This mechanism makes it possible to avoid software updates for adding new languages.

The table also contains the codes for transliteration languages, for example country code (GRC), language code (grc), and script code (Latn; Greek transliterated in Latin). In case of transliterated languages, the ISO language code and the ISO country code stay the same, but the script code is changed.

The language code `UNDEFINED_LANGUAGE_CODE` is defined as a constant and is not stored explicitly in this table. It is used as a placeholder for unknown and undefined languages, or for strings that cannot be assigned to a particular language such as numbers.

---

**Note** NDS uses XXX as ISO country code for international waters. This country code shall, however, only be used in the region metadata.

---

The language table exists only once per product database and is valid for all building blocks in this product database.

---

**Note** The language codes used for one product must be unique across update regions.

---

#### Language Names

As the language codes shall not be displayed to the user, the language names are stored in all available languages for language selection. An example is shown in Table 4-9.

---

**Note** Not all names for supported languages need to be translated. Some languages will not be displayed to the user, for example, transliteration languages.

---

Table 4-9 Language definition table, example

Language code	German translation	English translation	Chinese translation
1 (DEU, deu, Latn)	Deutsch	German	德语
2 (GBR, eng, Latn)	Englisch	English	英语
3 (CHN, cnm, Hani)	Mandarin (vereinfacht)	Mandarin Chinese (simplified)	汉语

## Transliteration Indicator

There are language codes for languages representing transliterations of other languages, for example, 4 (GRC, grl, Latn for Greek transliterated in Latin). In addition to the language code, these languages are marked with the transliteration indicator. Language code 3, for example, represents Greek and language code 4 represents Greek transliterated in Latin. In this case, the attribute `isTransliterationOf` is assigned to language code 4 with the value language code 3.

This relation between languages is required for the following reasons:

- Filtering of transliterations in location input (NVC trees), map display, route lists, and so on, to cover customer-specific requirements
- Supporting new transliteration language codes without the need to update the navigation software
- Supporting interoperability for transliterations without standardizing language codes

---

<b>Note</b>	To enable the distinction of different transliterations, it is necessary to use different language codes for different transliterations. If, for example, a Latin and a Cyrillic transliteration of Greek names is available, each of them needs its own language code.
-------------	---

---

## Diacritic Transliteration Indicator

A diacritic (diacritical mark, diacritical point, or diacritical sign) is an ancillary sign added to a letter, or a basic sign indicating an altered pronunciation. Diacritics may also be used for distinguishing similar words. For further information on diacritics, refer to <http://en.wikipedia.org/wiki/Diacritic>.

The diacritic transliteration indicator defines a language as a diacritic transliteration of another language. Language code 6, for example, represents Czech and language code 7 represents Czech transliterated in Latin, which basically means that the diacritics have been removed. In this case, language code 7 has the attribute `isDiacriticTransliterationOf` with value language code 6.

This relation between languages is needed due to the requirements outlined in Section *Transliteration Indicator* on page 71, and the fact that diacritic transliterations are possibly not to be displayed, because the similarity to other languages is high (for example, the British and Germans can read Czech characters, because they are similar to Latin).

---

<b>Note</b>	Different language codes must be used for different transliterations (see Section <i>Transliteration Indicator</i> on page 71).
-------------	---

---

## Character Code Chart

Character code lists are defined in Unicode to represent Latin-1, Hiragana, etc. The character code lists are stored in metadata and indicate the characters that are available for a language in order to allow the application to suppress characters that are not supported by the HMI and, hence, provide a useful error reporting to the user.

Character code lists also indicate the target alphabet of transliterations. This is because ISO language codes for transliterations are not always defined. GRL, for example, can represent Greek transliterated in Latin, but as this is not standardized in ISO, the character code list points to Latin-1.

## 4.6 Logical Mapping of Tables to Building Blocks

NDS differentiates between the logical and physical structure of an NDS database. Whereas the physical file structure of an NDS database is flexible, and can be defined individually by each database supplier, the logical structure explicitly defines the relation between database tables and building blocks.

The logical structure is defined in DataScript as follows:

- An `sql_database` structure is available for each building block. It contains all tables that logically belong to the respective building block, but may physically be stored in different files.
- The `sql_database` structure defines also the (metadata) tables of the `ROOT.NDS` file, and the tables and metadata tables of a product database being outside of a building block.

The definition of a logical structure facilitates the automatic validation of NDS databases as well as the automatic updating of tables belonging to different building blocks. The logical structure, however, does not interfere with the flexible file structure of an NDS database as described here and in *NDS – Compiler Interoperability Specification, 2.1 Distribution of Tables over SQLite Files* on page 39.

## 5 Overall Building Block: Update Region Metadata

The Overall building block stores the data that is common to all building blocks in the database. An application uses this data to retrieve information on variable database content and database properties, and, thus, to adapt itself to a given database.

The data in the Overall building block is defined for a specific update region or a specific region within an update region.

---

<b>Note</b>	The metadata specific for a product is stored in the product database ( <code>sql_database ProductDatabase</code> ).  For more information on product-specific metadata, refer to Section 4.5 <i>Structure, Data, and Metadata of Product Databases</i> on page 62.
-------------	---

---

The following data belongs to the Overall building block:

- Level metadata
- Region metadata
- Region-specific data, such as road numbers and prefixes, address formats, and speed profiles
- Additional road icons
- Time zone data

### 5.1 Level Metadata

The `LevelMetadataTable` contains the metadata for a given level and is stored per update region. The level metadata defines which data is available on this level.

DataScript location: `nds.overall.metadata`

Table 5-1 Level metadata

Metadata	Description
<code>levelNumber</code>	Defines the tile level number
<code>hasOrthoimageTiles</code>	Specifies if orthoimage tiles are available
<code>hasBmdTiles</code>	Specifies if basic map display tiles are available
<code>hasRoutingTiles</code>	Specifies if routing tiles are available
<code>hasTmcTiles</code>	Specifies if traffic information tiles are available
<code>hasDtmTiles</code>	Specifies if digital terrain model tiles are available
<code>hasPoiVirtualTiles</code>	Specifies if POI virtual tiles are available

Metadata	Description
dtmImageFormat	Defines the image format for digital terrain model data
dtmImageHeightResolution	Defines the height resolution for images
coordShift	Defines the coordinate shift that has to be applied to all coordinate offsets; the coordinate shift also implies the coordinate width
maxScaleDenominator	Maximum map scale denominator; equivalent to the smallest map scale recommended for the specified level
minScaleDenominator	Minimum map scale denominator; equivalent to the largest map scale recommended for the specified level
scaleDenominatorList	Lists all scale denominators for the scale sublevels of the specified level

## 5.2 Region Metadata

The RegionMetadataTable contains country- or state-specific metadata, such as right- or left-hand traffic. To find out, which region-specific metadata applies to a link, the application can use the following approach:

- If the update region has only one region definition, the `regionId` for this region is applied to all links belonging to the update region.
- If the update region has more than one region definition, the region of the tile determines the region, to which the links located in the respective tile belong (`METADATA_REGION_REFERENCE` attribute).
- If a tile has more than one region definition, the application evaluates the reference to the `METADATA_REGION_REFERENCE` attribute for each link in the tile.

DataScript location: `nds.overall.metadata`

Table 5-2 Region metadata

Metadata	Description
<code>regionId</code>	Identifies the region
<code>regionNamedObjectReference</code>	An update region may contain one or more regions; each region is identified by its region named object reference
<code>hasRightHandDriving</code>	Specifies whether traffic has to keep to the right or the left side of the road

Metadata	Description
<code>hasMetricSystem</code>	<p>Indicates whether a region uses the metric or imperial system of measurement; if set to FALSE, the region uses the imperial system. As link lengths are stored in centimeters only, the system of measurement indicates the default value for converting and displaying the route length to the user. Conversion is not required for other lengths like heights or widths on traffic signs, as these are stored in one of the measuring systems according to the real-world values.</p> <p>For exceptions from this default system of measurement for a region, the flexible attribute <code>KM_MI_INDICATOR</code> (DataScript location: <code>nds.common.flexibleAttributes</code>) can be used.</p>
<code>isoCountryCode</code>	<p>Specifies the ISO country code according to ISO 3166-1 alpha 3 code in three upper case characters</p> <p>The ISO country code is used to determine same regions for different building blocks and to map the country specifics within an application (for example different drawing styles per country). For international waters, NDS uses ISO code XXX.</p>
<code>isoSubCountryCode</code>	<p>ISO 3166-2 code defining the names of a country's main subdivisions, such as provinces or states</p>
<code>characterCharCodeColl</code>	<p>Defines the collection of Unicode character code charts that are available for the specified region, such as Latin-1 or Greek.</p> <p>Applications can use this data to retrieve information on available characters and, consequently, to suppress characters not supported by the fonts available in the navigation system. This facilitates useful error reporting to the user.</p> <p>The code of an available UTF-8 character chart is encoded by the first UTF-8 character of the chart. Latin-1, for example, is encoded by 0x0080.</p>
<code>usedAdminRoadClasses</code>	<p>Specifies the administrative road classes that are used in the specified region</p>
<code>groupedAttributeList</code>	<p>List of flexible attribute maps; see section <a href="#">5.2.1 Flexible Attribute Maps</a> on page 76</p>
<code>hasAdasBasicTypes</code>	<p>Indicates that basic ADAS types are available in the database, such as <code>PHYSICAL_WIDTH_METRIC</code>, <code>CURVATURE</code>, or <code>SLOPE</code> (see <a href="#">9.6 Data Structures for Advanced Driver Assistance (ADAS)</a> on page 134)</p>
<code>hasAdasExtendedTypes</code>	<p>Indicates that extended ADAS types are available in the database, such as <code>EXTENDED_CLOTHOID_META_DATA</code> and <code>EXTENDED_CLOTHOID</code> (see <a href="#">9.6 Data Structures for Advanced Driver Assistance (ADAS)</a> on page 134)</p>
<code>hasAdasBezierSpline</code>	<p>Indicates that ADAS types for Bezier splines are available in the database, that is, either <code>PATH_VECTOR_3D</code> or <code>SPLINE</code> (see <a href="#">9.6 Data Structures for Advanced Driver Assistance (ADAS)</a> on page 134)</p>

### 5.2.1 Flexible Attribute Maps

The flexible attribute maps define the following attribute groups:

- Legal speed limits

For legal speed limits, the following attributes are grouped in an attribute map:

- Speed limit
- Vehicle type
- Legal speed limit range, for example, within town
- Weight or other car property restrictions (optional)
- Time (optional)
- Weather conditions (optional)
- Single or dual carriageways (optional)

- Warning sign icons

For warning signs, the warning sign and a reference to the respective icon in the BMD icon table are grouped in an attribute map, see *NDS – Compiler Interoperability Specification, 17.6 Warning Sign Attributes* on page 246.

- Car light required all time (DAYTIME\_RUNNING\_LIGHT)

## 5.3 Region-specific Descriptive Names of Selection Criteria

Depending on the region, the name of a selection criterion can differ. For example, a federal state in the USA is called “State”, whereas in Canada it is called “Province”. The descriptive names for region-specific selection criteria are stored in the `RegionSpecificDescriptiveTable`.

DataScript location: `nds.overall.metadata`

Each selection criterion name consists of a list of name strings for all languages defined for a database. If these names are identical for the complete update region, the names for selection criteria are defined in the metadata of the Name building block. If different names are used for different regions within the same update region, the names for selection criteria are defined in the `RegionSpecificDescriptiveTable`.

For more information, refer to 10.5 *Selection Criteria* on page 184.

*Table 5-3 Data in the RegionSpecificDescriptiveNameTable*

Data	Description
<code>regionId</code>	Identifies the region
<code>selectionCriterionCode</code>	Specifies the named object feature class
<code>languageCode</code>	Specifies the language for the specified region
<code>nameString</code>	Name strings belonging to the selection criterion

## 5.4 Road Number Class Prefix Table

The `RoadNumberClassPrefixTable` contains information on administrative road number abbreviations, for example, “E” for European roads (within Europe). An update region may contain different regions with different classification systems. In this case, the road number prefix table is defined for each region separately.

DataScript location: `nds.overall.metadata`

*Table 5-4 Data in the RoadNumberClassPrefixTable*

Data	Description
<code>regionId</code>	Identifies the region
<code>prefixId</code>	Identifies the road number class prefix
<code>roadClass</code>	Specifies the administrative classification of the road number; in Germany, for example, European roads have road number class 1, highways have road number class 2
<code>roadNumberDisplayClass</code>	Specifies the local priority for the road classes; in Germany, for example, European roads have a lower local priority than highways
<code>acousticOutput</code> <code>textOutput</code> <code>prefixWithIconOutput</code> <code>iconInMapOutput</code>	Flags indicating the output type For <code>iconInMapOutput</code> , an icon set is referenced, which may have a reference to an icon text with a color (usually needed for road signs), see Chapter 20 <i>Icons</i> on page 355
<code>prefix</code>	String representing the prefix, for example „E“ for European roads
<code>languageCode</code>	Specifies the language code for the prefix string; this is important for transliterated road numbers
<code>isTransliterationOf</code>	Indicates the relation to the original road number for transliterated road numbers
<code>iconSetId</code>	Reference to the icon set that is used for rendering the road number icons in the BMD building block. If set to <code>NULL</code> , the road number prefix has no icon.
	<p><b>Note</b> An array of icon set identifiers may be referenced. This makes it possible to provide icons with different widths for road number strings with varying length. Example: “E1” needs a shorter icon than “E999”. Applications can thus fit the string into the icon.</p>
<code>exitIconSetId</code>	Specifies the default exit icon if no exit icon set ID is defined in the flexible attribute

Every road number class can be displayed in another hierarchical display class and has a different number prefix, such as A for highway (German: Autobahn) in Germany. The prefix E, for example, stands for “European Road” in Germany. European roads are assigned to the road number hierarchy class. In Germany, however, European roads are less important and, therefore, only displayed on third position (see Table 5-5).

Table 5-5 Road number class prefix table, example

Road class	Road no. display class	Status flags	Icon set reference	Icon text color	Prefix	Result	Explanation
1	3	PI	../e_twochars.png ../e_threechars.png ...	White	E		Prefix in the icon, no acoustic or textual output
2	1	ATI	../a_twochars.png ../a_threechars.png ...	White	A		No prefix in the icon, but acoustic and textual output as “A1”
3	2	ATI	../b_twochars.png ../b_threechars.png ...	Black	B		No prefix in the icon, but acoustic and textual output as “B1”
4	4	T	None	None	K	No icon	Textual output only, for example “driving on K120”

## 5.5 Additional Icons

The `AdditionalIconRefTable` contains additional icon types (`addIconType`) that are required to indicate the bannered road type per region (`regionId`) and administrative road class (`adminRoadClass`). For more information, refer to Section 20.5 *Additional Icons for Bannered Routes* on page 357.

DataScript location: `nds.overall.metadata`

Table 5-6 Data for additional icons

Data	Description
<code>regionId</code>	Identifies the region
<code>adminRoadClass</code>	Specifies the administrative classification of the road number; in Germany, for example, European roads have road number class 1
<code>addIconType</code>	Additional icon type
<code>addIconSetId</code>	Reference to the icon set

## 5.6 Address Format

The **AddressFormatTable** contains the address formats used in the database defined by a name type and a separator.

Address formats can be used to retrieve the hierarchy of named objects referring to a link in order to display the corresponding address. The address format is defined by a format element (**nameFormat**), a **namedObjectClass**, and a separation string (**addressSeparator**) for concatenating the subsequent element.

DataScript location: `nds.overall.metadata`

Example (Germany):

- Format definition // Realworld equivalent
- Road/standard/BLANK // Steintorweg 8
- House/house number/NEWLINE//  
zip code/standard/BLANK // 20099 Hamburg  
city/standard/EMPTY

## 5.7 Region-specific Profiles

The **RegionProfileTable** contains distinct and generic profiles for a specific region.

Generic profile attributes represent properties of features changing over time. They are stored per region and may be associated to links and to transitions of intersections in the Routing building block.

DataScript location: `nds.overall.metadata`

For more information, refer to Section 9.7 *Generic Profile Attributes* on page 145.

## 5.8 Time Zones

Navigation Data Standard provides time zone information. This information can be used for the following purposes:

- To adapt the estimated time of arrival to the local time of destinations for routes through different time zones.
- To decide whether a time-dependent restriction must be regarded during route calculation.
- To retrieve appropriate values from speed profiles during routing (see Section 9.7 *Generic Profile Attributes* on page 145.)

Time zones are defined in the **TimeZoneTable**. The time zone definition consists of an offset to coordinated universal time (UTC) in increments of quarters of an hour. A flag indicates if daylight saving time has to be used. If daylight saving time is used, the date and time information for switching between normal time and daylight saving time is stored in the table. Also, the offset of the daylight saving time from UTC can be specified in increments of quarters of an hour.

DataScript location: `nds.overall.metadata`

Table 5-7 Time zone data

Data	Description
<code>timeZoneId</code>	Identifies the time zone
<code>utcOffset</code>	Instance of <code>QuarterHourTimeOffset</code> used for UTC time; defines the time offset in quarter hour units using values from -48 to +56 (see <a href="http://en.wikipedia.org/wiki/Time_zone">http://en.wikipedia.org/wiki/Time_zone</a> )
<code>defaultName</code>	Default name of the time zone; language-specific time zone names are stored in the <code>TimeZoneNameTable</code>
<code>hasDayLightSaving</code>	Indicates whether daylight saving time applies to the respective time zone
<code>dayLightOffset</code>	Instance of <code>QuarterHourTimeOffset</code> used for daylight saving time
<code>startTimeOfDay</code> <code>startDayOfWeek</code> <code>startWeekOfMonth</code> <code>startMonthOfYear</code>	Determine the day and time the daylight saving time starts
<code>endTimeOfDay</code> <code>endDayOfWeek</code> <code>endWeekOfMonth</code> <code>endMonthOfYear</code>	Determine the day and time the daylight saving time ends

The **RegionTimeZoneTable** contains a list of time zones (`timeZoneId`) that are available for a specific region (`regionId`). If all tiles of an update region use the same time zone, no further attributes need to be assigned to the routing features of the update region. If more than one time zone applies for an update region, the flexible attribute `TIME_ZONE_REFERENCE` may be used to assign a time zone to a tile or specific routing features. The attribute contains a reference into the `RegionTimeZoneTable`.

The time zone names are stored in the `TimeZoneNameTable`. The table contains the `timeZoneId` to identify the time zone, the language code (`languageCode`) and the respective name string (`timeZoneNameString`).

## 6 Attributes

NDS defines attributes either as fixed or flexible attributes. Fixed attributes are an integral part of features within a class: A corresponding attribute value must be provided when the feature is instantiated. A typical fixed attribute is the administrative road class, which informs about the official classification of a road as defined by the authorities of a country (see Section 9.2.3 *Attributes of Link Features* on page 117). Another example is the number of vertices for area features in the Basic Map Display building block (see Section 11.5 *Area Features* on page 211).

In contrast to fixed attributes, flexible attributes are only assigned to features of a feature class if they are required to model a real-world situation, such as a speed limit on a freeway.

Flexible attributes are a powerful means to map, for example, complex traffic regulations. Typical flexible attributes for features in the Routing building block are:

- Speed limit
- Right-of-way regulations
- Prohibited passage regulations

---

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

---

### Related Topics:

- *NDS – Compiler Interoperability Specification, 3.5 Flexible Attribute Mechanisms* on page 60
- Complete list of the flexible attributes available for NDS: *Appendix D List of Flexible Attributes* on page 381
- Use cases for working with flexible attributes and attribute groups: *Chapter 17 Using Flexible Attributes and Attribute Groups* on page 221

### 6.1 Attribute Groups

Flexible attributes can be combined to form a so-called *attribute group* which enables, for example, the modelling of complex traffic regulations. The resulting attribute group can be assigned to one or more features. This allows for optimizing the use of attributes, especially in the Routing building block where a large amount of traffic regulations has to be modeled.

Attribute groups are defined per attribute layer (see Section *Attribute Layers* on page 82). Attributes and attribute groups that are used repeatedly are stored only once per tile to reduce the database size.

Some flexible attributes describe characteristics of features and may thus be assigned individually to features. An example of such an attribute would be the regulation that a road is blocked for trucks. Other flexible attributes specify conditions under which flexible attributes describing characteristics are valid. An example of a condition would be that a regulation is valid on Sundays only, for example, that a road is blocked for trucks on Sundays only.

To restrict an attribute describing a characteristic, the attribute shall be grouped with one or more flexible attributes describing conditions. The restriction expressed by this attribute is valid if **all** restrictions described by the different attributes describing conditions and grouped with the restriction attribute are met.

Some attributes describing conditions contain a bitmask, allowing to select m of n Boolean values, such as a set of pre-defined vehicle types. In addition to this, condition attributes with bitmasks and some other attributes describing conditions have a **isInclusive** flag with the following meaning:

- **isInclusive** flag is set to TRUE: The condition is met, if one of the selected Boolean values is met
- **isInclusive** flag is set to FALSE: The condition is met in all cases, except if one of the Boolean values is met.

---

**Note**

If the **isInclusive** flag is set to TRUE for an attribute with a bitmask type, then this does not have the same meaning as setting the bitmask inversely and the **isInclusive** flag to FALSE. If a restriction is valid because a condition for the restriction in an attribute group is met, but another condition with an **isInclusive** flag set to FALSE is met too, the latter one precedes and makes the restriction invalid.

---

## 6.2 Attribute Layers

For many features, mainly in the Routing building block, a large number of attributes is needed to describe the features. To facilitate the handling of these attributes for the application, NDS supports the classification of flexible attributes by means of attribute layers.

The following attribute layers are currently defined:

- Layer for truck attributes: This layer contains attributes for describing traffic restrictions for truck vehicle types, for example, weight limitation, height/width/length limitation, and road width.
- Layer for route guidance attributes. This layer is only used for level 13 and contains attributes describing maneuvers, right-of-way, traffic lights, and also contains signpost attributes, lane attributes, and toll attributes required for route guidance applications.

- Layer for name attributes: This layer contains attributes for assigning names to routing features.
- Layer for ADAS attributes: This layer contains attributes describing physical properties, such as curvature, slope, road, and lane width.

Attribute layers are typically modeled as separate columns of the routing tables. Besides the attributes, these columns contain attribute point lists.

The information, which attribute layers are available in an NDS database is stored in the Routing metadata (`nds.overall.metadata > RoutingMetadata`). For more information, see *Routing Metadata* on page 111.

## 6.3 Attribute Points and Shape Points

The spatial extension of attributes along line features is defined by grouped flexible attributes of the type `VALIDITY_RANGE` using shape points and additional attribute points, which can be interleaved with shape points (see *Flexible Attribute for Defining Validity Ranges* on page 349). A range can start with the start point of a line feature and end with the end point of the same line feature, or with any shape point or attribute point in-between. Shape points are stored in shape point lists directly within the features. They are limited to vertices or changes of attributes relevant for map display, such as road class, link type, etc. Attribute points are organized in attribute point lists per feature and are assigned as an ungrouped attribute within the respective attribute layers (DataScript location: `nds.common.flexibleattributes > AttributePointList`). They contain a sequence of attribute points to describe the location of attribute changes at positions where no shape point exists.

---

<b>Note</b>	Attribute point lists are provided as flexible attributes for features in the Routing building block separately for each attribute layer. As they are not grouped with other attributes, they can be reused by different attributes (types and values) in the same layer.
-------------	---

---

Common condition attributes, such as validity ranges or time restrictions, can be used in more than one layer. Other attributes are specific for one layer and must not be used in other layers, such as the `CURVATURE` attribute for the ADAS attribute layer. This ensures consistency.

For the use of shape points and attribute points, the following rules apply:

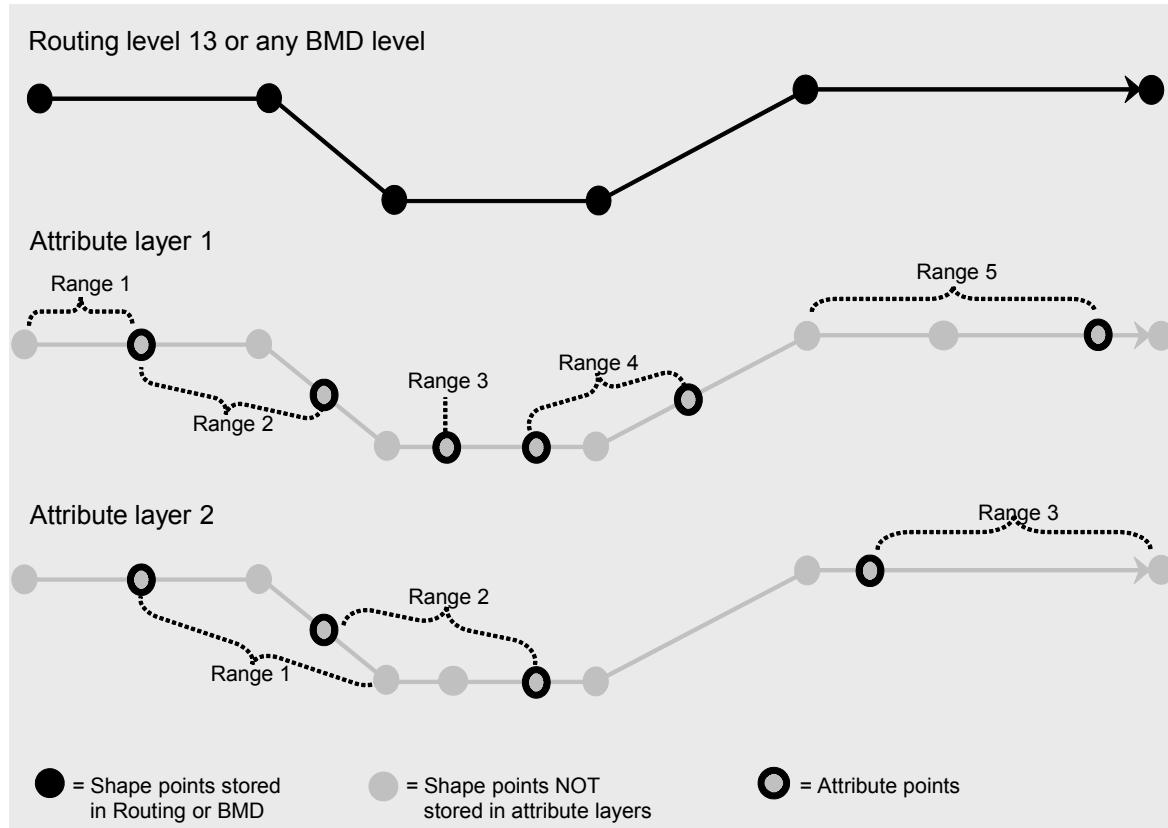
- Shape points are used for attributes belonging to the basic set of attributes stored in the routing data; attribute points are used for attributes assigned to attribute layers.
- Attribute points are stored for positions where attributes change along a feature and no shape point exists. Where shape points exist, they can be reused also for attributes in attribute layers.
- Attribute points are encoded with the same positional accuracy as shape points and by a coordinate difference to the preceding shape point or start node.

For the definition of validity ranges for attributes, shape points and attribute points are used. The following rules apply:

- Ranges may include start or end points of line features; these points are not stored in any of the separate attribute layers.
- If a point of an attribute change coincides with a shape point, no attribute point will be generated and the shape point is used instead. Coinciding attribute points of different attribute layers are stored separately in the respective layers.
- Attribute points shall not coincide with neighboring shape points but may coincide with other shape points of the same feature if the feature is self-intersecting.
- Before an application applies validity ranges to a line feature for a specific attribute layer, it shall interleave the shape points of the line feature with the attribute points for this feature in the given layer.
- If no attribute points for a feature are needed in an attribute layer, the validity ranges refer to shape points. Also, if there are no shape points available along a straight line feature, but attribute points, the validity ranges refer to attribute points only. In these cases, interleaving takes place at least between start and end point.
- A validity range may be a single point (for example, for a signpost attribute).

The following figure shows examples for shape points, attribute points, and validity ranges for routing level 13 and two attribute layers.

Figure 6-1 Line features and associated shape points



## 6.4 References between Attributes and Features

Flexible attributes are stored in lists per attribute type and linked to features by means of feature references. Such references can be unidirectional or bidirectional.

*Bidirectional references* are used for properties that are common to both directions of a line feature, for example, the road width attribute. *Unidirectional references* are used for properties which are common to only one direction of a line feature. Unidirectional references always refer to the right side of the line feature which is defined as follows:

- Right side for forward direction: Seen from start to end of the line feature
- Right side for reverse direction: Seen from end to start of the line feature

Some attributes can have unidirectional and bidirectional references. An example would be the road name, for which unidirectional references can occur as rare exceptions.

House numbers can refer to the right and the left side of a line feature. In this case, the direction is interpreted as side of the link. This is especially important for the application as it needs to make a house number to be accessible on the reverse direction from either travel direction.

## 6.5 Common Attributes

Common attributes are flexible attributes, which can be assigned to features of more than one building block. They can, for example, be used for grouping purposes. The following sections describe the common attributes defined for NDS and their use.

### Flexible Attribute for Defining Validity Ranges

The flexible attribute **VALIDITY\_RANGE** uniquely identifies a specific validity range within a list of points. A range can start with the start point of a base link or road geometry line and end with the end point of the same base link or road geometry line, or with any shape point or attribute point in-between. If start and end point coincide, a validity range can be defined by only one point.

The domain of the start and end line reference for the validity range is [0..n], where 0 is the first point of the geometry. For routing geometry, 0 is the start intersection and n is the end intersection. Other points between 0 and n are the shape points of the geometry.

### Flexible Attributes for Specifying Data Quality

The completeness of the digitized road network can be described by means of flexible attributes for digitization status and attribution status (see Table 6-1).

*Table 6-1 Attributes for data quality*

Flexible attribute	Values	Description
DIGITIZATION_STATUS	FULLY_DIGITIZED	Every road of the road network is digitized.
	INTERCONNECTING_ROADS	Only interconnecting roads of the road network are digitized.
	MAJOR_ROADS_ONLY	Only major roads are digitized.
ATTRIBUTION_STATUS	NOT_FIELD_SURVEYED	Attribute information is derived from map material or aerial photos.
	PARTLY_ATTRIBUTED	Partly attributed
	FULLY_ATTRIBUTED	Applied if all attributes relevant for routing are available

The default data quality values for the features of a road network are derived from the input data. For values deviating from the default value, a different digitization and/or attribution status can be set per feature. See also *NDS – Compiler Interoperability Specification*, 5.6 *Assigning Data Quality Attributes* on page 84.

### Flexible Attributes for Specifying Time Periods

The flexible attributes in Table 6-2 specify time periods. They can be combined to attribute groups to express complex time conditions, for example, “each Friday in the 2nd week of March from 10 to 12 am”.

DataScript location: `nds.common.flexibleattributes`

*Table 6-2 Flexible attributes specifying time periods*

<b>Flexible attribute</b>	<b>Description</b>
TIME_RANGE_OF_DAY	Specifies start and end time of a period To specify a single point in time, set identical values for start and end time.
TIME_RANGE_OF_WEEK	Specifies start and end time of a period in hours and minutes The minimum range that can be defined is 1 minute. The range contains all times between start and end time including the start and end times.
DATE_RANGE_OF_YEAR	Specifies start and end day of a time period
DAYS_OF_WEEK	Specifies for each day of a week by one bit each (m-of-n selection) if the attribute is valid
DAYS_OF_MONTH	Specifies for each day of a month by one bit each (m-of-n selection) if the attribute is valid
DAY_OF_YEAR	Defines a specific day of a year by means of month and day-in-month
MONTHS_OF_YEAR	Specifies for each month of the year one bit (m-of-n selection) if the attribute is valid
Year	Specifies the number of years
WeekdayInMonth	Defines a weekday in a month, for example, 3rd Monday in June
TimeOfDay	Time measured in minutes beginning from Monday, 0:00 The maximum is 7 x 24 x 60 minutes.
onOddDaysOnly	Defines if a certain restriction is only available on odd days (TRUE) or even days (FALSE).
FERRY_TRAVEL_TIME	Travel time in minutes for a one way trip.

### Flexible Attributes for Routing and Basic Map Display Building Block

The flexible attributes in Table 6-3 are used by both, routing and basic map display features.

DataScript location: `nds.common.flexibleattributes`

*Table 6-3 Flexible attributes shared between Routing and Basic Map Display building blocks*

<b>Flexible Attribute</b>	<b>Description</b>
ROAD_Z_LEVEL	Additional height attribute points for map lines, base links, or road geometry lines, representing the height level of roads
SLOPE	Defines the slope of a road
PHYSICAL_WIDTH_IMPERIAL and PHYSICAL_WIDTH_METRIC	Defines the width of a road
FERRY_TYPE	Operating type of the ferry, for example, ship or hovercraft
PARKING	Defines the type of a parking facility, for example, open parking space, underground parking, or multi-storey car parks
TYPE_OF_PAVEMENT	Defines the pavement type of a road The following attribute values are possible: Unknown, sandy, gravel, cobblestone, concrete, asphalt, other

Flexible Attribute	Description
DIGITIZATION_STATUS and ATTRIBUTION_STATUS	Describes the completeness of the digitized road network (see <i>Flexible Attributes for Specifying Data Quality</i> on page 86)
OVERPASS	Indicates that a line or link passes over another line or link, for example, used for bridges
UNDERPASS	Indicates that a line or link passes under another line or link Underpasses are usually used for tunnels. However, there can be exceptions, for example, a bridge passing under another bridge.
TOURIST_ROUTE_TYPE	Flags a link or line which is of special interest for tourists
STUBBLE	Indicates that a line or link is not verified by field survey, but just digitized when passing by Stubbles are needed to give correct guidance advices and ignore such lines for map display.
NON_DEFAULT_TRAFFIC_SENSE	Defines an exception to the default traffic sense as defined in the respective administrative region of the tile
SPEED_LIMIT	Defines a speed limit
MINIMUM_SPEED	Defines the allowed minium speed
UNDER_CONSTRUCTION	Indicates that a routing feature is still under construction and not opened for traffic. This attribute does not indicate a construction area on a road.
<b>Note</b> This attribute should be stored within the main routing data.	

## Flexible Attribute for Assigning Named Objects to Routing and Basic Map Display Features

The following flexible attributes are used for assigning named objects to routing and basic map display features (DataScript location: `nds > common > flexibleattributes`):

- `NAMED_OBJECT_REFERENCE` or the combination of `NAMED_OBJECT_BASE_REF` and `NAMED_OBJECT_RELATIVE_REF`
- `ADMIN_HIERARCHY_NAMED_OBJECT_REFERENCE`
- `POSTAL_CODE_NAMED_OBJECT_REFERENCE`

For more information, refer to *NDS – Compiler Interoperability Specification, 7.6.4 Assigning Named Objects to Routing and Basic Map Display Tiles and Features* on page 154.

## Flexible Attribute for Defining Currency Values

The flexible attribute `CURRENCY` can be used for defining currency values if no concrete toll charge values are given („You can pay in Euro and in Yen“) and for POIs to define price ranges for hotels.

## 7 Partitioning of Geographic Data

This chapter explains the concepts used in Navigation Data Standard to map geographic data to a database. Based on the underlying coordinate system, the tiling scheme used for partitioning geographic data is introduced.

### 7.1 WGS 84 and EGM96

Navigation Data Standard uses *WGS 84*, the World Geodetic System dating from 1984, as the standard coordinate reference system for representing latitude and longitude of objects. WGS 84 defines a fixed global reference frame for the whole earth for use in geodesy and navigation. By using this standard coordinate reference system, it is possible to uniquely identify any point on the Earth's surface – except the North Pole – by two numbers, the x-coordinate and the y-coordinate of the point, where x corresponds to the longitude and y to the latitude.

Navigation Data Standard uses *EGM96*, the Earth Gravitational Model from 1996, as the standard reference system for representing absolute heights. In contrast to the WGS 84 ellipsoid, EGM96 specifies the height above a so-called geoid which corresponds to the height above sea level. Thus, the heights represented by EGM96 are closer to reality.

---

**Note** Navigation Data Standard uses the following terms:

- *Elevation* denotes heights on the Earth's surface
  - *Altitude* denotes heights above the Earth's surface
  - *Height* denotes relative heights, such as the height of a building
- 

#### Related Topics:

- World Geodetic System (WGS 84): [http://en.wikipedia.org/wiki/WGS\\_84](http://en.wikipedia.org/wiki/WGS_84)
- EGM 96: <http://en.wikipedia.org/wiki/EGM96>
- EGM 96: <http://en.wikipedia.org/wiki/Geoid>
- Elevation: <http://en.wikipedia.org/wiki/Elevation>

### 7.2 Coding of Coordinates

For coding coordinates, a scaling factor is applied so that  $360^\circ$  correspond to  $2^{32}$ , to exhaust the full range of 32 bit integers. Hence,  $-2^{31} \leq x < 2^{31}$  in longitude and  $-2^{30} \leq y < 2^{30}$  in latitude. In Navigation Data Standard, a coordinate unit corresponds to  $90/2^{30}$  degrees of longitude or latitude.

The origin of the coordinate system is the intersection of the WGS 84-0-meridian and the equator.

The intention of this encoding is to store the x-coordinate as signed 32-bit integers

$x = x_{31} \dots x_1 \ x_0$  and the y-coordinate as signed 32-bit integers:

$y = y_{30} \dots y_1 \ y_0$  with  $x_{31}$  and  $y_{30}$  as the most significant bits and  $x_0$  and  $y_0$  as the least significant bits.

## Related Topics

*NDS – Compiler Interoperability Specification, 3.3 Reducing Coordinate Precision on page 57.*

### Example – Part 1

The examples in Table 7-1 illustrate the integer coding of WGS 84 coordinates, which are given as floating point values.

Table 7-1 Integer coding of WGS 84 coordinates given in floating point values, examples

Location	WGS 84		NDS (decimal)		NDS (hexadecimal)	
	x-coordinate (longitude)	y-coordinate (latitude)	x-coordinate (longitude)	y-coordinate (latitude)	x-coordinate (longitude)	y-coordinate (latitude)
Eiffel Tower	2.2945°	48.858222°	27374451	582901293	0x1a1b373	0x22be5e2d
Statue of Liberty	-74.044444°	40.689167°	-883384626	485440671	0xcb589ece	0x1cef3c9f
Sugarloaf Mountain	-43.157444°	-22.948658°	-514888362	-273788154	0xe14f6d56	0x6fae5306
Sydney Opera House	151.245278°	-33.856111°	1804068537	-403919137	0x6b87e6b9	0x676ecaedf
Near the Millennium Dome (O2)	0.0°	51.503°	0	614454724	0	0x249fd5c4
Near Quito	-78.45°	0.0°	-935944956	0	0xc8369d04	0

- 
- Note**
- Due to the floating point arithmetic, the result values can differ in the least significant bits between different implementations. As the data precision in NDS is usually higher than the data precision provided by map suppliers, this has no negative impact on the overall data quality. Nevertheless it is important to ensure a consistent conversion.
  - For the final conversion from floating point values to integer values it is recommended to use the operation `floor`, not the operation `truncate` nor the operation `round`. The reason is that, for negative numbers, `truncate` does not round in the same direction as the `floor` function: `truncate` always rounds towards zero, the `floor` function rounds towards negative infinity. Operation `round` rounds towards the closest integer. Using `floor` results in a better consistency with the tiling scheme.
  - As the hexadecimal presentation indicates, NDS uses the usual two's complement format. This allows to derive the values for the 31-bit y-coordinates by using ordinary 32-bit integers and masking-off the highest bit. For example, the value  $-273788154$  equals `6fae5306` as 31-bit integer and `efae5306` as 32-bit integer (due to sign-bit extension).
- 

## Morton Codes

From a coordinate, which is defined by two integer values for longitude ( $x$ ) and latitude ( $y$ ), the *Morton code* can be derived, which is a single number. Thus, two dimensions are mapped into one dimension. To be more precise, for a given coordinate with

$$x = x_{31} \ x_{30}...x_1 \ x_0 \text{ and } y = y_{30}...y_1 \ y_0$$

the Morton code  $c$  is given by the 63-bit integer

$$c = x_{31} \ y_{30} \ x_{30}...y_1 \ x_1 \ y_0 \ x_0$$

that results from a bit interleaving from x- and y-coordinate, hence,  $0 \leq c < 2^{63}$ . If stored in a 64-bit integer, the Morton code  $c$  is prefixed with a 0 bit, thus always positive.

If items are ordered according to their Morton code, we speak of a *Morton order*.

- 
- Note**
- For general information regarding Morton order, refer to [http://en.wikipedia.org/wiki/Z-order\\_\(curve\)](http://en.wikipedia.org/wiki/Z-order_(curve)). The treatment within NDS differs as it takes into account (a) signed values for x and y and (b) different value ranges for longitude and latitude.
- 

## Example – Part 2

Continuing the example above, we get the following Morton codes:

Table 7-2 Morton codes derived from the values given in part 1 of the example

Location	Morton code (decimal)	Morton code (hexadecimal)
Eiffel Tower	579221254078012839	0x809cea967ad1da7
Statue of Liberty	5973384896724652798	0x52e5b9ea4bf4d2fe
Sugarloaf Mountain	8983442095026671932	0x7cab98fd365b113c
Sydney Opera House	4354955232480651243	0x3c6fe8b5dcbe7eb
Near the Millenium Dome (O <sub>2</sub> )	585611620934393888	0x82082aaa222a020
Near Quito	5782627506097029136	0x5040051441510010

Within NDS, the Morton code of a given coordinate and the associated Morton order have several applications, for example, in the POI building block (see Chapter 13 *POI Building Block* on page 253) and in the tiling scheme, which is explained in the following section.

## 7.3 Tiling Scheme

WGS 84 ensures that the position of a feature can be uniquely identified by its WGS 84 coordinates. An additional mechanism must be provided for identifying, grouping and locating the features with geographical references. This mechanism has to support high-performance data management, for example, for different levels of data granularity, for quick data access, or for incremental database updates.

Navigation Data Standard uses a global *tiling scheme* for this purpose. The scheme forms a uniform grid that divides the whole surface of the earth into tiles, thus partitioning the data that is available. The tiles form approximately rectangular territorial sections.

In NDS, tiles represent a unit for storing data in the database. That means a tile contains the data of all the features located within the geographic borders defined by the tile. The features are encoded in DataScript in a compact way and stored in the database in BLOBs (binary large object) and/or relationally, depending on the respective building block.

Tiles provide multiple advantages:

- They enable applications to load all data belonging to the same geographic area with one single access to the database. This accelerates, for example, map display and route calculation.
- Tiles may serve as a unit for updates, meaning they represent a versioned item in NDS. Thus, versioning information is stored per tile, not per individual feature. This reduces the space overhead considerably.
- Tiles enable a hierarchical reference schema for features (see also Table 3-1 on page 51). To refer to several features belonging to the same tile, the `tileId` has to be specified only once.

### Tiles and Building Blocks

Depending on the building block, database content is stored in BLOBs and/or relationally. The NDS tiling scheme is applied for building blocks that store their data BLOB-oriented:

- Basic Map Display (`nds.bmd.main > BmdTileTable`)
- Routing (`nds.routing.main > RoutingTileTable and RoutingAuxTileTable`)
- Orthoimages (`nds.orthoimages.main > OrthoImageTileTable`)
- Digital Terrain Model (`nds.dtm.bdam > DtmBdamTileTable and nds.dtm.heightmap > DtmHeightMapTileTable`)
- Traffic Information (`nds.ti.tmc.location > TmcTileTable`)

For the POI building block, which uses relational data storage, NDS additionally features *virtual tiles* to support specific search scenarios, see 13.6.3 *Virtual Tiles for POIs* on page 277.

#### Related Topics:

- Glossary definition of tiles: *Tile* on page 367
- Glossary definition of tiling scheme: *Tiling scheme* on page 367
- NDS versioned items: *NDS – Update Specification, 4.1 Versioned Items* on page 33

### 7.3.1 Tiles and Levels

The tiling scheme makes use of the levels defined in a database. The union of the maximum number of tiles available on each level covers the whole surface of the Earth. All tiles on one level are bounded by tile borders of constant latitude and constant longitude, so the size of the tiles on one level is always equal in terms of WGS 84 coordinate units (fixed tiling scheme).

---

**Note** As the length of tile borders is equal in terms of coordinates, and not in meters, a tile corresponds to a trapezoid, rather than a rectangle on the earth's surface. This effect increases from equator towards the poles.

---

---

**Note** The fixed tiling scheme may cause densely populated tiles, especially on level 13. The following NDS wiki article describes how to avoid this: [https://psf.poffice.eu/mediawiki/index.php/Big\\_tiles](https://psf.poffice.eu/mediawiki/index.php/Big_tiles).

---

The highest level in a database has two tiles. On the next lower level, each of these tiles is split into four. This is continued through all levels down to the lowest level, thus approximating a hierarchical structure from global into detail. Each level has its own underlying tiling structure and higher-level tiles are represented as the union of lower-level tiles.

The highest level in a database is labeled level 0. The lowest level is labeled level 15. Not all levels need to be available in a database.

Navigation Data Standard uses level 13 for the most detailed routing data. For map display, however, levels below level 13 can be added. For example, it might be useful to store a detailed city map on level 14.

Figure 7-1 Tiling scheme for partitioning the Earth's surface: Level 0

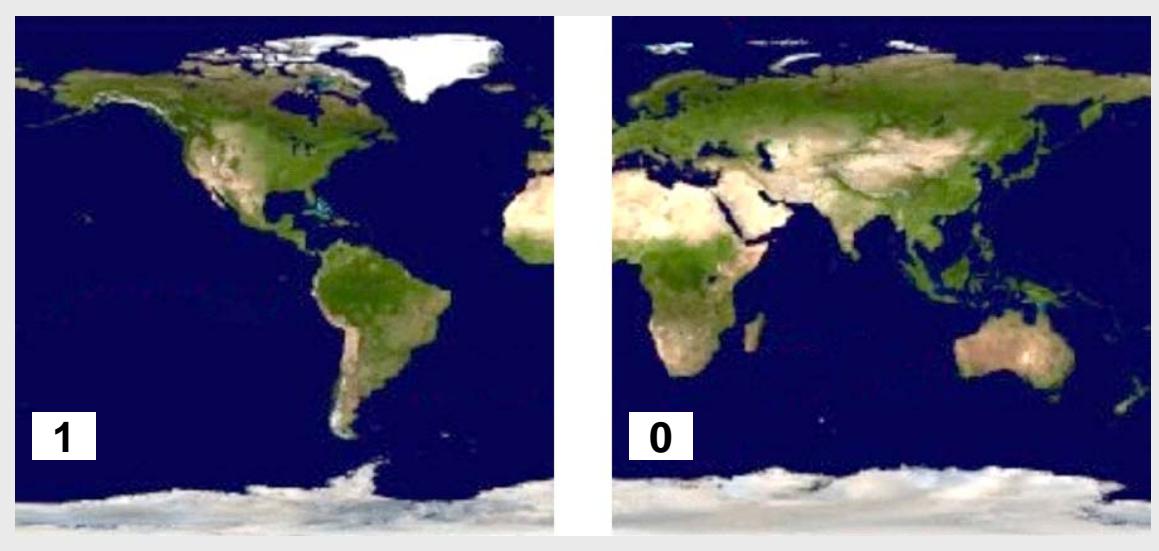


Figure 7-1 shows the two tiles on level 0. Whether a location on the Earth belongs to tile 0 or tile 1 on level 0, is determined by the most significant bit of its longitude in coordinate units ( $x_{31}$ ). This means for tile 0 longitudes in the range of  $0^\circ \leq x < 180^\circ$  and for tile 1 longitudes in the range of  $180^\circ \leq x < 360^\circ$  (signed:  $-180^\circ \leq x < 0^\circ$ ). Thus, for all coordinates contained in a level 0 tile, the tile number is identical to the value of the most significant bit of their Morton code.

Figure 7-2 Tiling scheme for partitioning the Earth's surface: Level 1

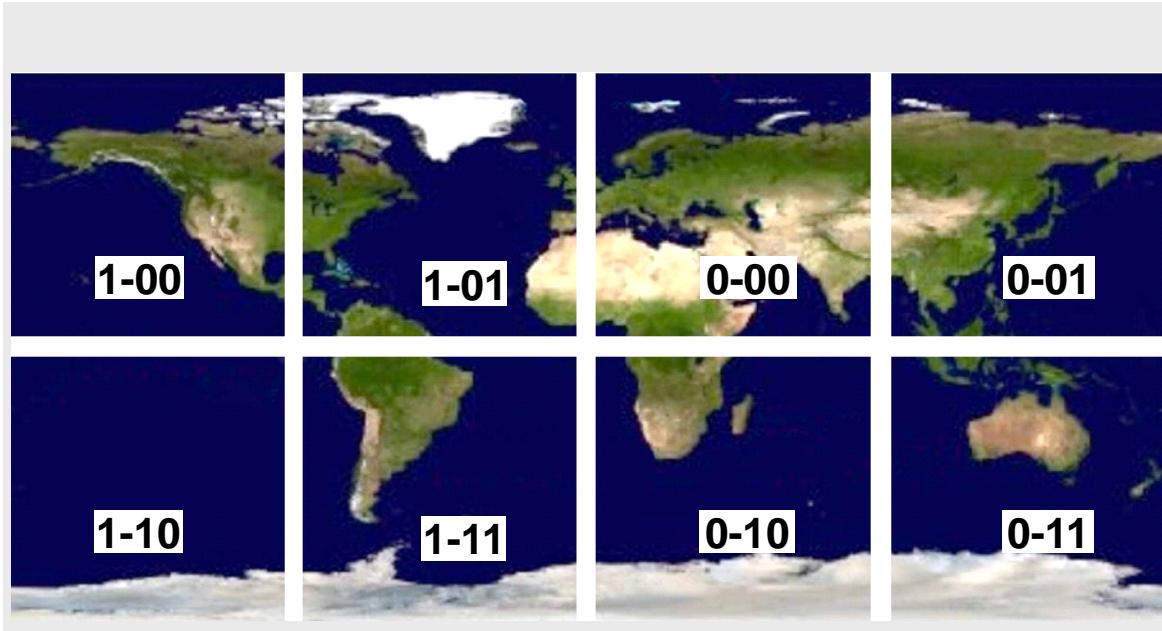


Figure 7-2 shows the tile numbering on level 1 according to a Morton scheme. Here, the tile numbers are identical to the three most significant bits of the coordinates' Morton codes. As indicated by the dash, each tile number also contains the tile number of its parent tile as a prefix.

*Figure 7-3 Tiling scheme for partitioning the Earth's surface: Level 2*

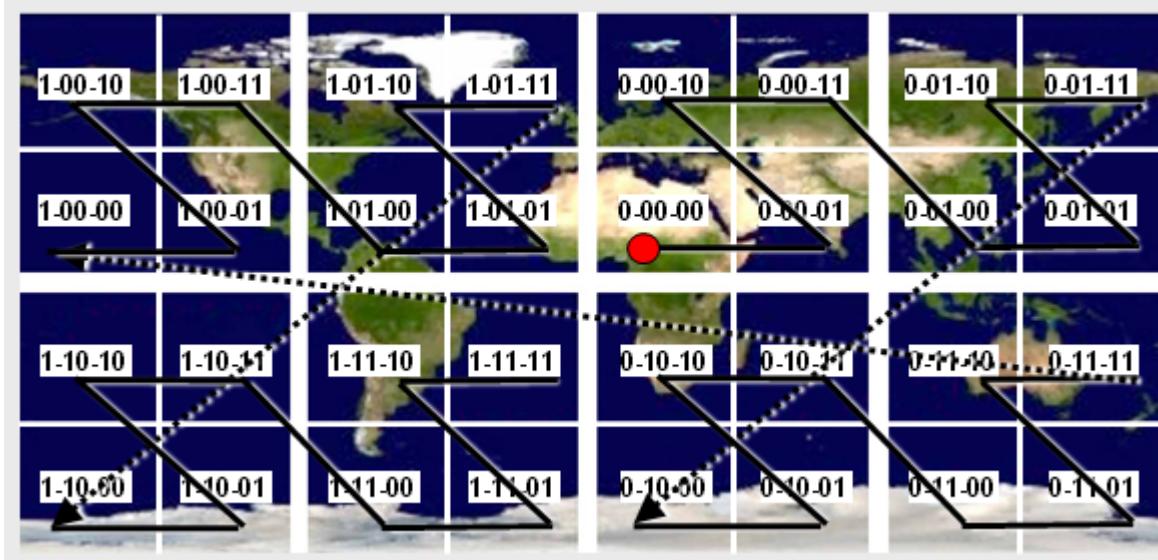


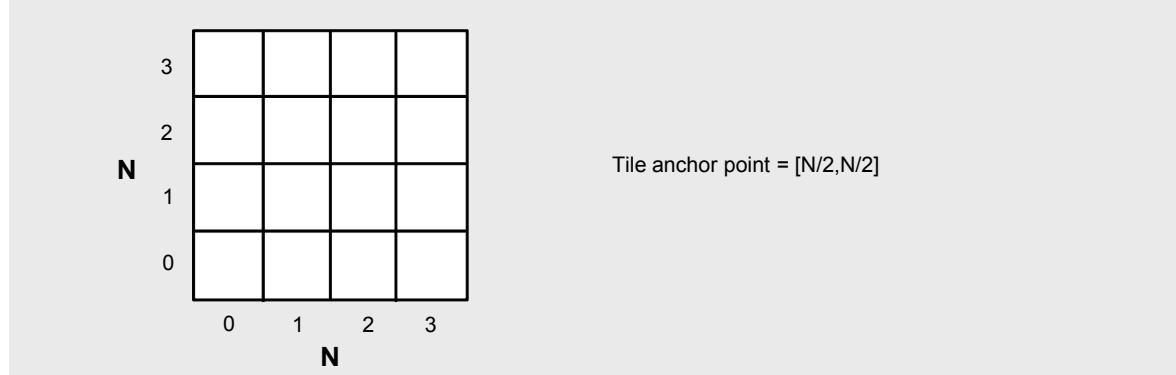
Figure 7-3 shows the tile numbering for level 2 according to the Morton scheme described above. The arrows indicate the Morton order of the tile numbers.

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. The number of cells per row or per column is  $2^{31-k}$  ( $k = \text{level number}$ ).

The addressable number of cells per tile depends on coordinate width. In terms of coordinate units, the side length of a level  $k$  tile is  $2^{31-k}$ . The level  $k$  tiling has  $2^{2k+1}$  tiles. The tile number of a level  $k$  tile is given by the most significant  $2^{k+1}$  bits of the Morton codes of the coordinates covered by that tile.

The tile anchor point is defined as number of rows divided by two and the number of columns divided by two.

Figure 7-4 Tile anchor point definition



The tiles in a database are identified via their tile ID. Each tile ID consists of the level number followed by the tile number. The physical encoding of a tile ID packs both components together into one 32 bit value (DataScript location: `nds.common > PackedTileId`; see Section 3.1 *Generating Tile IDs* on page 53).

### Example – Part 3

Table 7-3 shows the tile numbers of the covering tiles on level 10 and level 13 for each location:

Table 7-3 Tile numbers in level 10 and level 13

Location	Level 10 tile number		Level 13 tile number	
	(decimal)	(hexadecimal)	(decimal)	(hexadecimal)
Eiffel Tower	131699	0x20273	8428778	0x809cea
Statue of Liberty	1358190	0x14b96e	86924190	0x52e5b9e
Sugarloaf Mountain	2042598	0x1f2ae6	130726287	0x7cab98f
Sydney Opera House	990202	0xf1bfa	63372939	0x3c6fe8b
Near the Millenium Dome (O <sub>2</sub> )	133152	0x20820	8521770	0x82082a
Near Quito	1314817	0x141001	84148305	0x5040051

For level 13 tile numbers, the hexadecimal presentation makes the close relation to the Morton codes obvious (see Table 7-2).

### 7.3.2 Tiles and Clipping

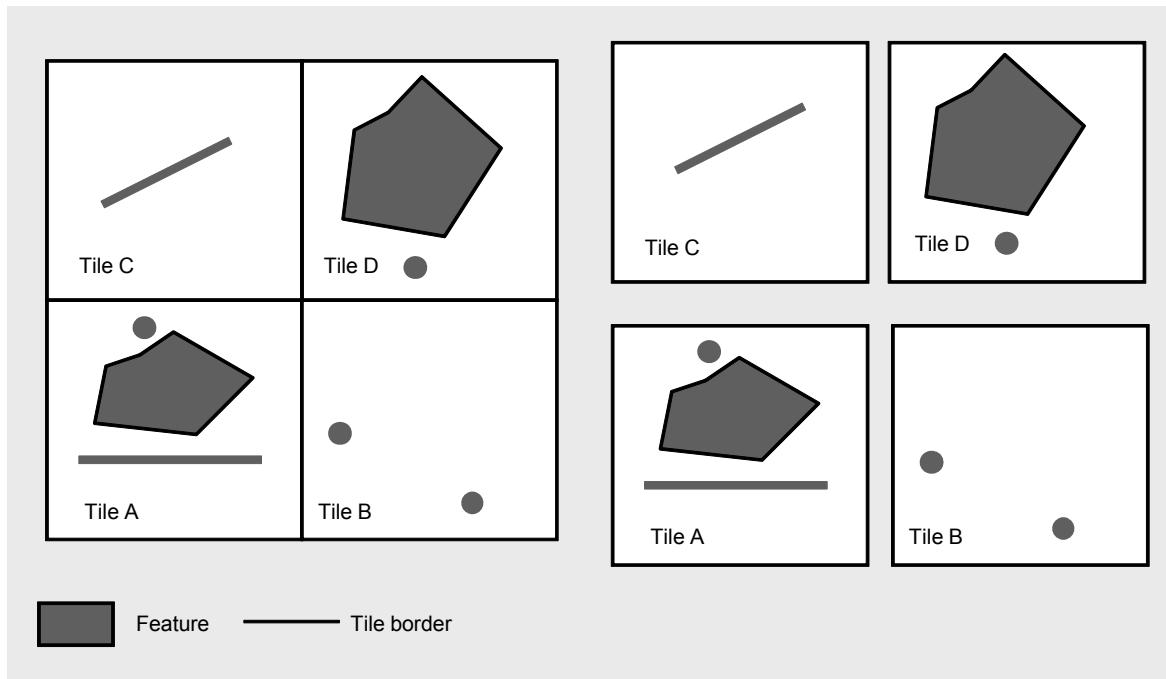
Navigation Data Standard has to ensure that the position of a feature can be uniquely identified. Features are positioned either on one or on several tiles. Certain provisions are necessary for exactly describing the position of a feature.

The following feature positions have to be distinguished:

- A feature is positioned on one tile (see Figure 7-5).
- A feature extends over more than one tile (see Figure 7-6).

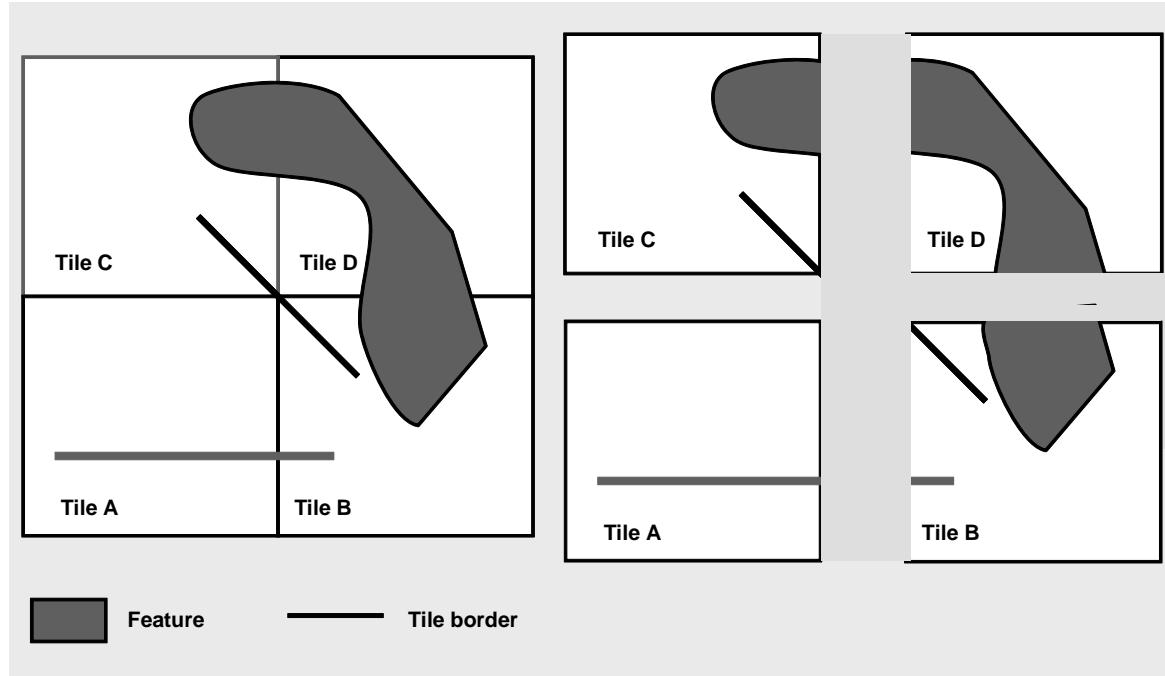
The features depicted in Figure 7-5 are assigned to exactly one tile. They do not cross tile borders.

*Figure 7-5 Features on one tile*



All features depicted in Figure 7-6 have geometries that extend over more than one tile. Navigation Data Standard uses a clipping mechanism to describe the geometry of these features.

Figure 7-6 Features extending over more than one tile



The geometry of any feature that extends over more than one tile is clipped at the tile borders and the individual parts of the feature are assigned to the corresponding tiles.

A feature that is split results in individual features located on the neighboring tiles. The points or lines on the tile border are duplicated on coordinate level. Any new feature resulting from a clipping operation can be uniquely assigned to one of the neighboring tiles.

The resulting features remain correlated by references to the same object, such as named object or routing link. In Figure 7-6, for example, the line feature that extends from tile A to tile B is split into two line features located on neighboring tiles. The part of the line feature located in tile A references named object 55, and the part of the line feature in tile B references the same named object. This results in an indirect reference between the two parts of the line feature. Split road geometry lines reference the same routing link.

### Related Topics:

- Clipping rules
  - *NDS – Compiler Interoperability Specification, 5.3 Splitting Road Features* on page 81
  - *NDS – Compiler Interoperability Specification, 8.4 Relation between Source Features and Compiled Features* on page 163
- Assigning individual feature types to tiles:
  - *NDS – Compiler Interoperability Specification, 5.5 Assigning Intersections to Tiles* on page 83
  - *NDS – Compiler Interoperability Specification, 8.1.1 Point Features* on page 157

### 7.3.3 Content Indices for Update Regions

For map display purposes, applications have to quickly determine which update regions contain data for a given tile. Because update regions may be defined on country borders (meaning with irregular outlines), this decision may be more time-consuming than acceptable. Comparisons with bounding boxes are not sufficiently precise to prevent useless accesses to the database.

For this reason, NDS provides optional content indices that contain the information whether a tile is present in a given update region and building block (DataScript location: `nds.all > ProductDatabase > TileContentIndexTable`). Content indices are represented as highly compact bit arrays. Thus it is feasible for an application to keep them in main memory. Content indices are only available for building blocks using tiles or virtual tiles.



## 8 Encryption

This chapter introduces the concept of encryption for the content of NDS databases. It covers the following topics:

- Scope of encryption
- Cipher algorithm
- Cipher key metadata

---

**Note** NDS provides data structures for applying encryption to database files and BLOBS. The handling of encryption keys, for example, distribution and storage of keys, is outside the scope of NDS.

---

### Related Topics:

- *NDS – Compiler Interoperability Specification, 4 Encryption* on page 71

### 8.1 Scope of Encryption

NDS supports the encryption of data stored in the navigation database to prevent unauthorized use of map data. The encryption concept for NDS, however, does not prevent copying an NDS database.

It is not mandatory to encrypt NDS databases. The prevention of illegal use of NDS, however, must be ensured. 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 URL field in the `UrBuildingBlockVersionTable`.
- Encryption of BLOBS containing tile data for a particular building block. BLOBS are defined by the BLOB field in the building block tile table, for example, `bmdTileTable`.

The fields `useBlobEncryption` and `useFileEncryption` in the `UrBuildingBlockVersionTable` (DataScript location: `nds.overall.productdbversion`) indicate whether encryption is applied to the building block instance of a given update region (see Table 8-1).

Table 8-1 Encryption indicator in the building block version table

Indicator	Description
useFileEncryption	<p>Indicates whether encryption is applied to the database file:</p> <ul style="list-style-type: none"> <li>- Value NULL indicates that encryption has not been applied.</li> <li>- Value different from NULL indicates that encryption has been applied. The value represents the cipher key info ID.</li> </ul> <p>For more information, refer to Section 8.1.1 <i>Database File Encryption</i> on page 102.</p>
useBlobEncryption	<p>Indicates whether encryption is applied to BLOBS:</p> <ul style="list-style-type: none"> <li>- Value 0 indicates that encryption has not been applied.</li> <li>- Value 1 indicates that encryption has been applied.</li> </ul> <p>For more information, refer to Section 8.1.2 <i>Encryption of Data BLOBs</i> on page 103.</p>

Encryption does not affect the tile version or other NDS data that is relevant for update processes. Also, encryption does not affect interoperability because the application decrypts data before using it for navigation.

---

<b>Note</b>	The NDS application shall decrypt data only in memory. It shall not write decrypted data to secondary storage.
-------------	--

---

### 8.1.1 Database File Encryption

To apply encryption to a database file, a cipher key needs to be assigned to the corresponding file. The cipher key is specified by the cipher key info ID which is stored in the field `useFileEncryption`.

NDS allows encryption of N bytes of a database file starting from offset O in order to reduce the impact of encryption on the performance of the application. N and O are specified per cipher key in the `CipherKeyInfoTable`. N+O must be smaller than the length of the database file. The bigger N is, the more security is provided. The smaller N is, the better performance can be achieved.

---

<b>Example</b>	O = 8192, N = 65536. That means that the first 8192 bytes of the file are not encrypted. The following 65536 bytes are encrypted. The remaining bytes are not encrypted.
----------------	--

---

#### Related Topics:

- Cipher key: Section 8.3 *Cipher Key Info Structure* on page 104

### 8.1.2 Encryption of Data BLOBs

Encryption of data BLOBs means that the BLOB containing the tile data has been encrypted. In order to be able to decrypt the tile, a cipher key needs to be assigned to each encrypted tile. The cipher key for a given tile is specified by the cipher key info ID, which is stored in the field `cipherKeyInfoId` of a tile table.

NDS allows encryption of N bytes of a data BLOB starting from offset 0 in order to reduce the impact of encryption on the performance of the application. There is no need to specify N and O for each tile. It is sufficient to specify N and O per cipher key (in the `CipherKeyInfoTable`).

The following applies:

- If the length of a tile in bytes is  $\geq 0+N$ , N bytes of a tile starting at 0 are encrypted.
- If the length of tiles in bytes is  $< N+0$ , all bytes starting from 0 are encrypted.

The bigger N is, the more security is provided. The smaller N is, the better performance can be achieved.

## 8.2 Cipher Algorithm

Usually, NDS databases are compiled, encrypted and distributed to many users. In this case, symmetric key ciphers are more suitable than asymmetric key ciphers.

For this reason, NDS uses the cipher algorithm AES-128 with CTR mode of operation. This algorithm/mode of operation has the following general properties:

- It uses symmetric keys, that means a private key cryptography.
- It produces a bit size of encoded data that is equal to the bit size of the original data (unvarying transformation of original data). Therefore, a part of a tile of arbitrary length can be encrypted.

### AES-128 with CTR Mode of Operation

The Advanced Encryption Standard (AES) is the result of an elaborated selection process to define a standard encryption algorithm. It is therefore well-investigated and can be considered to be secure. AES is also recommended by the National Institute of Standards and Technology (NIST). AES is royalty-free. For more information, refer to <http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

AES-128 uses 128-bit keys. This allows for a safety margin in case attacks come up that allow reducing the search space. NDS uses AES-128 because it is slightly more performant than AES-192 or AES-256. Also, AES-256 is more vulnerable to certain specialized forms of attack.

AES is a block cipher, encrypting blocks of 128 bits into blocks of 128 bits. To extend it for longer sequences as well as to provide unvarying transformation of original data, NDS uses the Counter (CTR) mode. For more information, refer to:

- [http://en.wikipedia.org/wiki/Block\\_cipher\\_modes\\_of\\_operation#Counter\\_.28CTR.29](http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation#Counter_.28CTR.29)
- <http://www.rfc-archive.org/getrfc.php?rfc=3686>

AES-128/CTR requires an initialization vector of 128 bits. It is not necessary to keep the initialization vector secret. It must, however, be random and is applied to only one encryptable item for the given key. In order to ensure that these requirements are met, NDS defines six values for the initialization vector for an encryptable item:

- Random number RND, stored per key in the `CipherKeyInfoTable` (see Section 8.3 *Cipher Key Info Structure* on page 104)
- Supplier identifier (SID), stored in the `NdsDatabaseSupplierTable` table
- Encryptable item code (EIC), stored in `encryptableItemCodeTable` or in `UrBuildingBlockVersionTable` (see Section 8.4 *Encryptable Item Code* on page 105)
- BLOB ID (BID) identifying the row containing the BLOB. Value 0 is used if the encryptable item is a file, not a BLOB.
- Version identifier of the BLOB (BV)
- Counter (CTR); represents an index of the AES data block which is part of the given encryptable item

The initialization vector is created by concatenating the values for RND, SID, EIC, BID, BV, and CTR.

### Protection of the Symmetric Key

The symmetric key has to be protected from being compromised during the transport to a user, for example, by encrypting a symmetric key with an asymmetric cipher algorithm.

---

<b>Note</b>	The transport and protection of the symmetric keys is outside the scope of NDS.
-------------	---

---

## 8.3 Cipher Key Info Structure

NDS databases store information about the applied cipher keys. This information can be used by decryption software for data decryption.

The cipher key information is stored per product in the `cipherKeyInfoTable` table (see Table 8-2).

Table 8-2 Fields of the cipher key info structure

Field	Description
<code>keyInfoId</code>	Uniquely identifies the cipher key info The ID is used to assign the cipher key info to a BLOB or a database file. If the value for the encryption key is set to NULL, no encryption has been applied to the respective tile.
<code>offset</code>	Number of first non-encrypted bytes in a database file or a BLOB
<code>length</code>	Number of encrypted bytes in a database file or a BLOB starting at <code>offset</code>

Field	Description
rnd	Random number which is used for constructing the initialization vector used with this key
cipherKeyData	BLOB containing the cipher key info data

## 8.4 Encryptable Item Code

The encryptable item code is used for constructing the initialization vector applied to that encryptable item. All item codes identifying BLOBS are stored in the `encryptableItemCodeTable`, which is stored per product (see Table 8-3).

Table 8-3 Fields of the encryptable item code table

Field	Description
eic	Number representing an encryptable item The number is unique per data supplier and product and is used for constructing an initialization vector applied to the encryptable item by the AES-128/CTR cipher algorithm.
columnName	SQL column name of the encryptable item
tableName	SQL table name of the encryptable item

If the encryptable item is a database file, the EIC is stored in the `eic` field of the `UrBuildingBlockVersionTable`.

## 8.5 Decryption

The application uses the cipher key data for decryption as follows:

1. The application loads the encrypted data, its cipher key ID, and the five components for the initialization vector (RND, SID, EIC, BID, BV) from the NDS database.
2. The application calls the decryption software and provides the encrypted data, the cipher key data corresponding to the cipher key info ID, and the five components for the initialization vector. The decryption software generates the last component for the initialization vector (CTR).
3. The decryption software returns the decrypted data to the application.

---

**Note** The specification of the decryption software and the internal structure of the cipher key data are outside the scope of NDS.

---

The following information can, for example, be stored in the cipher key data:

- Identification of a symmetric key (the symmetric key is stored externally, not in the NDS database)
- Symmetric key encrypted with a public key, which has been provided by the navigation system

## 9 Routing Building Block

The Routing building block groups together the features for routing applications:

- **Route calculation.** The Routing building block contains a topological representation of the road network to be used for calculating routes. Route calculation has to find a route from a given start position (usually the current vehicle position) to a given destination position using the data structures of the Routing building block.
- **Map matching.** The Routing building block contains road geometry data to be used for map matching. Map matching derives the current vehicle position from a raw position that is delivered by GPS or other sensors. The position is then determined on the basis of the digital map represented in the Routing building block.
- **Route guidance.** Road topology and road geometry data is necessary for route guidance. Route guidance checks the current vehicle position against the detailed route computed by route calculation and derives the most adequate maneuver descriptions.
- **Advanced driver assistance (ADAS).** ADAS is supported by an additional attribute layer within the Routing building block. The layer contains an extended set of attributes, such as curvature, road width, slope, etc. For spatial definition of such attributes, additional attribute points are introduced to attach attributes with high positional accuracy to road geometry.

For more information, refer to Section 6.2 *Attribute Layers* on page 82.

Instead of storing their own names, the features of the Routing building block make use of the name features stored in the Name building block via references. Names can thus be shared between building blocks and can be maintained separately in the Name building block. For more information on name features, refer to Chapter 10 *Name Building Block* on page 163.

For speech output for signpost names, the assigned name features are referenced by phonetic transcriptions stored in the Speech building block (see Chapter 14 *Speech Building Block* on page 281).

The Routing building block is a mandatory building block and is the only one of its type in a product database/update region complying with Navigation Data Standard.

### Quick Route Calculation Through a Reduced Number of Links

With the limited resources of navigation systems, long-distance routes cannot be calculated on the basis of large and highly detailed networks within acceptable time. Route calculation therefore benefits from the existence of higher routing levels representing thinner networks. Using higher routing levels with a lower number of links reduces the search space and allows a faster route calculation for applications.

To further reduce the number of links to be evaluated by routing algorithms, connections of links that do not represent intersections in the real-world and thus do not provide a branch for route alternatives are not used for route calculation.

## Maximal Performance of Routing Algorithms

In the Routing building block, the data is structured in a way that enables routing algorithms to run efficiently. A number of algorithms exploit the purely topological network: They run without using any coordinates.

Yet, there are also algorithms that require an estimate of the remaining travel time and travel distance of each path evaluated and such estimates often depend on the distance between each end point of the paths and the destination. Therefore, the coordinates of the start and end points have to be provided. However, no further geometry data is provided for route planning.

## Consideration of the Topological Network at Tile Borders

The data stored in the Routing building block is partitioned on the basis of the global tiling scheme that simplifies data storage. To support route calculation, topological links are not cut at tile borders. This is because rectangular tiles are optimal for geometrical data, but not for topological data.

## 9.1 Building Block Structure and Content

The Routing building block consists of up to five levels (currently L4, L6, L8, L10 and L13). It contains the definitions of the routing features and of building block-specific metadata. On level 13, the functions of the Routing building block share data with the functions of the Basic Map Display building block.

*Figure 9-1 Routing building block structure*

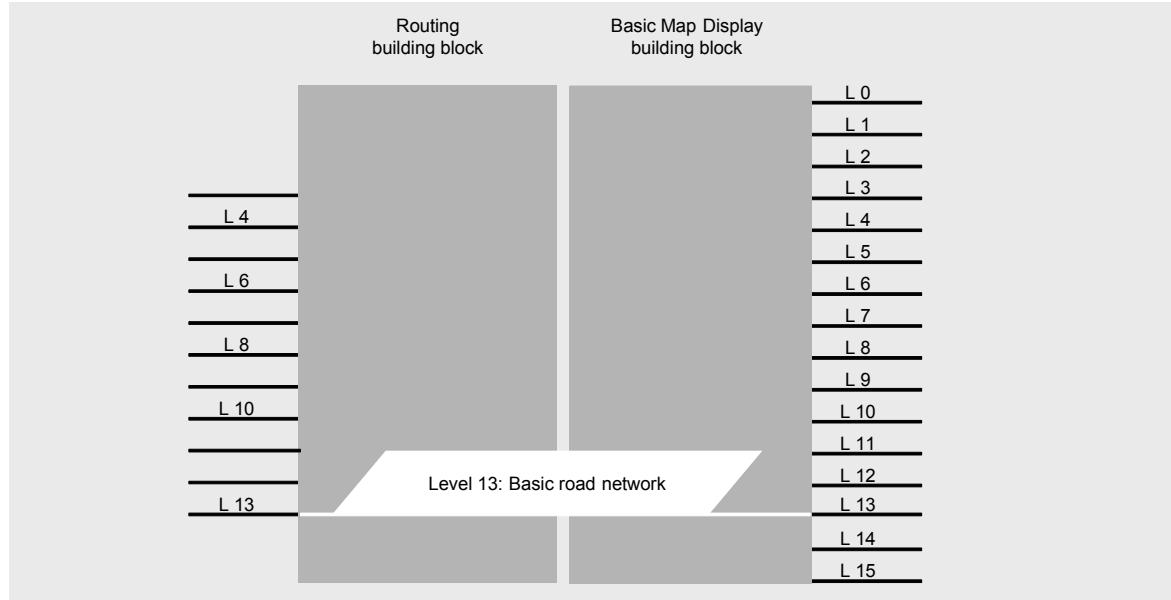


Table 9-1 shows the data contained in the Routing building block.

Table 9-1 Data in Routing building block

Type of data	Description
Routing features	Feature class definitions are provided for all routing features. The following classes are available: <ul style="list-style-type: none"> <li>- Link features (see Section 9.2 <i>Link Feature Class</i> on page 113)</li> <li>- Intersection features (see Section 9.4 <i>Intersection Feature Class</i> on page 126)</li> <li>- Road geometry line features (see Section 9.3 <i>Road Geometry Line Feature Class</i> on page 122)</li> </ul>
Metadata	The metadata of the Routing building block comprises the following: <ul style="list-style-type: none"> <li>- Definition, how many and which levels are provided in the Routing building block (see Section <i>Levels</i> on page 111)</li> <li>- Available attribute types for routing (see Section <i>Routing Metadata</i> on page 111)</li> </ul>

## Attributes and Attribute Layers

The features of the Routing building block are described by a set of fixed attributes and a large number of additional flexible attributes. The flexible attributes are needed to ensure proper support of routing and guidance functions as well as to model functions for map-based driver assistance systems.

A set of these flexible routing attributes is directly integrated in partitions of the routing data (DataScript location: `nds.common.flexibleattributes`):

- Attributes describing properties supporting route options, such as `SCENIC`, `FERRY_TYPE` (ship or train), and `TYPE_OF_PAVEMENT`
- Attributes describing traffic restrictions, such as `PROHIBITED_PASSAGE`, `CONTINUED_TURN_RESTRICTION`, `SPEED_LIMIT`
- Attributes specifying conditions, such as time ranges (`TIME_RANGE_OF_DAY`, `DATE_RANGE_OF_YEAR`) and vehicle types (`FREQUENTLY_USED_VEHICLE_TYPES`, `EQUIPMENT_AND_OCCUPANCY`)
- Length attribute for generalized specifics on upper routing levels, which can be grouped with road class attributes and others (`LENGTH_ALONG_LINK`)
- Attributes used to calculate toll costs of a selected route, for example, `TOLL_VIGNETTE`, `TOLL_CHARGE`, `CURRENCY`

To facilitate attribute handling for the application, the remaining flexible attributes are stored in layers according to their main use. The following attribute layers are currently defined:

- Layer for truck attributes: This layer contains attributes for describing traffic restrictions for truck vehicle types, for example, weight limitation, height/width/length limitation, and road width.
- Layer for route guidance and map display attributes. This layer is only used for level 13 and contains attributes describing maneuvers, right-of-way, traffic lights, and also contains signpost attributes, lane attributes, and toll attributes required for route guidance applications. This layer is stored in the routing geo tile (see Section *Routing Tiles and Routing Geo Tiles* on page 110).
- Layer for name attributes: This layer contains attributes for assigning names to routing features.
- Layer for ADAS attributes: This layer contains attributes describing physical properties, such as curvature, slope, road, and lane width.

For more information on the attribute layer concept, refer to Section 3.3.2 *Attributes* on page 52.

For examples of modelling specific real-world situations by means of flexible attributes, refer to *NDS – Compiler Interoperability Specification*, 17 *Using Flexible Attributes and Attribute Groups* on page 221.

## Routing Tiles and Routing Geo Tiles

The data of the Routing building block is stored in two different tiles, which are stored in separate tables (DataScript location: `nds.routing.main`):

- Routing tile: Contains the data relevant for route calculation, meaning links, intersections, and routing-relevant attributes, such as transition masks.
- Routing geo tile: Contains data which is not relevant for routing, but only relevant for route guidance, map display, and map matching. Examples: Z level information, road geometry lines, shape point lists, references from links to road map lines.

---

**Note**

It is possible that a routing tile is empty, for example, because it is only intersected by route links which start and end outside this tile, but the corresponding routing geo tile is not empty, because it needs to store the road geometry lines for the route links. In this case, the empty routing tile must not be left out: The road geometry lines need the `externalTileIdList` in the routing tile and they may also have fixed attribute sets which are stored in the routing tile (see *A Road Geometry Line can Reference a Set of Fixed Attributes* on page 126). Therefore, a routing geo tile with an external tile ID list and (optionally) a fixed attribute list is required.

---

For a list of routing attributes and their assignment to routing tiles or routing geo tiles, refer to Appendix D *List of Flexible Attributes* on page 381.

## Levels

Within the Routing building block, the number of levels is restricted to five. NDS recommends to use the following levels for storing routing data: L13, L10, L8, L6 and L4. Level 13 is mandatory. The other four levels are optional. Depending on the network size, they can be necessary for optimizing route calculation performance. The optional levels shall only be introduced in consecutive sequence and in a consistent way within one update region. It is not allowed, for example, to use level 8 without level 10 in case level 8 is used somewhere in the update region. Which levels are used for storing the routing data is set during compilation (see *NDS – Compiler Interoperability Specification*, 5.8 *Generalizing Features for Higher Levels* on page 85). The information on levels with routing data is stored in the metadata; DataScript location: `nds.overall.metadata > LevelMetadataTable > hasRoutingTiles`.

Level 13 is the level where most of the routing data is located. This is where the topological data of the road network is enhanced by geometry data which is needed for route guidance, map matching, and map display. In this way, redundant storing of data for routing and map display functions is avoided.

## Routing Metadata

The routing metadata contains information about the available flexible attribute type codes for an update region. This information helps the application to estimate storage space needed for attributes, enables respective functionalities (for example, lane guidance when lane attributes are available), and can also be used for release notes and hints for the test of database content.

DataScript location: `nds.overall.metadata > RoutingMetadata`

To facilitate access and to provide information, which attribute layers are available in the database, the following available attribute layers are listed:

- Layer for Routing attributes: `routingAttributeTypeAvailability`
- Layer for ADAS attributes: `adasAttributeTypeAvailability`
- Layer for route guidance attributes: `guidanceAttributeTypeAvailability`
- Layer for name attributes: `nameAttributeTypeAvailability`
- Layer for truck attributes: `truckAttributeTypeAvailability`

The routing metadata also contains information about eco routing attributes:

- Availability of eco routing attributes: `hasEcoRoutingAttributes`
- Thresholds for excess slope values: `upExessSlopeThreshold` and `downExessSlopeThreshold`

For detailed information on eco routing, refer to 9.9 *Data Structures for Eco Routing* on page 151.

## Dependencies to Name Building Block

The Name building block provides the name information for the features in the Routing building block. Routing applications need the names for generating route lists. On level 13 of the Routing building block, the names are also used for map display (shared use with map display functions).

For more information on how the name information is provided for the routing features, refer to Chapter 10 *Name Building Block* on page 163.

## Dependencies to Basic Map Display Building Block

To enable the graphical representation of a route, references are defined from the upper levels of the Routing building block to the levels of the Basic Map Display building block. On level 13, references are not necessary because the routing functions and the map display functions share the road network data stored in the Routing building block.

For more information, refer to Section 9.5 *References of Routing Features* on page 128.

## Dependencies to Traffic Information Building Block

To enable the automatic consideration of traffic-related information in dynamic route calculation, references are defined to the data of all levels that are used in the Routing building block. Local lists of attributes are organized in tiles on different levels. They set up references between location codes (coming from messages with information about traffic events) and NDS features in the Routing building block.

For more information, refer to Section 12.2.5 *Linking TMC Information to NDS Features* on page 237.

## Dependencies to Junction View Building Block

Junction views give the driver realistic visual information of a junction and the required maneuver. To indicate that junction view information is available, the flexible attribute HAS\_JUNCTION\_VIEW is assigned to links. The references between routing features and the junction views are stored in the following tables:

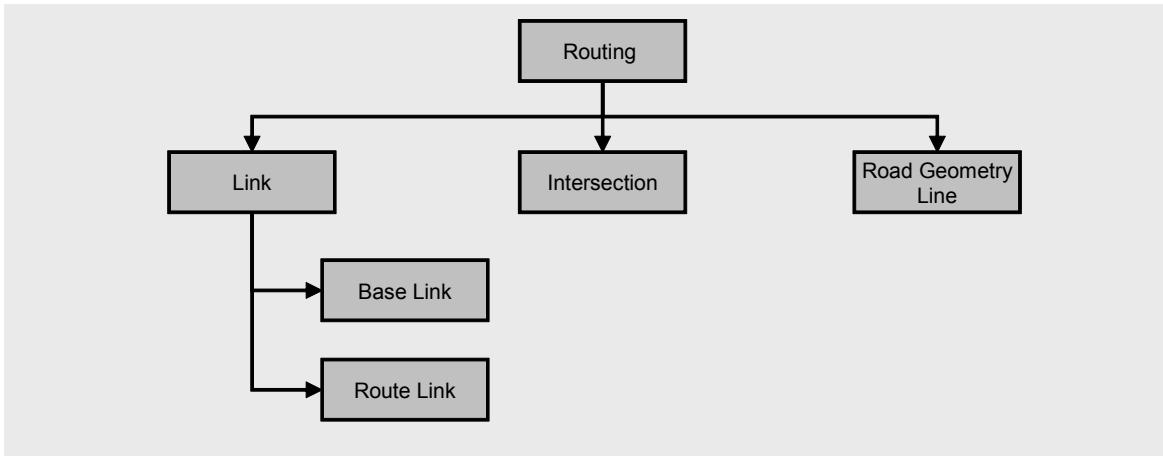
- JvFromLink (DataScript location: `nds.jv.main`) containing references from junction view images to routing information (`fromLink`)
- JvFrom2ToLink (DataScript location: `nds.jv.main`) containing references from junction view images to routing information defining a route (`fromLink` and `toLink`)

For more information, refer to Chapter 18 *Junction View Building Block* on page 341.

## Overview of Routing Features

The figure below gives an overview of the features of the Routing building block. The following sections offer basic conceptual information. Detailed information on the individual feature classes used to represent the routing features in the database as well as on their properties can be found in Sections 9.2 to 9.4 (page 113 ff.).

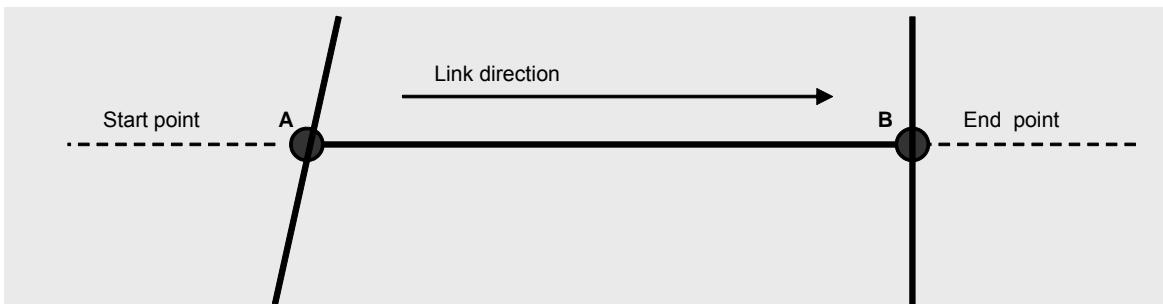
Figure 9-2 Logical diagram of routing features



## 9.2 Link Feature Class

A link describes a road segment between two intersections. In the real world, it represents a road or a carriageway. Carriageways of a freeway or other dual-carriageway roads are represented by a link for each carriageway.

Figure 9-3 Link between two intersections



Each link has a **start point** and an **end point**. The start and end points are not represented as independent features in the database. They always belong to the intersection at the start or end of the link, respectively.

Figure 9-3 shows a link with a direction from intersection A (start point) to intersection B (end point). The direction from the start to the end is labeled **forward direction**. The **backward direction** is the other way round.

The link is **outbound** at intersection A and **inbound** at intersection B in forward direction. In backward direction, it is inbound at intersection A and outbound at intersection B (see *NDS – Compiler Interoperability Specification*, 5.1 Adjusting Direction of Links on page 80).

---

<b>Note</b>	The link direction does not describe the <b>traffic flow direction</b> . The link direction and the traffic flow direction do <b>not</b> need to be identical. The traffic flow is generally assumed to be bidirectional, unless it is restricted by corresponding attributes.
-------------	--

---

Link features are characterized by the following main properties:

- Fixed and flexible attributes, see Section 9.2.3 *Attributes of Link Features* on page 117
- Road geometry, see Section 9.2.4 *Road Geometry for Link Features* on page 120
- References to other routing features and names, see Section 9.2.5 *References of Link Features* on page 120

Navigation Data Standard differentiates two types of links:

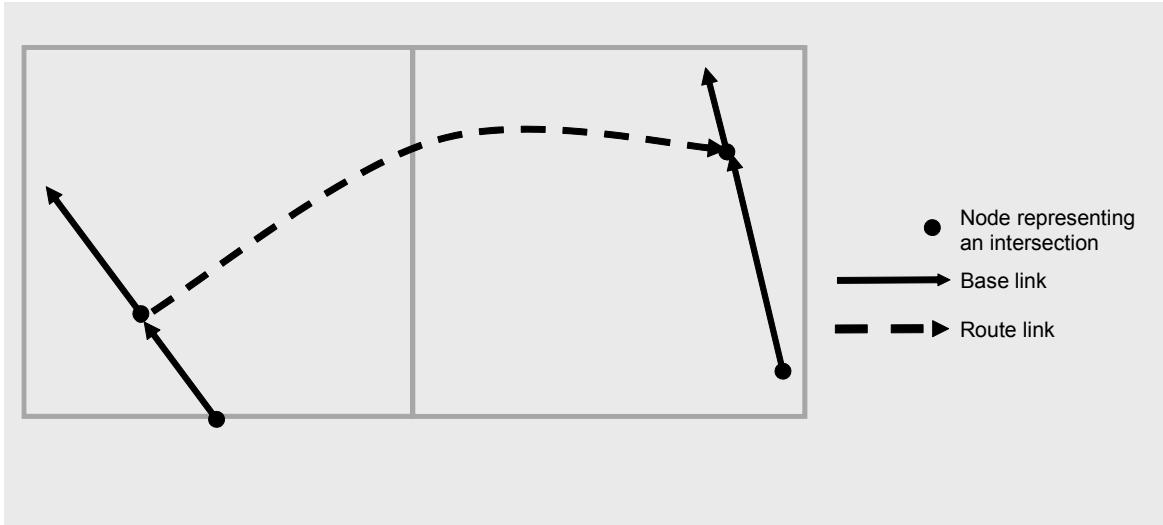
- Links located on one tile only (so-called base links)
- Links that extend over more than one tile (so-called route links)

### 9.2.1 Base Links

The majority of links that have to be mapped to the database are located on one tile only; these links are called *base links*. To achieve maximum efficiency, geometry and topology information of base links is stored directly with the link.

Base links exist only on level 13 of the Routing building block. On upper levels of the database, road geometry is stored in the Basic Map Display building block only.

Figure 9-4 Links located on one tile only and links extending over more than one tile



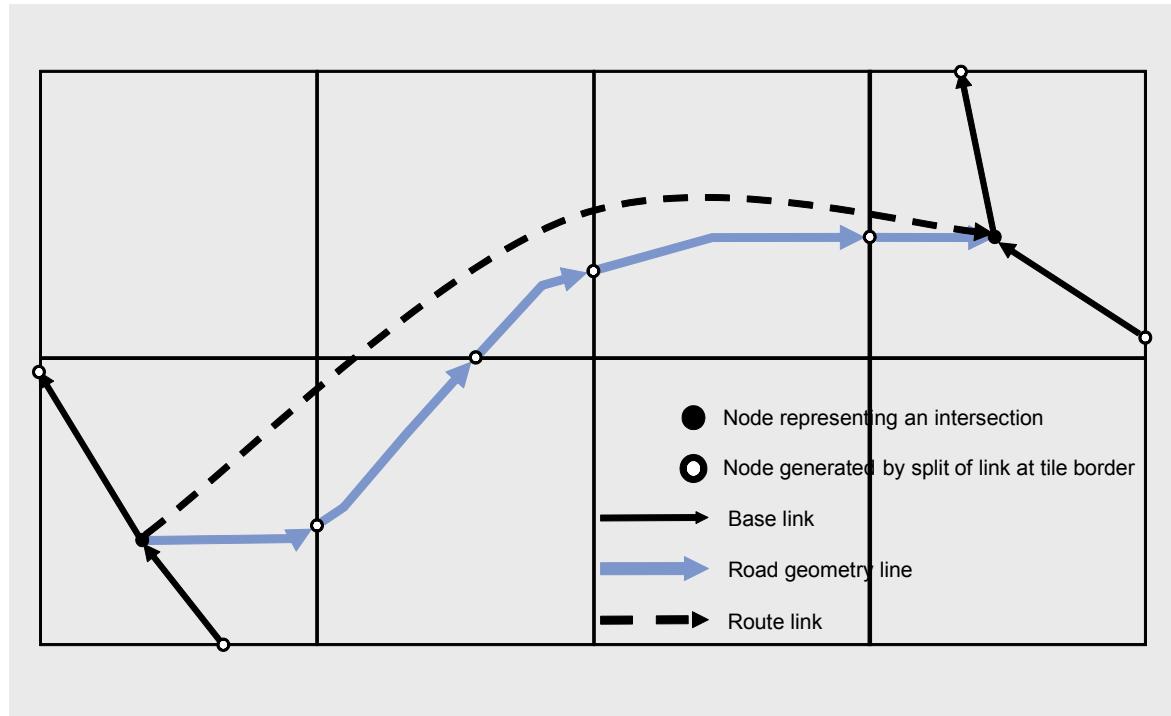
A base link feature has its complete geometry and its intersections located within one tile. Whether a link is stored as a base link or a route link on level 13 is determined during compilation (see *NDS – Compiler Interoperability Specification*, 5.3 *Splitting Road Features* on page 81).

### 9.2.2 Route Links

In contrast to base links, route links are purely topological routing features. On level 13 of the Routing building block, route links have to be supplemented by road geometry data, for example, for map matching, route guidance and map display. Road geometry is implemented via road geometry line features. For more information, refer to Section 9.3 *Road Geometry Line Feature Class* on page 122 and Section 9.2.4 *Road Geometry for Link Features* on page 120.

On level 13, route links are used exclusively for links that extend over more than one tile as opposed to base links that are located on one tile only. The road geometry of a route link is clipped at the tile borders and represented by individual road geometry line features. To access the geometry, references are defined between the route link and all tiles where corresponding road geometry lines can be found.

Figure 9-5 Route links on level 13 of the Routing building block



On the upper levels of the Routing building block, route links are used exclusively for a topological representation of the road network. On these levels, a route link can be located on one tile or can extend over several tiles. The road geometry is contained in the Basic Map Display building block only where it is represented by line features. If the route link is visible in the corresponding map display levels, meaning if its length is bigger than one pixel, it must have a reference to at least one map line of the respective basic map display level.

**Example** Level 10 has route links, and level 10, 11 and 12 are used for map display. In this case, the level 10 route link refers to map lines on level 10, 11 and 12 that represent its geometry. This is required as there are no map line relations between different levels.

A route link is always stored in the tile on which its start point is located. Whether a link is stored as a base link or a route link on level 13 is determined during compilation.

#### Related Topics:

- *NDS – Compiler Interoperability Specification, 5.3 Splitting Road Features on page 81*

### 9.2.3 Attributes of Link Features

The **fixed attributes** listed in Table 9-2 belong to link features. They are automatically assigned to all features within the link feature class. A corresponding attribute value must be provided when the feature is instantiated.

*Table 9-2 Fixed attributes of link features*

Attribute	Description
<b>In nds.common.fixedattributes</b>	
AdminRoadClass	<p>Gives information on the official classification of a road This classification usually corresponds to the classification defined by the authorities of a country or federal state. Lower values stand for higher road classes. 0 is used for an unknown administrative road class, 1 is always the highest road class (for example, freeway or Autobahn). The meaning of the individual values 1...7 is country-specific.</p> <p><b>Note</b> The producer of a database shall ensure that administrative road classes are comparable between adjacent countries. This can imply that values are left unused in countries with less administrative classes than an adjacent country, and that road classes must be merged in countries with more administrative road classes.</p>
FunctionalRoadClass	<p>Gives information on the importance of a road within the network Low numbers (0) describe roads of high importance, whereas higher numbers indicate lower importance.</p>
LINK_TYPE	<p>Describes properties of a link which are mainly needed for route guidance purposes Indicates, for example, whether a link is a ramp, part of a roundabout, a parallel lane in a freeway interchange, a service road, a main road having service roads, a square, a pedestrian zone or the main carriageway in a freeway interchange.</p> <p><b>Caution</b> There is also a flexible attribute for the link type (LINK_TYPE) that is grouped with LENGTH_ALONG_LINK and assigned to aggregated links on higher levels in order to provide information about the type of the links that have been aggregated. This flexible attribute must not be used on level 13.</p>
<b>In nds.common.fixedattributes &gt; RoutingRoadAttributes</b>	

Attribute	Description
urban	Indicates whether a link is located in an urban, built-up area This information can, for example, be used to estimate an average speed for route calculation or to calculate routes that avoid city areas.
motorway	Indicates that the link is part of a motorway. Can be evaluated by routing applications.
complexIntersection	Indicates whether the link belongs to a complex intersection This attribute is needed to assist guidance to detect all links belonging to an intersection.
pluralJunction	Indicates that a junction is made up of multiple links. It is used to indicate whether a driver will perceive a junction as one or as multiple intersections. The fixed attribute <code>pluralJunction</code> also indicates that a maneuver may require different explication than implied by the geometry. For more information, see <i>Differences Between Plural Junction and Intersection</i> on page 119.
insideCityLimitStatus	Used to indicate if a link is inside a city limit or not. The attribute is used for advanced driver assistance and can be assigned to base and route links, and to road geometry lines on all levels. The additional status UNKNOWN is required for ADAS applications which need to switch off in case of unclear situations. This attribute can be used, for example, for switching off/on the upper beam automatically if the car comes from a link outside the city limit to a link inside, and vice versa. Whereas <code>urban</code> can only be used for build-up areas, applications can use <code>insideCityLimitStatus</code> also outside build-up areas.

In `nds.common.fixedattributes > SharedRoadAttributes`

<code>travelDirection</code> <code>FERRY</code> <code>TUNNEL</code> <code>TOLL</code>	Indicates whether a link is: <ul style="list-style-type: none"> <li>- A one way link and if so in which direction</li> <li>- Part of a ferry connection</li> <li>- Partly a tunnel or a bridge</li> <li>- Part of a toll way or has a toll booth; if so, the direction for which the toll must be paid is given</li> </ul>
--	---

In `nds.routing.link > Link`

<code>averageSpeed</code>	Average speed (for example, in km/h) that cars driving in normal traffic flow can achieve on the road represented by the link The value for the average speed is calculated during compilation based on information from the input data.
<code>startAngle</code> <code>endAngle</code>	
<code>length</code>	Specifies the length of a link in centimeters The value must be greater than zero.

A large number of **flexible attributes** is available for links. In contrast to the fixed attributes above, they are only assigned to individual features of the link feature class when they are required to model a real-world situation.

Flexible attributes are a powerful means to map, for example, complex traffic regulations. Typical flexible attributes for base link features in the Routing building block are:

- SPEED\_LIMIT
- Entry restrictions, such as PROHIBITED\_PASSAGE
- TIME\_ZONE\_REFERENCE: One tile can intersect several time zones. In this case, an additional attribute can be assigned to links to specify a time zone which is different from the time zone assigned to the tile as a whole (see Section 5.8 Time Zones on page 79).

The flexible attribute ENHANCED\_GEOMETRY indicates that a link has a higher geometry precision than usual. If the ADAS attribute layer is not used, this attribute may be assigned to a link to enhance the map matching functionality.

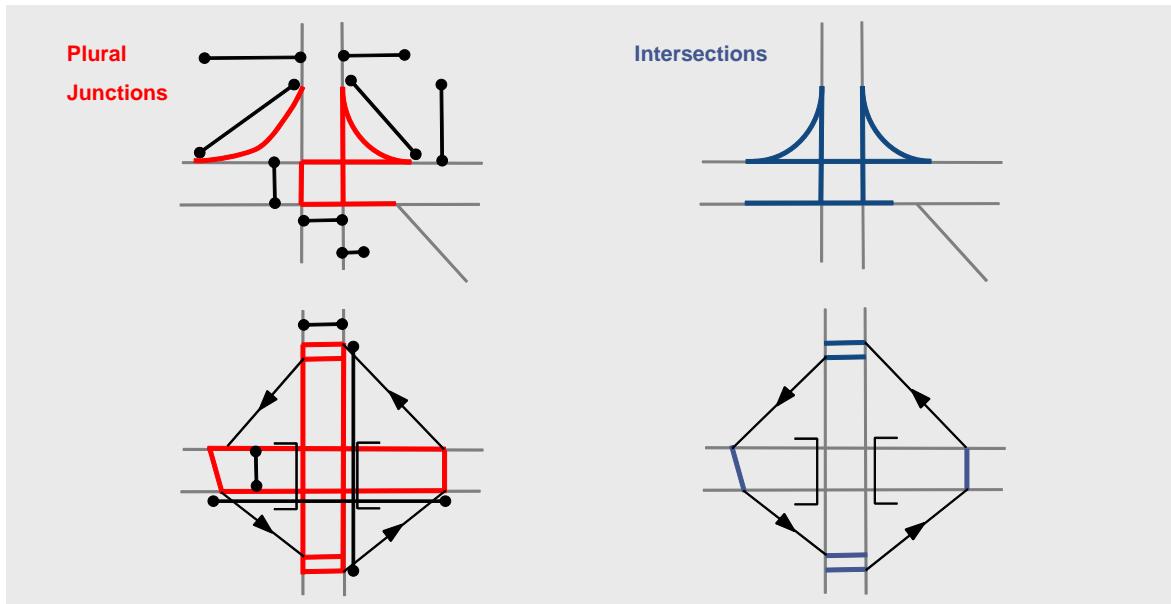
### Differences Between Plural Junction and Intersection

The capturing rules for plural junctions and intersections are completely different. Table 9-3 and Figure 9-6 show the main differences:

*Table 9-3 Differences between plural junction and intersection*

Plural Junction	Intersection
GDF level 1 feature	GDF level 2 feature
Attribute of road elements	Composed of road elements and/or junctions
No opposing nodes/sides	Opposing nodes/sides

*Figure 9-6 Examples of plural junctions and intersections*



## 9.2.4 Road Geometry for Link Features

Road geometry for link features is described by means of *shape points* and stored in the routing geo tile. The shape point information for all links in the link list of the routing tile is stored in the **ShapePointInfoList** of the routing geo tile. The sequence of links in the link list corresponds to the sequence of **ShapePointInfo** entries in the shape point info list so that the shape point information for a specific link can be easily identified.

Each **ShapePointInfo** entry contains the following information:

- **ShapePointType**: Indicates whether a link is a route link or a base link.
- **ShapePointList**: Indicates the geometry of a base link by a list of shape points. Geometry information is not stored in a shape point list if the base link consists only of a start and an end point connected by a straight line. See also *Definition of Geometry for Base Links* on page 120.

If the link is a route link, the road geometry of the link is defined in the road geometry lines, which are also stored in the routing geo tile. For detailed information on road geometry lines, refer to Section 9.3 *Road Geometry Line Feature Class* on page 122.

DataScript locations:

- Routing geo tile: `nds.routing.main > RoutingAuxTileTable`
- **ShapePointInfoList**: `nds.routing.link`
- **ShapePointList**: `nds.routing.link`

### Definition of Geometry for Base Links

The shape point list defines the geometry of base links and includes all shape points between the start and the end point of a link. Each shape point has a pair of WGS 84 coordinates describing its geographic position.

On the basis of the shape point list, the geometry is described as a polyline obtained by joining the start point to the first shape point in the list, each shape point to its successor and the last shape point in the list to the end point.

## 9.2.5 References of Link Features

The following references are relevant for link features:

### A Link is Referenced by Intersections

A link is always referenced by the two intersections that represent its start and its end. Note that a link is also referenced by two intersections if it represents a dead-end road.

## A Base Link can Reference a Route Link on a Higher Level of the Building Block

For route calculation purposes, inter-level references can be defined for links: A base link can have a route link on a higher level as a parent.

A reference from a base link to a route link can either be defined on the subsequent level or it can be used across several generalization stages, for example, a reference from a base link on level 13 to a route link on level 8. For more information on inter-level references, refer to Section 9.5 *References of Routing Features* on page 128.

## A Link can Reference a Named Object in the Name Building Block

The Name building block provides the name information necessary for a link. For this purpose, name references can be established from a link feature in the Routing building block to the following named object features in the Name building block:

*Table 9-4 References from links to named objects*

Referenced named object feature	Purpose
Road	Generating route lists, location input
Signpost	Displaying signposts
House	Displaying houses, location input
Region (administrative area)	Determining the respective administrative area and retrieving its metadata, such as legal speed limits

For more information, refer to Table 9-8 on page 129 and Chapter 10 *Name Building Block* on page 163.

## A Link can be Referenced by Junction View Images

Junction views give the driver realistic visual information of a junction and the required maneuver. To indicate that junction view information is available, the flexible attribute HAS\_JUNCTION\_VIEW is assigned to links. The references between routing features and junction views are stored in the following tables:

- JvFromLink (DataScript location: nds . jv . main) containing references from junction view images to routing information (*fromLink*)
- JvFrom2ToLink (DataScript location: nds . jv . main) containing references from junction view images to routing information defining a route (*fromLink* and *toLink*)

For more information, refer to Chapter 18 *Junction View Building Block* on page 341.

## A Link can be Referenced by TMC Locations

References from TMC locations to features in the Routing and the Basic Map Display building block are stored as flexible attributes in tiles of the TMC building block. The TMC attributes assign location codes and TMC-relevant properties, such as direction and flags, to base links, route links, road geometry lines and map lines. References are established for each location code. Thus, location codes may be assigned to more than one feature.

For more information, refer to Section *Local Attribute Lists* on page 245.

## 9.2.6 References of Route Links

The following references are only used for link features extending over more than one tile:

### A Route Link can be Referenced by Two or More Road Geometry Lines

For map matching and route guidance, references can be provided between the route link and all tiles where road geometry lines are located that belong to the route link (relevant only on level 13 of the Routing building block).

### A Route Link on an Upper Level Must Reference Links on the Next Lower Level

A route link must reference route links on the next lower level or – if the next lower level is level 13 – base links and/or route links.

### A Route Link can Reference One or Several Road Map Lines in the Basic Map Display Building Block

To display a route in a map, references may be defined from a route link on one of the upper levels of the Routing building block to one or several road map lines in the Basic Map Display building block. The references enable the link's graphical representation. References to road map lines are stored in the routing geo tile.

If the referenced map line is in the same tile ID as the route link, it is not necessary to store the tile ID.

## 9.3 Road Geometry Line Feature Class

Route links are purely topological routing features. On level 13 of the Routing building block, they are supplemented by features of the road geometry line feature class. Road geometry lines contain the geometry information needed, for example, for map matching, route guidance, and map display.

Road geometry lines have all the attributes necessary for the graphical representation of roads. They contain a sequence of shape points describing a polyline that represents the course of a road. The direction of a road geometry line is defined by the sequence of these points. Road geometry lines belonging to route links must have the same direction as the route link.

<b>Note</b>	Detailed road geometry can be described by ADAS attributes, such as clothoids. For more information, refer to Section 9.6 <i>Data Structures for Advanced Driver Assistance (ADAS)</i> on page 134. The flexible attribute ENHANCED_GEOMETRY indicates that a link has a higher geometry precision than usual. If the ADAS attribute layer is not used, this attribute may be assigned to a link to enhance the map matching functionality.
-------------	---

Road geometry line features are characterized by the following main properties:

- Fixed and flexible attributes
- References to other routing features
- Road geometry
- Drawing order

### 9.3.1 Attributes of Road Geometry Line Features

The **fixed attributes** listed in Table 9-5 are relevant for road geometry line features. They are automatically assigned to all features within the road geometry line class. A corresponding attribute value must be provided when the feature is instantiated.

DataScript location: `nds.routing.line`

*Table 9-5 Fixed attributes of road geometry line features*

Attribute	Description
<code>numVertices</code>	Number of vertices of the road geometry line
<code>shapes</code>	Shape point information for the line. Start and end points are always contained in the shape point lists defined for road geometry lines.
<code>routeLinkFeatureId</code>	Identifies the route link that this road geometry line belongs to
<code>ordinalNumber</code>	Ordinal number of the road geometry line within a route link, starting at 0
<code>averageSpeed</code>	To support the graphical representation, the road geometry line can be categorized according to the average speed that cars driving in normal traffic flow can achieve on the road represented by the road geometry line.  The value for the average speed is calculated during compilation based on information from the input data.  DataScript location: <code>nds.routing.link</code>

Attribute	Description
<code>administrativeRoadClass</code>	<p>Gives information on the official classification of a road to support the road's graphical representation</p> <p>This classification usually corresponds to the classification defined by the authorities of a country or federal state.</p> <p>Lower values stand for higher road classes. 0 is used for an unknown administrative road class, 1 is always the highest road class (for example, freeway or Autobahn). The meaning of the individual values 1...7 is country-specific.</p> <p>DataScript location: <code>nds.common.fixedattributes &gt; SharedRoadAttributes</code></p>
	<p><b>Note</b></p> <p>The producer of a database shall ensure that administrative road classes are comparable between adjacent countries. This can imply that values are left unused in countries with less administrative classes than an adjacent country and that road classes must be merged in countries with more administrative road classes.</p> <p>For more information, refer to <i>NDS – Compiler Interoperability Specification, 5.11 Handling Urban Indicator, Administrative Road Class, and Traffic Sense</i> on page 93.</p>
<code>functionalRoadClass</code>	<p>Gives information on the importance of a road within the network to support map display</p> <p>Lower numbers (0) describe roads of higher importance, whereas higher numbers indicate lower importance.</p> <p>DataScript location: <code>nds.common.fixedattributes &gt; RoutingRoadAttributes</code></p>

Attribute	Description
travelDirection ferry tunnel bridge toll	Indicate whether the road geometry line represents: <ul style="list-style-type: none"> <li>- A one way link and if so in which direction</li> <li>- Part of a ferry connection</li> <li>- Partly a tunnel or a bridge</li> <li>- Part of a toll way or has a toll booth; if so, the direction for which the toll must be paid is given</li> </ul> DataScript location: <code>nds.common.fixedattributes &gt; SharedRoadAttributes</code>
linkType	Describes properties of a road geometry line which are mainly needed for geometrical requirements in map matching and map display. Typical link types are main road, slip road, or traffic circle. This information can also be used for route guidance. DataScript location: <code>nds.common.fixedattributes &gt; SharedRoadAttributes</code>

A large number of **flexible attributes** is available for road geometry lines. In contrast to the fixed attributes above, they are only applied to individual features of the road geometry line feature class when they are required to model a real-world situation.

For road geometry lines, all flexible attributes that are available for route links and base links can be defined, provided that the attribute can be represented graphically in a map or provided it can be used by map matching or route guidance.

Typical flexible attributes for road geometry line features in the Routing building block are:

- SLOPE
- SCENIC (tourist route type)
- TYPE\_OF\_PAVEMENT

### 9.3.2 Drawing Order of Road Geometry Line Features

For road geometry lines, the same drawing order rules as for features of the Basic Map Display building block apply, see 11.9 *Drawing Order* on page 215.

In general, road geometry lines should be stored in the same order as they are required to be drawn. If this general rule cannot be applied, for example in case of incremental updates, the DRAWING\_ORDER shall be assigned to the road geometry lines.

### 9.3.3 References of Road Geometry Line Features

The following references are relevant for road geometry line features:

#### A Road Geometry Line References a Route Link

To describe the road geometry of a route link for map display, references can be defined from one or several road geometry lines to one route link.

## A Road Geometry Line can Reference a Named Object in the Name Building Block

The Name building block provides the name information necessary for a road geometry line. For this purpose, name references can be established from a road geometry line feature in the Routing building block to the following named object features in the Name building block:

*Table 9-6 References from road geometry lines to named objects*

Referenced named object feature	Purpose
Road	Generating route lists, location input
Signpost	Displaying signposts
House	Displaying houses, location input

For more information, refer to Table 9-8 on page 129 and Chapter 10 *Name Building Block* on page 163.

## A Road Geometry Line can be Referenced by Junction View Images

Junction views give the driver realistic visual information of a junction and the required maneuver. To indicate that junction view information is available, the flexible attribute HAS\_JUNCTION\_VIEW is assigned to road geometry lines. The references between routing features and junction views are stored in the following tables:

- JvFromLink (DataScript location: nds.jv.main) containing references from junction view images to routing information (`fromLink`)
- JvFrom2ToLink (DataScript location: nds.jv.main) containing references from junction view images to routing information defining a route (`fromLink` and `toLink`)

For more information, refer to Chapter 18 *Junction View Building Block* on page 341.

## A Road Geometry Line can Reference a Set of Fixed Attributes

The reference to a `FixedRoadAttributeSet` is stored as follows:

- For the reference type EXPLICIT, the referenced `FixedRoadAttributeSet` is stored with the road geometry line structure.
- For the reference types INT8 and INT16, the referenced `FixedRoadAttributeSet` is stored in the `FixedRoadAttributeSetList` of the corresponding routing tile.

## 9.4 Intersection Feature Class

An *intersection* represents a road junction where two or more roads meet or cross. An intersection is also used to model the end point of a dead-end road. Two roads that cross each other but are not connected topologically do not form an intersection.

Navigation Data Standard uses the concept of *transitions* to model the way from one link to another through an intersection. Transitions represent the topological connection of links through intersections. Transitions are not represented as independent features but as properties of the corresponding intersection.

Intersections bind one or more base and/or route links. An intersection is represented by exactly one point in the road network.

Intersections are characterized by the following main properties:

- Position
- Transitions
- Number of links
- Fixed and flexible attributes
- References to other routing features
- Name references

#### 9.4.1 Position

Each intersection has a pair of coordinates defining the intersection's geographic position relative to the center of the tile on which it is located.

#### 9.4.2 Transitions

An intersection has topological connections from one connected link to another. Flexible attributes are used to define which combinations of connected links form valid transitions.

Any pair of links that is connected to an intersection can be combined to form a transition. A link can also be combined with itself to model a U-turn. Thus, assuming that  $n$  is the total number of connected links,  $n \times n$  link transitions are possible. By means of the `TRANSITION_MASK_*` flexible attributes, it is determined which transitions exist.

Transitions can have attributes, such as right of way, traffic lights, speed profiles, curvatures, and turning restrictions.

#### 9.4.3 Number of Links

This property specifies how many base links and/or route links are connected to an intersection.

#### 9.4.4 Attributes of Intersection Features

A large number of **flexible attributes** is available for intersection features. They are only applied to individual features of the Intersection feature class when they are required to model a real-world situation.

Flexible attributes are a powerful means to map, for example, maneuver descriptions. Typical flexible attributes for intersection features in the Routing building block are, for example, the `TRANSITION_MASK_*` attributes, which indicate allowed and not allowed driving maneuvers at intersections.

### 9.4.5 References of Intersection Features

The following references are relevant for intersection features:

#### An Intersection References a Link

An intersection always references at least one base link or route link that is connected to it. The intersection represents either its start or end point.

Flags are used to denote whether:

- The link is a base link or a route link.
- The intersection forms the start or the end of the link.

#### An Intersection can Reference a Named Object in the Name Building Block

The Name building block provides the name information necessary for an intersection. For generating route lists, a name reference can be established from an intersection feature in the Routing building block to a road named object feature in the Name building block.

For more information, refer to Table 9-8 on page 129 and Chapter 10 *Name Building Block* on page 163.

## 9.5 References of Routing Features

The following referencing patterns are relevant for the Routing building block:

- References between routing features on different tiles
- References to features in the Name building block and Basic Map Display building block
- References between routing features on different levels of the Routing building block

---

<b>Note</b>	References between routing features that are located on the same tile are not explicitly mentioned here. They are described in Sections 9.2.5, 9.2.6, 9.3.3, and 9.4.5 for the individual feature classes.
-------------	--

---

### 9.5.1 Inter-Tile References

References exist between features that are stored in different tiles of the same level of the Routing building block. The references have to be stored in a way that allows compiling individual tiles independent of the existing references.

For more information, refer to *NDS – Compiler Interoperability Specification*, 5.12.1 *References between Features on Different Tiles* on page 94.

The following table shows the inter-tile references that are relevant for routing features. Note that the logical relation is always bidirectional. Yet, only the direction listed in the table is stored physically. The opposite direction has to be established by the application when loading the tile data.

*Table 9-7 Inter-tile references for routing features*

From	To
Intersection	Link (base link or route link)
Road geometry line	Route link

## 9.5.2 Inter-Building Block References

References exist between features that are stored in different building blocks:

- From routing features to name features
- From routing features to basic map display features

These references are stored in the `RoutingGeoTile` (DataScript location: `RoutingAuxTileTable`). As references to name features are required for level 13 and higher levels, and references to basic map display features are required for higher levels, the `RoutingAuxTileTable` must be filled on all routing levels.

For more information, refer to *NDS – Compiler Interoperability Specification, 5.12.3 References to Features in Other Building Blocks* on page 95.

### Dependencies to Name Building Block

The Name building block provides the name information for the features in the Routing building block. Routing applications need the names for generating route lists. Names are a prerequisite for identifying route destinations in the road network.

The following table shows the name references that can be established from the routing features to the named object features:

*Table 9-8 References from routing features to named object features*

Routing feature	Named object feature
Base link, route link, road geometry line, intersection	House
Base link, route link, road geometry line, intersection	Road
Base link, route link, road geometry line, intersection	Region (administrative area)
Base link, route link, road geometry line, intersection	Signpost
Intersection	Crossroad

For more information on how the name information is provided for the routing features, refer to Chapter 10 *Name Building Block* on page 163.

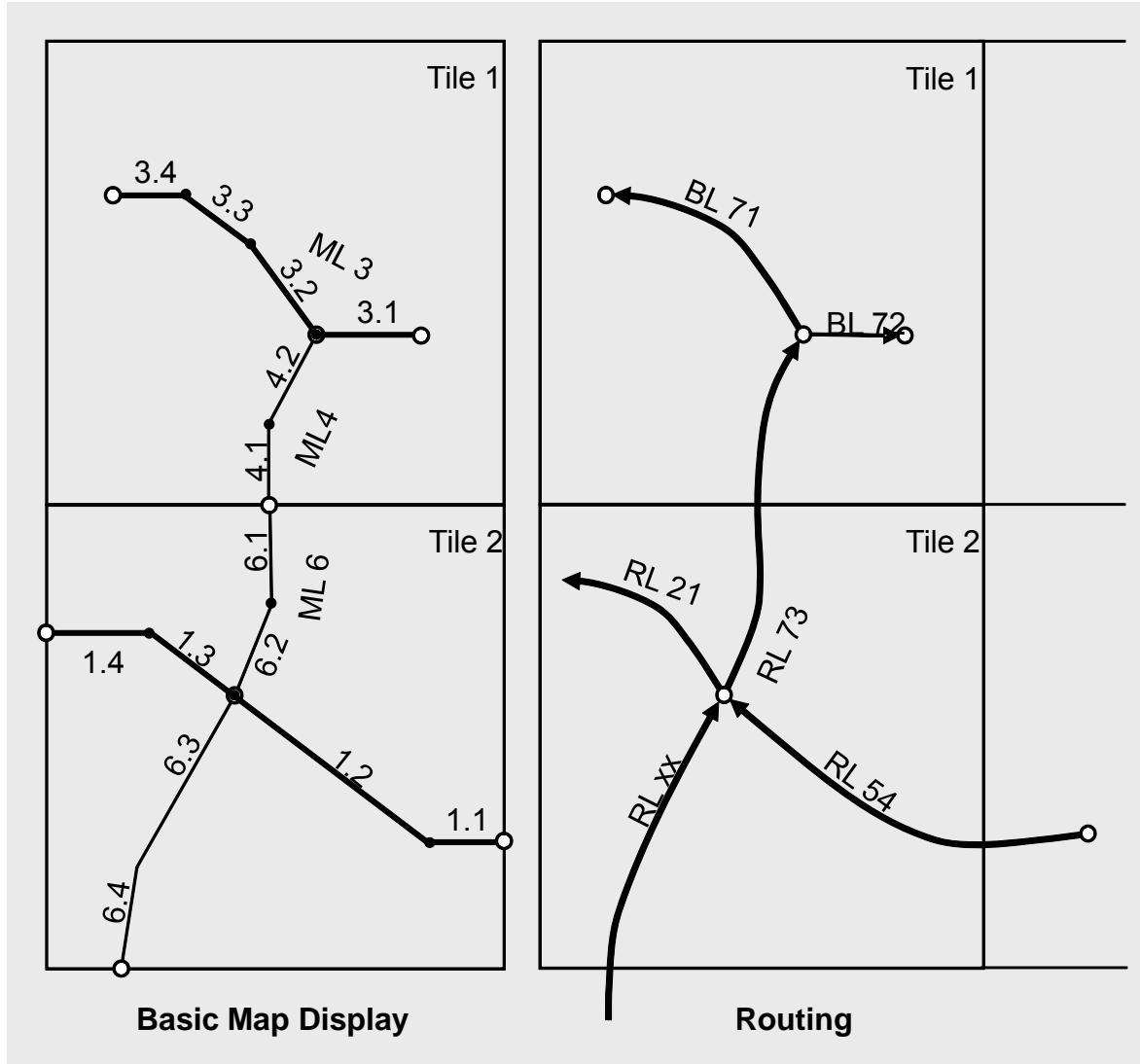
## Dependencies to Basic Map Display Building Block

To enable the graphical representation of a route, references are defined from route links in upper routing levels to their corresponding map lines in the Basic Map Display building block. On level 13, references are not needed because routing functions and map display functions share the road network data stored in the Routing building block.

The references are defined from route link features to map line features in the Basic Map Display building block. Due to different compiler optimizations it is neither guaranteed that route links and map lines have a one-to-one correlation, nor that they are always oriented in the same direction. The references, therefore, need to support ranges and an orientation flag.

Figure 9-7 shows two tiles each in the Basic Map Display and the Routing building block. The route link RL73 in tile 2 refers to two sections of the map line ML6 in tile 2 of the Basic Map Display data, and to two further sections of map line ML4 in tile 1 (these two sections make up the whole map line ML4).

Figure 9-7 References of routing and basic map display features



The corresponding references are:

- Tile 2: Map line ML6, start section 6.2, number of sections 2, reverse direction
- Tile 1: Map line ML4, start section 4.1, number of sections 2, forward direction (as ML4 has only two sections, it is not necessary to specify a range; the reference type TOTAL\_MAP\_LINE is sufficient).

**Note** A map line may extend across intersections. Thus, different links can refer to different parts of a map line.

Map line references use an index starting with 0. A map line with four points and three segments, for example, has the following index:

- Segment 0: (point 0 – 1)
- Segment 1: (point 1 – 2)
- Segment 2: (point 2 – 3)

### 9.5.3 Inter-Level References

References between road networks on the different levels of the Routing building block are in upward or downward direction:

- **Upward references**, for example, from level 13 to level 10, are needed for route calculation. They are necessary for propagating the travel time and travel distance of paths to links on higher levels in order to continue route calculation there.  
Upward references are also needed for manual or automatic blocking as a result of traffic information.
- **Downward references** are also needed for route calculation. They are also used for mapping a route that has already been calculated on a different level to routes on lower levels (down to level 13) that can be used for route guidance.

For more information, refer to *NDS – Compiler Interoperability Specification, 5.12.2 References between Features on Different Levels* on page 95.

*Figure 9-8 Upward and downward references between links on different levels*

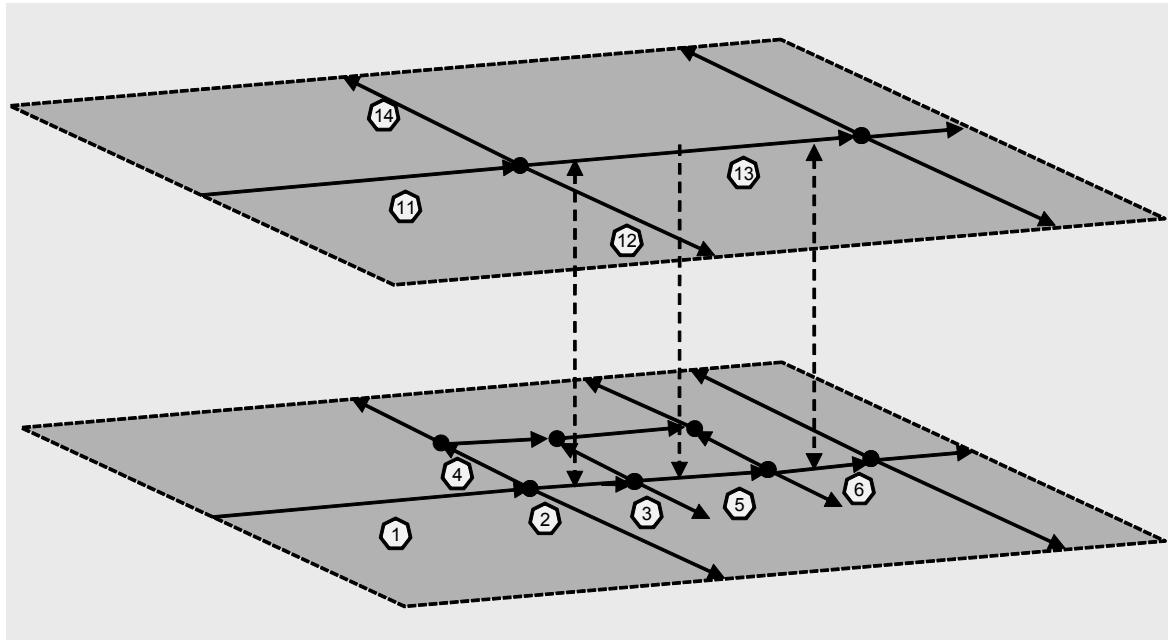


Figure 9-8 shows two levels of a road network. The network on Level 13 is denser than the network on level 10, the higher level. The links 3, 5 and 6 on level 13 have been generalized to form a common link on level 10. The first and the last link on level 13 reference the generalized link on level 10 (upward references). The generalized link on the next upper level references all links on level 13 (downward references).

### Upward References

Upward references can be defined between the following routing features:

- From a base link (stored on level 13 only) to a route link on level 10
- From a route link to a route link on the next higher level

References may only be defined to links on subsequent levels.

In order to propagate the travel time and travel distance of paths to links which are stored on higher levels, it is not necessary that all links on the lower level reference the link on the higher level.

### Downward References

References to lower levels can be defined between the following routing features:

- From a route link to route links on the next lower level
- From a route link on level 10 to base links on level 13

Like upward references, downward references may only be used between subsequent levels.

In order to map a route that has already been calculated on a higher level to a more detailed route on level 13, the link on the higher level must explicitly or implicitly reference all links on the lower level that belong to it. To optimize storage use, some of these explicit references may be omitted physically, as long as the other references do not become ambiguous.

Downward references may be omitted if a higher level link maps onto more than two links on the next lower level tile and if these lower level links constitute a sequence. If the sequence is not interrupted by crossing links of the same or lower functional road class, all downward references to links in-between first and last link of the sequence may be omitted. References to the first and last link in the sequence are required for routing and shall not be omitted.

After optimization, a higher level link with two downward references can map to either two related links or a sequence of links on the lower level. Consequently, an application always has to check for omitted links.

## 9.6 Data Structures for Advanced Driver Assistance (ADAS)

Advanced Driver Assistance (ADAS) functionality is a system-based functionality which is supported by map data. NDS supplies data structures which make it possible to supply appropriate data to different ADAS protocols.

NDS supports ADAS functionality by means of ADAS attributes, enhanced ADAS attributes, and additional ADAS geometry data. The approaches build on each other and, thus, enable a different level of detail for ADAS functions depending on navigation system and use case. Another reason for separating ADAS attributes and ADAS geometry is that ADAS attributes may also be used for other purposes, for example, map matching.

The routing metadata contains the information whether ADAS data is available for the update region (sequence `adasAttributeTypeAvailability` in `nds.overall.metadata > RoutingMetadata`). See also *Routing Metadata* on page 111.

The region metadata table contains the information which type of ADAS data is available for the given region (DataScript location: `nds.overall.metadata > RegionMetadataTable`). The ADAS availability type provides the following information:

- Data contains basic ADAS information, such as physical width, curvature, slope, and turn geometry (`hasAdasBasicTypes`)
- Data contains extended ADAS information, such as extended clothoid and gradient data (`hasAdasExtendedTypes`)
- Data contains spline and 3D vector information (`hasAdasBezierSpline`)

### Related Topics:

- *NDS – Compiler Interoperability Specification*, 5.15 Advanced Driver Assistance (ADAS) on page 98

### 9.6.1 Attributes for ADAS

Attributes for Advanced Driver Assistance (ADAS) are collected in a separate ADAS attribute layer. ADAS attributes are assigned to base links, road geometry lines, and transitions on level 13 and are related to road geometry.

As attribute changes can occur at locations where the road feature has no shape point, ADAS attributes additionally use attribute points for the spatial description.

For more information on attribute layers and attribute points, refer to Section 6.2 *Attribute Layers* on page 82.

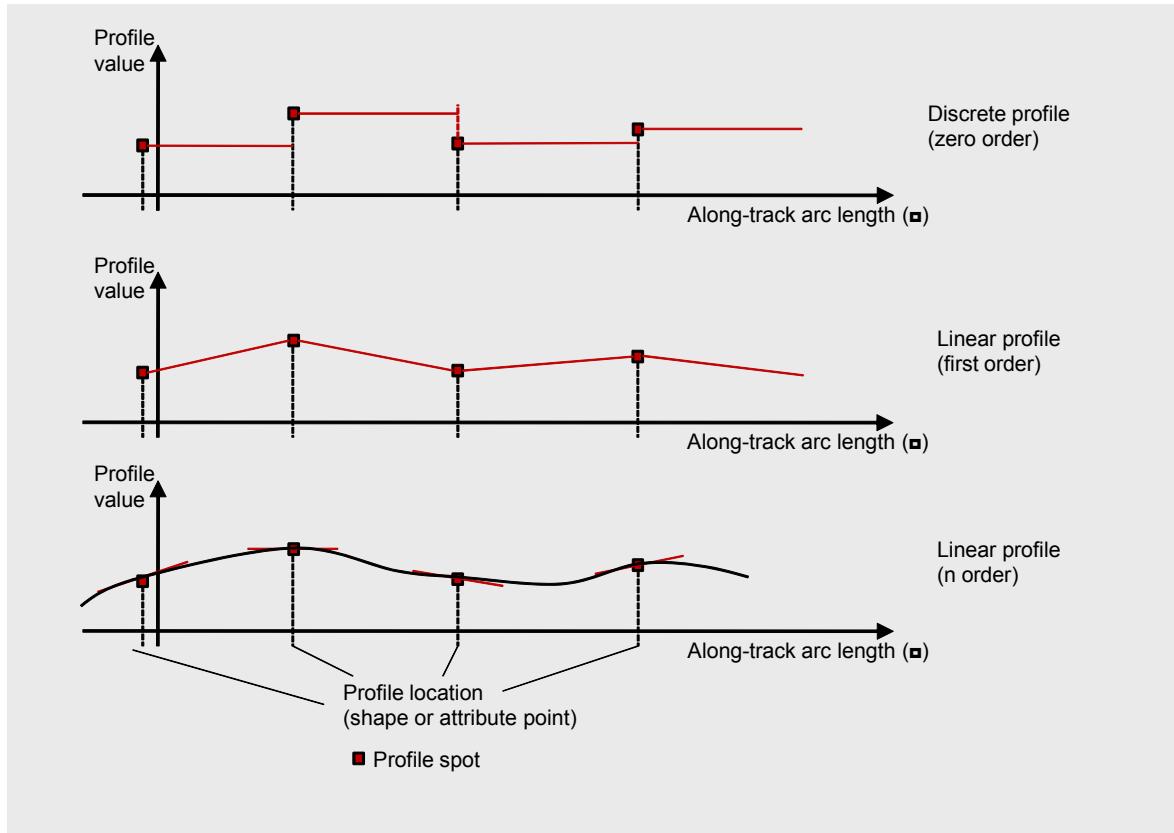
ADAS attributes contain information about detailed geometry. Currently, the following attribute types are available (DataScript location: `nds.common.flexibleattributes >`):

- **CURVATURE:** Curvature values can be stored as attributes attached to shape or attribute points of road features with linear interpolation between the values. To minimize quantization errors, the curvature is encoded as in ADASIS. For more information, refer to the description of the CURVATURE attribute in DataScript.
- **SLOPE:** The slope is described by a sequence of points for which a slope value is given.
- **PHYSICAL\_WIDTH\_METRIC** and **PHYSICAL\_WIDTH\_IMPERIAL**

Curvature and slope use the linear profile representation (see Figure 9-9). The ADASIS Standardization Forum also uses the linear profile representation for profile information on map-based track previews.

The linear profile representation calculates the profile value for coordinates ( $s$ ) along-track by interpolation of two neighboring profile spots. This is depicted in Figure 9-9.

*Figure 9-9 Different kinds of interpolation for profiles (profile view)*



Further ADAS attributes may be added in future versions of NDS. These attributes may use other kinds of interpolation. Attributes defining speed limits or number of lanes, for example, use the discrete profile (zero order) which calculates a constant value between the points identifying step changes.

---

<b>Note</b>	It is also possible to use attributes from other attribute layers for ADAS, such as warning signs and speed limits.
-------------	---

---

## Curvature Profile

The along-track curvature profile is represented by curvature values attached to shape points or attribute points along road geometry (so-called profile spots). The curvature value between two profile spots is calculated by linear interpolation. This results in a geometry similar to cubic spline or clothoid representation which ensures a smooth curvature representation. This representation supports most ADAS applications using curvature as a limiting factor for speed (for example, curve speed warning, curve ACC (adaptive cruise control)), or assistance functions based on vehicle-relative concepts (for example, adaptive curve light, lateral steering support, ACC target loss prediction).

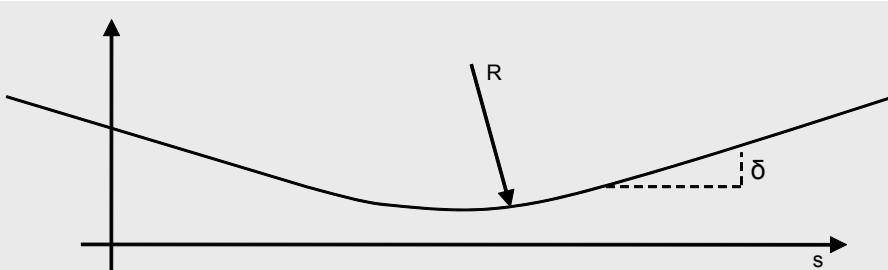
Curves constructed from this profile may, however, differ to some degree from the shape point geometry. ADAS applications relying on an absolute ground truth of the track require enhanced ADAS geometry information (see Section 9.6.2 *Enhanced ADAS Geometry* on page 137).

## Slope Profile

Slope (or grade line) profiles are represented by linear interpolated profiles for local slope or grade values. For this representation profile, the transition arcs are approximated by parabolas: The reciprocal  $c$  of the transition arc radius  $R$  reduces to the 'vertical curvature' or change  $d\delta$  of grade by distance  $s$ :

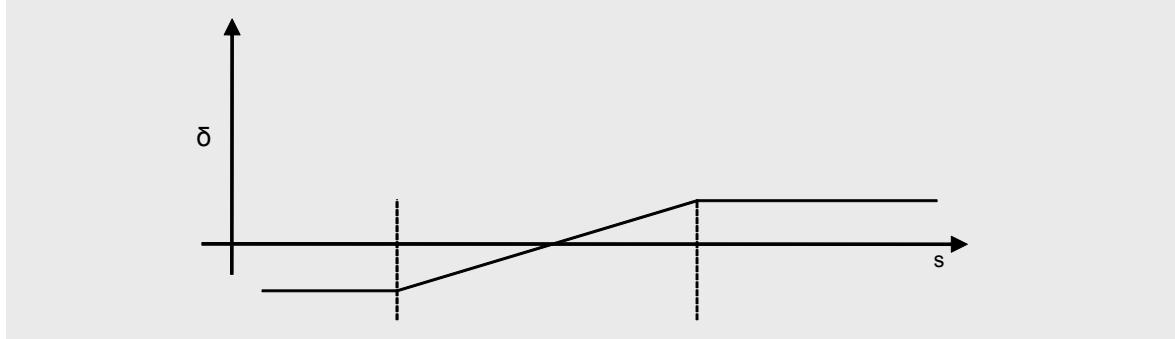
$$\frac{1}{R} = c = \frac{d\delta}{ds}$$

Figure 9-10 Road elevation profile (grade line) given by a piecewise linear grade profile (profile view)



If absolute elevation values are not relevant, the road elevation profile in Figure 9-10 can be represented by the equivalent road slope or grade profile along-track. As transition arcs represent a constant change of slope over track length  $s$ , they can be well approximated by a piecewise linear profile as shown in Figure 9-11.

Figure 9-11 Piecewise linear profile corresponding to the grade line in Figure 6-8 (profile view)



## 9.6.2 Enhanced ADAS Geometry

ADAS functions can use the line geometry information stored in NDS databases, combined with ADAS attributes as well as other attributes supplied in NDS. In addition to that, however, refined geometry information for ADAS may be supplied by means of clothoids (see <http://www.wikipedia.org/wiki/clothoid>). This geometry information is optional and makes it possible to describe the road geometry as a continuous and piecewise smooth curve.

### Clothoids

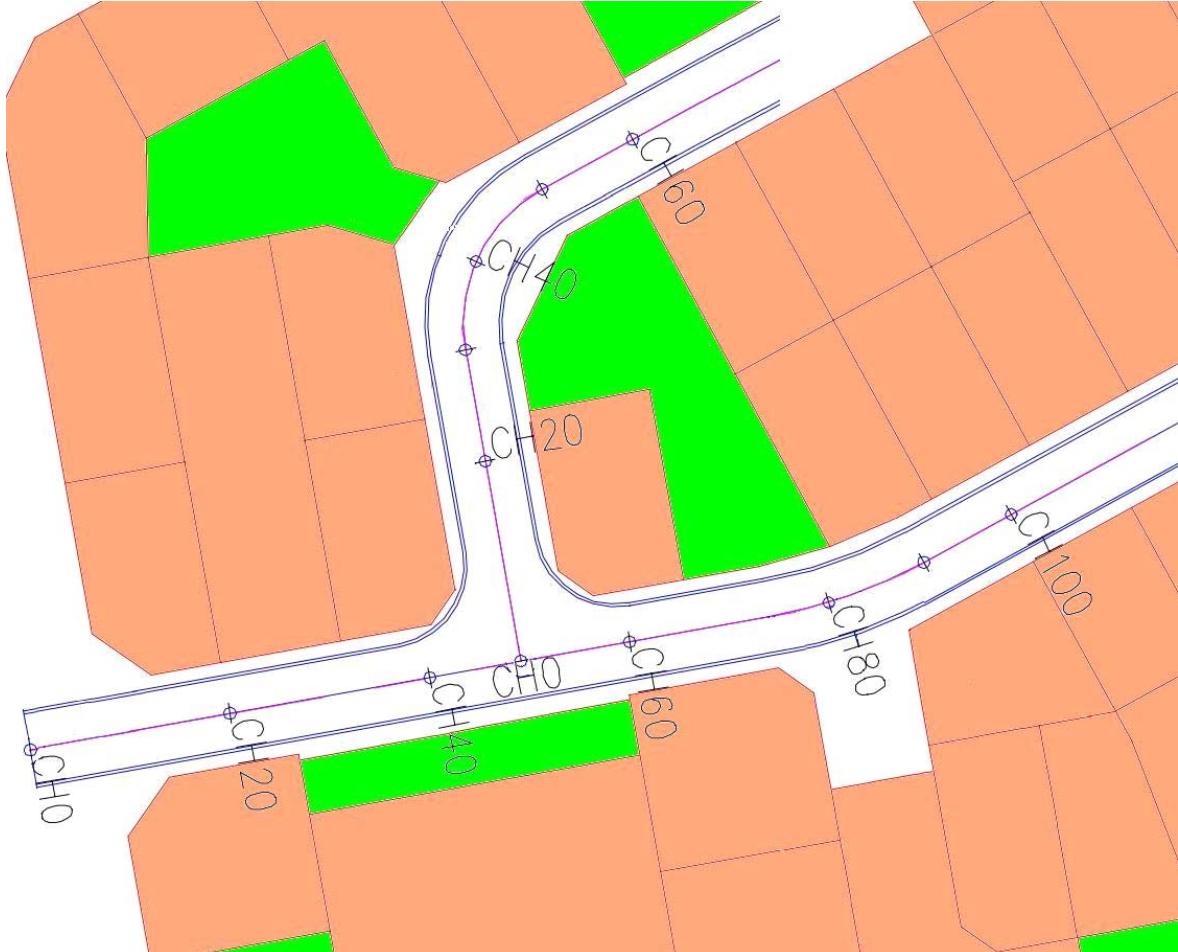
NDS defines a clothoid segment by means of the coordinates of its start and end point, and the start and end tangent. The tangent arc is defined as the clockwise arc between the y-axis (meaning north direction of WGS 84) and the tangent direction. This angle is called ellipsoidal azimuth.

<b>Note</b>	<p>The clothoid is described by points and tangents and not by curvatures, because the parameters have a different characteristic:</p> <p>For clothoid encoding by means of curvatures, the different resolution requirements for low and high curvature values cause negative effects, because a non-linear encoding schema is required. This does not apply for clothoid encoding by means of tangent angles and chord length. Hence, clothoids are described in NDS by tangent angles of <math>0^\circ - 360^\circ</math> and the chord length, which is defined by start and end point.</p> <p>Clothoid reconstruction from tangents causes a high computational effort, and NDS therefore recommends to reconstruct clothoids by means of lookup tables, which store the clothoid parameters as a function of the tangents and the distance of start and end point.</p>
-------------	--

For smoothly connected clothoids, the end tangent of the current clothoid, and the start tangent of the following clothoid are identical. The value for the end tangent is, therefore, an optional value which is only needed for modelling corners between consecutive clothoids.

The arc length of the clothoid or the road center line is represented by the chainage parameter (see <http://en.wiktionary.org/wiki/chainage>). The parameter defines the chainage related to the start point of the clothoid and is used to localize the position of attributes along the clothoid. Thus, clothoid-related information is expressed by chainage. For a vehicle position related to chainage, the vehicle distance to every attribute position can be calculated by addition or subtraction.

Figure 9-12 Two road center lines with marked chainages (plan view)



### Relation between Clothoids and Shape Points

To describe road geometry for ADAS as a continuous and piecewise smooth curve, it is not sufficient to use shape points. Additional parameters are required for the specification of the following alignment elements:

- Clothoid
- Circular arc
- Line

The shape points are used to connect the generated clothoids to the road network. That means that clothoids always need existing shape points as a reference.

Road geometries have different accuracies: For high functional road classes, a high accuracy is achieved by additional shape points, whereas lower functional road classes have standard shape points. For the quality of road geometry, standard deviation and degrees of freedom are defined (DataScript location `nds.common.flexibleattributes`):

- `degreeOfFreedom` is calculated from the number of points  $n$  which are needed to calculate the geometry elements minus 1 ( $n-1$ ). For clothoids extending over more than one link,  $n$  contains the number of shape points for the corresponding links. The degree of freedom value is then assigned to all corresponding links.
- The `standardDeviation s` is calculated as follows:

$$s = \frac{\sqrt{\sum d_i^2}}{n-1}$$

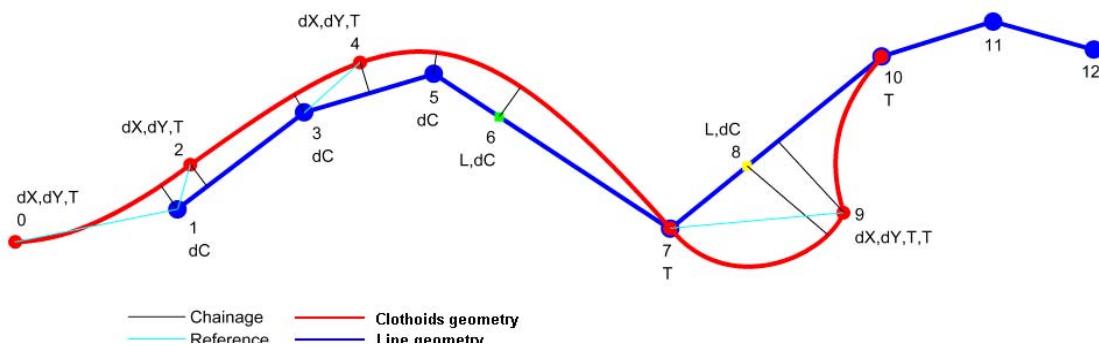
$d_i$  is the perpendicular distance of point  $P_i$  to the road center line ( $P_i$  being the starting point for clothoid calculation). A confidence value can be calculated by using the standard deviation, the degrees of freedom, and an individually defined significance level.

The standard deviation is assigned to all clothoids of a link.

### Offset between Clothoids and Shape Points

For more complex geometries, clothoids and shape point based geometry can differ, and additional data may be stored to describe the offset. This is depicted in Figure 9-13.

Figure 9-13 Clothoids and shape points with differing geometry (plan view)



The distance between clothoids and shape points is modeled by means of a coordinate offset to the related shape points. The attributes related to the clothoids are expressed by the chainage offset of the clothoid points from the start point of the clothoid. The chainage offset can be calculated in two ways:

- By using distance proportions; results in an approximate offset value
- By using a perpendicular; results in an exact offset value

**Note** Clothoid points always reference shape points and not attribute points.

An application reads the information along the shape and clothoid points and calculates the clothoid from this information. Assuming a sequential data stream along the road, fitting the clothoid information from Figure 9-13 into the sequential shape point model results in Table 9-9.

As ADAS clothoid data is optional and separated from the shape points, only the following geometry information will be stored in the ADAS data table.

Table 9-9 Logical structure of the clothoid data (from Figure 6-18)

Steps	dd	dX	dY	Ts	Te	dC	Note
0	DTs	dXC <sub>0</sub>	dYC <sub>0</sub>	Ts <sub>0</sub>			Initial cursor at shape point 1
0	C					dC <sub>1</sub>	
0	DTs	dXC <sub>2</sub>	dYC <sub>2</sub>	Ts <sub>2</sub>			
1	C					dC <sub>3</sub>	Cursor at shape point 3
0	DTs	dXC <sub>4</sub>	dYC <sub>4</sub>	Ts <sub>4</sub>			
1	C					dC <sub>5</sub>	Cursor at shape point 5
1	C					dC <sub>6</sub>	Cursor at shape point 6
1	Ts			Ts <sub>7</sub>			Cursor at shape point 7
1	C					dC <sub>8</sub>	Cursor at shape point 8
0	DTsT e	dXC <sub>9</sub>	dYC <sub>9</sub>	Ts <sub>9</sub>	Te <sub>0</sub>		Clothoid point references P7, because P8 is an attribute point
1	Ts			Ts <sub>10</sub>			Cursor at shape point 10

#### Table legend:

- Step: Describes the increment of an imaginary cursor within the shape points. The current shape point index results from the total of the preceding steps. All following clothoid data refers to the shape point at the current cursor position.
- dd = ADAS data present flag with
  - D = Coordinate difference dX, dY
  - Ts = Start tangent
  - Te = End tangent (only needed if Ts not equal to Te)
  - C = Chainage difference

In Table 9-9 the data from lines 1 to 3 are related to the first shape point, data from line 4 and 5 are related to the second shape point (#3), data from line 6 is related to the third shape point (#5), and so on. Thus, all clothoid data is related to the current shape point until steps is greater than 0.

To reconstruct the clothoid geometry, both the ADAS clothoid data from Table 9-9 and the shape point coordinates are needed. Combining both data results in the set of data described in Table 9-10. Here, X and Y are the shape point coordinates; the other values are the ADAS data from Table 9-9. The data from Table 9-10 also corresponds to the example from Figure 9-13.

*Table 9-10 Set of data obtained by combining clothoid data and shape point geometry (from Figure 6-11)*

X	Y	dX	dY	Ts	Te	dC	Note
X <sub>1</sub>	Y <sub>1</sub>						Start with first shape point
		dX <sub>c0</sub>	dY <sub>c0</sub>	Ts <sub>0</sub>		dC <sub>1</sub>	Offset to current shape point and Ts
							Related to current clothoid
		dX <sub>c2</sub>	dY <sub>c2</sub>	Ts <sub>2</sub>			Finishing first clothoid, beginning second
X <sub>3</sub>	Y <sub>3</sub>						Next shape point
						dC <sub>3</sub>	Related to current (second) clothoid
		dX <sub>c4</sub>	dY <sub>c4</sub>	Ts <sub>4</sub>			Finishing second clothoid, beginning third
X <sub>5</sub>	Y <sub>5</sub>						
						dC <sub>5</sub>	
X <sub>6</sub>	Y <sub>6</sub>						
						dC <sub>6</sub>	
X <sub>7</sub>	Y <sub>7</sub>						
				Ts <sub>7</sub>			
X <sub>8</sub>	Y <sub>8</sub>						
						dC <sub>8</sub>	
		dX <sub>c9</sub>	dY <sub>c9</sub>	Ts <sub>9</sub>	Te <sub>9</sub>		Finishing fourth clothoid, beginning fifth - discontinuity
X <sub>10</sub>	Y <sub>10</sub>						
X <sub>11</sub>	Y <sub>11</sub>			Ts <sub>10</sub>			Finishing last clothoid
X <sub>12</sub>	Y <sub>12</sub>						

### Table legend

- X = X coordinate value (shape point)
- Y = Y coordinate value (shape point)
- dX = Delta for X coordinate
- dY = Delta for Y coordinate
- Ts = Start tangent

- $T_e$  = End tangent (only needed if  $T_s$  does not equal  $T_e$ )
- $dC$  = Chainage difference

## Tiles and Clipping and Clothoids

For shape point geometries extending over more than one tile, the NDS clipping mechanism is used. Clothoid geometry may deviate from the shape point geometry. Thus, with regard to pure geometry, clothoids may extend over different tiles than the shape point geometry, but are always assigned to the same tile as the corresponding shape points.

Hence, clothoid geometry located within a tile is clipped if the referenced shape point geometry extends over more than one tile.

*Figure 9-14 Example of shape points and related clothoid in different tiles (plan view)*

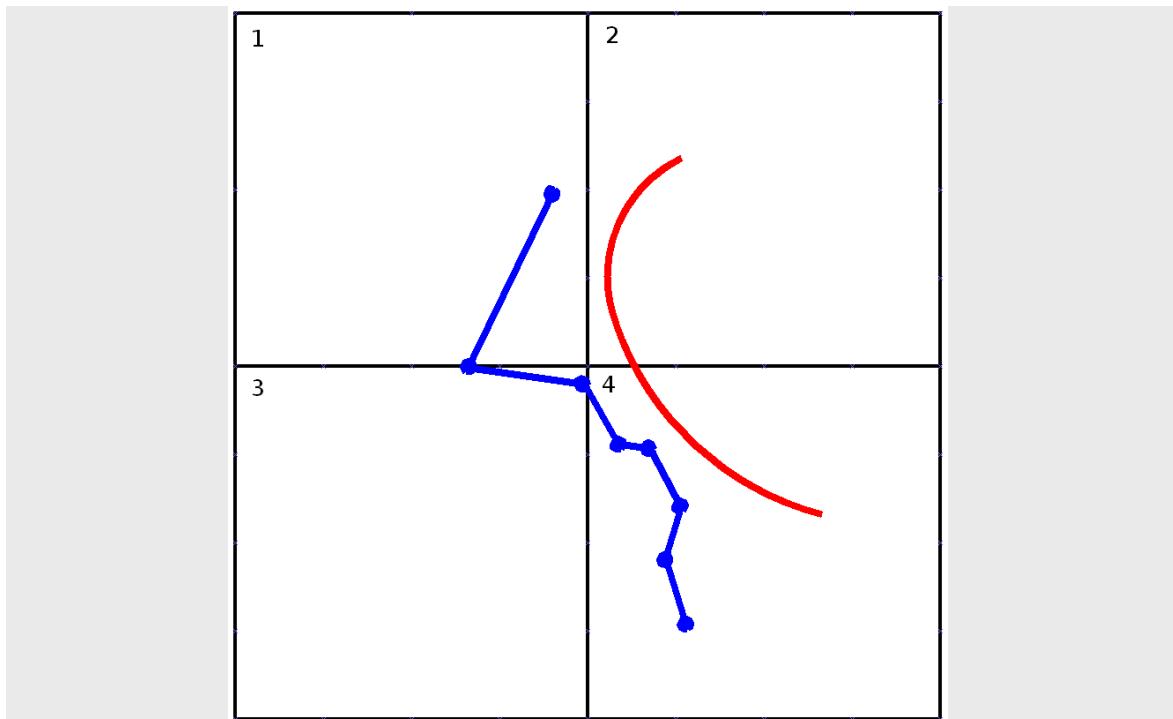


Figure 9-14 shows an example. The clothoid extends over tile 2 and tile 4, whereas the corresponding shape point geometry extends over tile 1, tile 3, and tile 4. In this case, the clothoid data belongs to tiles 1, 3, and 4, and not to tile 2 and tile 4.

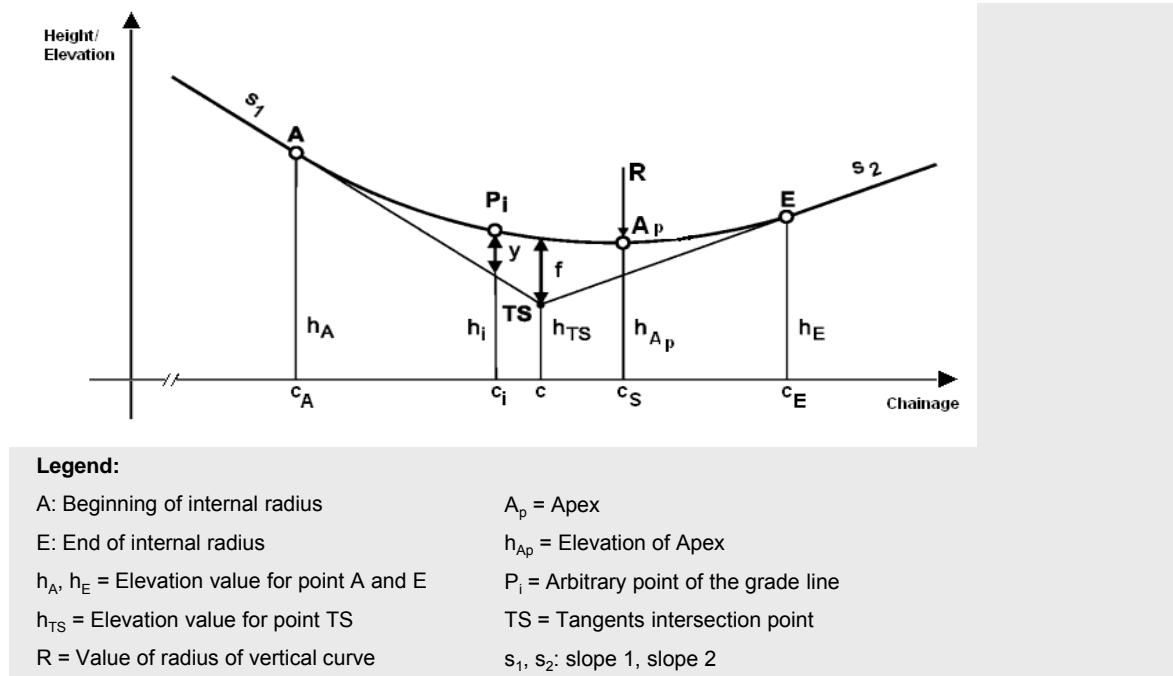
### 9.6.3 Enhanced ADAS Attributes for Elevation Profiles

In addition to the curvature and slope attribute, NDS provides attributes for modelling elevation profiles. Elevation profiles can be modeled as follows:

- Simple elevation profile with a slope attribute assigned to shape or attribute points
- Grade line elevation profile with straight lines connected by transition circle arcs (internal radii), meaning elevation and radius of vertical curve information assigned to shape or attribute points (DataScript location: nds > common > flexibleattributes > ExtendedGradient)

Figure 9-15 shows grade line tangents with an internal radius.

*Figure 9-15 Part of a grade line with a sag vertical curve (profile view)*



The elevation  $h_i$  at point  $P_i$  at station  $c_i$  is calculated as follows:

$$h_i = h_s + (c_s - c_i)/2R$$

$$h_s = h_A + (c_s - c_A)/2R$$

For more information on the formulas, for example, refer to Franz Josef Gruber, Rainer Joeckel *Formelsammlung für das Vermessungswesen*, Teubner B.G. GmbH; Auflage: 12 A. (August 2005).

The transition circle arc radius depends on the road class and, therefore, for example, on speed. To get a maximal line of sight, the radius is usually big, meaning more than 5 kilometers.

The database stores absolute elevations and, optionally, the radius of the transition circle arc. These values are mapped to the projected ground geometry. Table 9-11 gives an example.

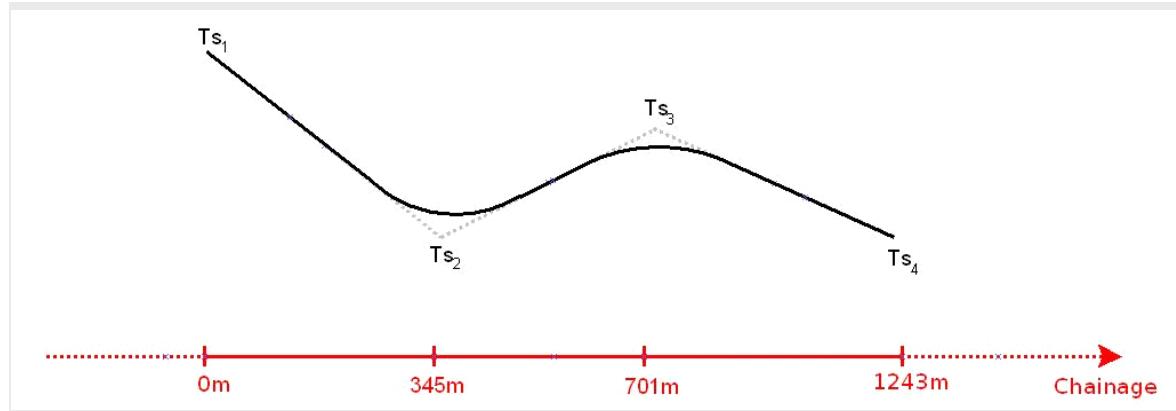
Table 9-11 Example of mapped values (elevation profiles)

Shape/attribute point index	Chainage [m]	Elevation [m]	Radius [m]
1	0	1553	-
2	345	1473	-3278
3	701	1517	8432
4	1243	1475	-

The algebraic sign for the radius indicates if the transition circle is a crest (+) or a sag (-). It is not necessary to store it in the database, because it can also be derived by the straight line elevation profile.

As ADAS geometry is fully linked with line geometry, elevation, slope, and grade line values can be mapped as well.

Figure 9-16 Grade line based on values from Table 6-10 (profile view)



The elevation and transition arc radii are stored as individual attributes, which are combined using attribute groups. Elevations can be used separately, but transition circle arc radii must be grouped with elevation.

### Clipping Grade Lines

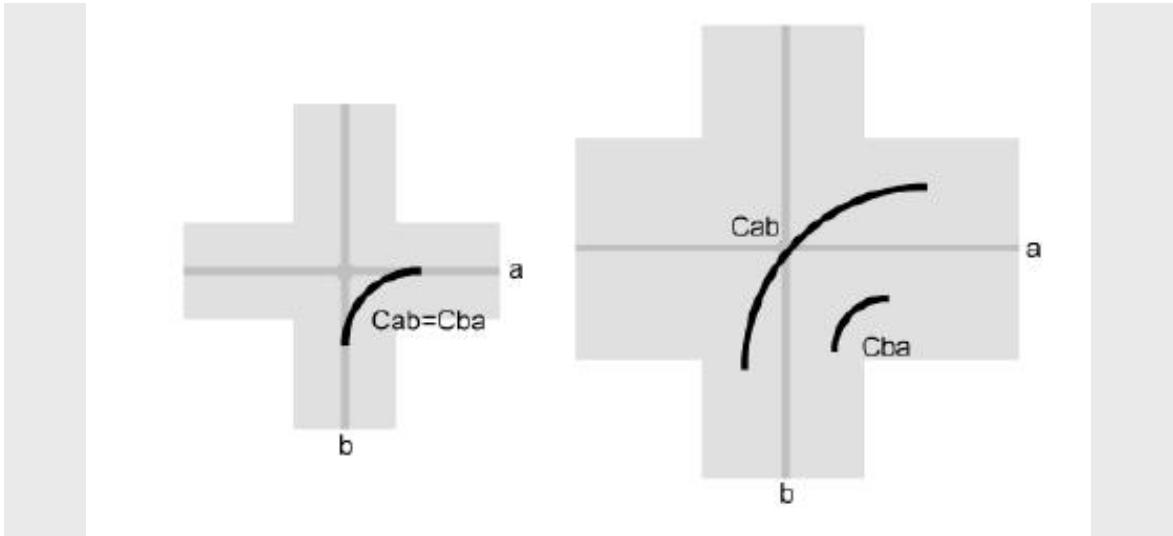
Even though a grade line references the clothoids chainage, it does not need the real clothoid geometry. Grade line clipping is, therefore, not necessary. All grade line data is stored in the same tile as the initial clothoid data and attached to the first link.

### 9.6.4 Enhanced ADAS Geometry Information for Intersection Features

Although intersections are point features, they can carry geometry information, such as directions, curvature, slope/grade line, superelevation, etc. This information usually depends on the actual transition between links over the intersection.

The different curvatures for left turns (link a -> link b) and right turns (link b -> link a) in an intersection are depicted in Figure 9-17. The respective curvature value may be delivered in the raw data, or can be calculated.

*Figure 9-17 Enhanced geometry information for intersection features*




---

<b>Note</b>	At intersections, the elevation differences can be neglected and an explicit description of the turn slope is therefore not needed. A linear interpolation is sufficient.
-------------	---

---

## 9.7 Generic Profile Attributes

Navigation Data Standard provides profile attributes to represent properties of features that change over time. Profile attributes can be associated to links and to transitions of intersections in the Routing building block.

### Speed Profiles

This section explains the concept of profile attributes by means of speed profiles. Speed profiles are already commercially available and are therefore the first use case for profile attributes in NDS. The NDS format, however, supports any type of profile attribute: It only requires to define a dedicated flexible attribute as done for speed profiles.

Speed profile information indicates how driving speeds vary over the course of a day for a given road section and travel direction. Each day can be divided into equal intervals. The length of an interval can be freely chosen, from a few minutes to several hours. For each of these periods as well as for each travel direction of a road section, an average speed is recorded in a speed profile. There is usually one speed profile for each day of a week. Additional speed profiles can be available, for example, for holidays.

A speed profile can be represented as follows:

- As an array of average speeds for time intervals of equal length (time array)
- As an array of the most relevant parameters of the discrete cosine transformation

Commercially available data for speed profiles often contains only a moderate number of speed profiles, each of which is assigned to many links or transitions. Therefore, it is sensible to factor out the speed profile data into a table and to store references into this table at links or transitions. The profile table exists only once per update region and is therefore stored in the region metadata (DataScript location: `nds.overall.metadata > RegionProfileTable`).

Speed profiles base on real experiences and therefore allow navigation systems to provide realistic time estimations for users of navigation systems. By selecting the most suitable route that avoids, for example, congested roads during rush hours, users can save time and reduce the fuel consumption.

### 9.7.1 Flexible Attributes for Speed Profiles

Speed profiles are modeled in NDS by means of the flexible attribute `SPEED_PROFILES_OR_AVG_SPEEDS_PER_WEEK`.

The attribute is applied to one or both directions of a link, or to a transition of an intersection. For every day of the week it contains either a reference to a speed profile, or an average value for the whole day. The attribute may be used as part of an attribute group assigned to a link in conjunction with the vehicle type.

For more information, refer to *NDS – Compiler Interoperability Specification*, 5.14 Generic Profile Attributes on page 97.

The flexible attribute `SPEED_PROFILES_OR_AVG_SPEEDS_PER_WEEK` includes all information relevant for speed profiles for a single link or transition for a given direction in a compact way, in order to save storage space (a link may have up to 14 profiles which are stored in 2 flexible attributes).

The bitmask `daysOfWeek` indicates the days for which a detailed profile is available. If at least one detailed profile is there, a `baseSpeed` is provided, also known as the free flow speed. For each weekday either a reference to a profile (`profileId`) or an `averageSpeed` is provided. The profile itself is stored in the metadata (see below). The profile defines percentage values of the `baseSpeed` (in km/h) that apply to the link over the specified time range in time intervals. The allowed percentage values are as follows: 0 – 327.67 with 0.01% increment.

The attribute is not applied to links without daily profiles or average speeds. In this case, the `averageSpeed` of the link (stored in the link data structure) is to be used.

## Profile Header Information

The profile header contains the following meta information:

- **Profile representation:** Enumeration describing how the profile is represented, for example, by a particular mathematical function, such as the Discrete Cosine Transformation coefficients (see Section 9.7.3 *Discrete Cosine Transformation* on page 149) or an array of discrete values over time.
- **Number of values:** For a particular mathematical function, the number of values describes the number of coefficients provided. For an array with percentages, the number of values equals the number of time intervals. (It is assumed that days always start at midnight.) The number also allows calculating the duration of the time interval in case of a time array representation by dividing the time range of the profile (for example, one day) by the number of values (for example, 24).
- **List of values:** The semantics for list of values depend on the profile type and the representation: There are, for example, speed profiles which are represented by arrays with percentage values. In this case, the list of values represents a list of percentages that indicate the percentage of the free flow speed in the middle of the given time interval (`baseSpeed` is a member element of the flexible attribute `SPEED_PROFILES_OR_AVG_SPEEDS_PER_WEEK`). The first value in the array corresponds to the first time interval. If a profile applies to the time span of a day, the first value in the array represents the value at 0:00:00, marking the middle of the time interval, and corresponds to the entry time into the link. If the link is entered at a given time, the profile value of the time interval containing this time must be used. (The left end point of the link belongs to the interval.)

### 9.7.2 Example

A one way link has the flexible attribute `SPEED_PROFILES_OR_AVG_SPEEDS_PER_WEEK` with the following values:

- `daysOfWeek: Su=1 Mo..Sa=0 I/E=1` (Sunday detailed profile, other days average speed, `isInclusive` flag set to TRUE)
- `baseSpeed: 50 km/h`
- `profileOrAvgSpeedSunday: 23` (profile ID)
- `profileOrAvgSpeedMonday: 55` (average speed, km/h)
- [... other days ...]
- `profileOrAvgSpeedSaturday: 65` (average speed, km/h)

The `profileId = 23` points to the profile table which is stored in the update region metadata. It indicates the following:

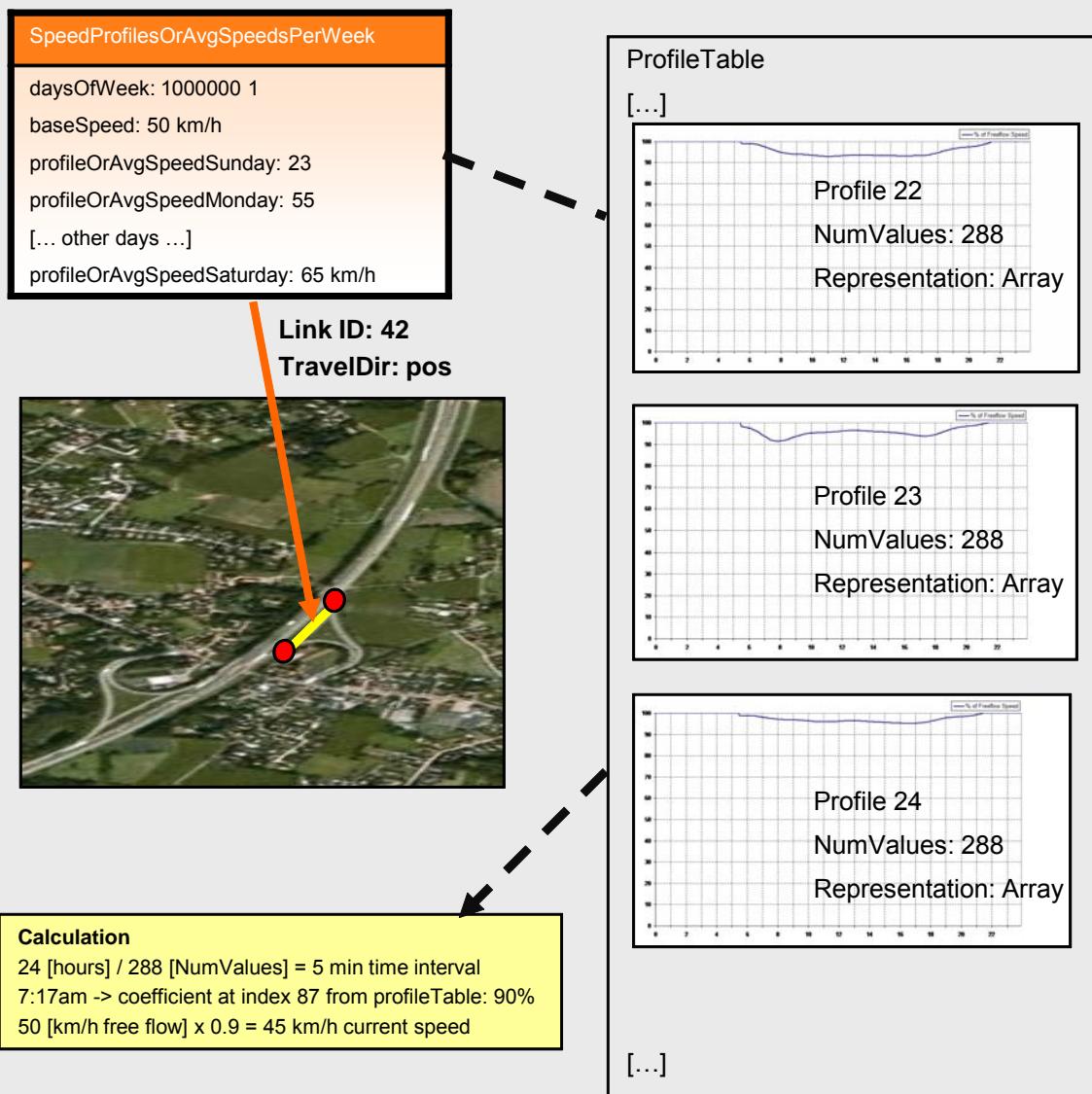
- Array as representation
- 288 values (intervals), indicating that there is a value for every five minutes of the day

Assuming a car is driven on a Sunday, and the current time is 7:17 am. This equals the 88th element in the array, meaning the array element with index 87. Looking up this element gives a coefficient equal to the speed percentage value of this link on the given day at 7:15 am, for example, 90%. This represents the percentage of speed along this link at the current time from the indicated `baseSpeed`, so it corresponds to 90% of 50 km/h → 45 km/h. This is the value the route cost function would use for the respective link at the respective time.

A more advanced use would be to interpolate between the percentage values for 7:15 am and 7:20 am.

The following figures illustrates this example.

Figure 9-18 Speed profile, example



### 9.7.3 Discrete Cosine Transformation

The discrete cosine transformation (DCT) is another way to represent profile attributes in a more compact way.

The discrete cosine transform is an invertible function  $F: \mathbf{R}^N \rightarrow \mathbf{R}^N$

The N-dimensional vector  $A = (x_0, \dots, x_{N-1})$  is transformed into  $B = (X_0, \dots, X_{N-1})$  by using the following function:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \quad k = 0, \dots, N-1.$$

The original vector can be obtained by applying the inverse DCT:

$$x_k = \frac{1}{N} X_0 + \sum_{n=1}^{N-1} \frac{2}{N} X_n \cos\left[\frac{\pi}{N}\left(k + \frac{1}{2}\right)n\right] \quad k = 0, \dots, N-1.$$

Since many profiles show a sinusoid form, a dimension reduction method based on sine/cosine basis functions is a well suited, high-quality size reduction method for profile attributes, such as speed profiles, in NDS. Instead of storing all N measurements, it is sufficient to store a much smaller number N' of DCT transformed coefficients. Using a function also makes the interpolation between two values redundant.

Figure 9-19 and Figure 9-20 depict comparisons of traffic patterns in array representations (96 values) and the DCT reconstructions (10 coefficients). NDS recommends to reduce the number of coefficients in order to keep the mean square error below a certain value. NDS does **not** recommend to define a fix number of coefficients.

Figure 9-19 Traffic patterns in array representations vs. DCT reconstructions, example 1

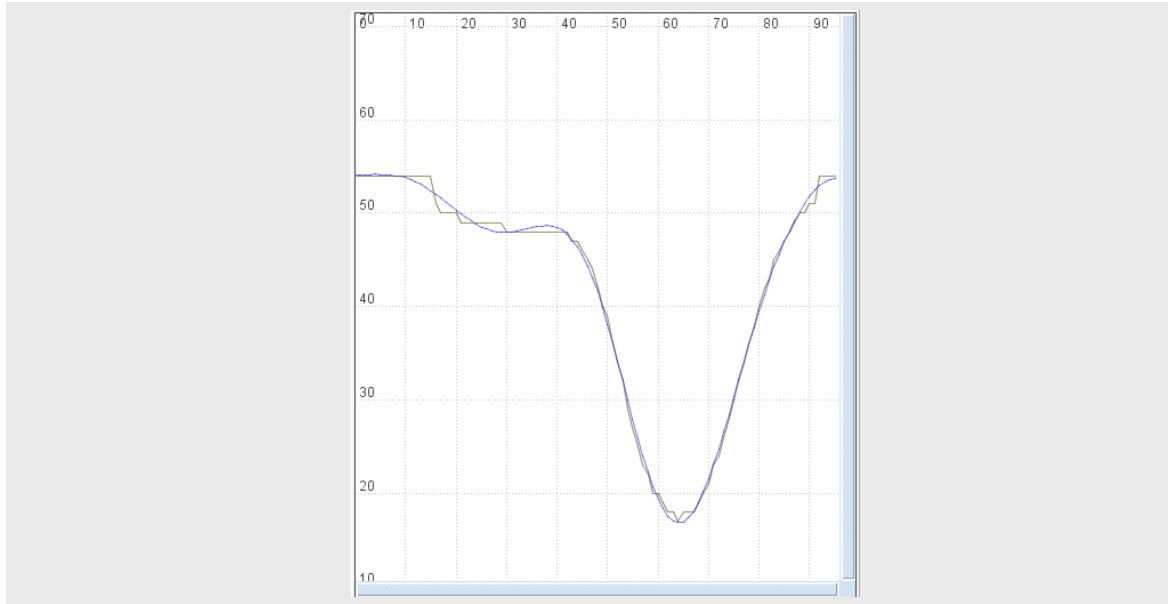
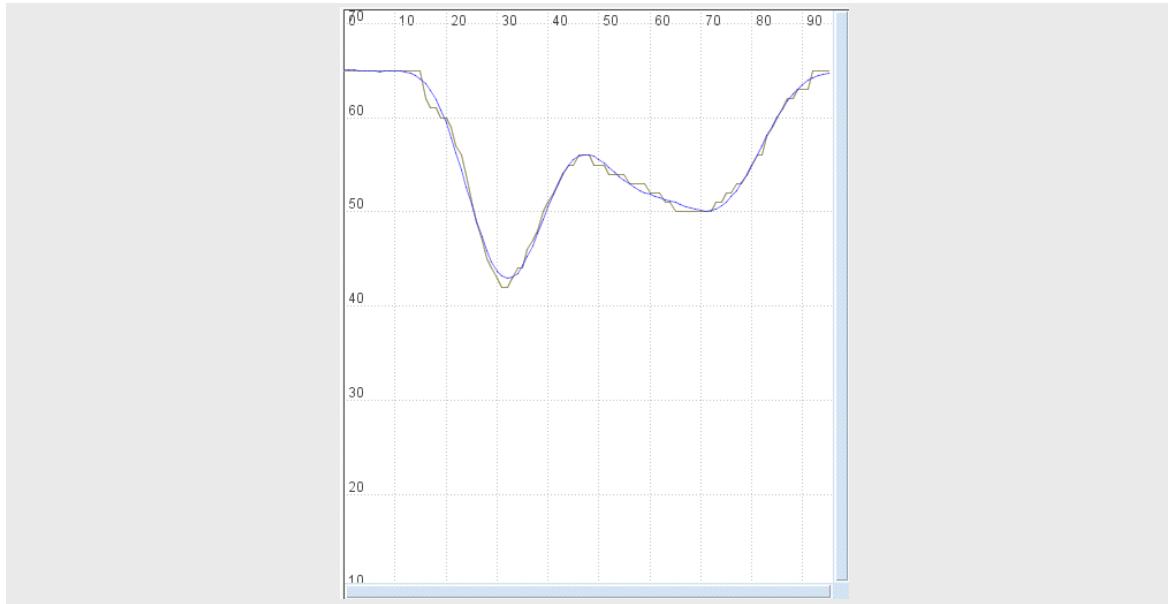


Figure 9-20 Traffic patterns in array representations vs. DCT reconstructions, example 2



The coefficients of the DCT are stored in the list of values of the pattern in the following order:

Index	0	1	2	3	...	$N'-1$
Value	$X_0$	$X_1$	$X_2$	$X_3$	...	$X_{N'-1}$

The coefficients reflecting the lowest frequencies are stored first.

For more information, refer to [http://en.wikipedia.org/wiki/Discrete\\_cosine\\_transform](http://en.wikipedia.org/wiki/Discrete_cosine_transform).

## 9.8 Address Points

Address points are used to model the correct location of house numbers on the map as well as the correct access point for routing to a specific address. Address points are assigned to house number ranges and are stored in the routing geo tile (DataScript location: `nds.routing.main > RoutingAuxTileTable > nameAttributeMapList`).

Although address points are stored in the `RoutingAuxTileTable`, they logically belong to the Name building block.

For more information on address points, refer to Section 10.9 *Address Points* on page 199.

## 9.9 Data Structures for Eco Routing

In addition to the shortest distance and shortest duration routes, *eco routing* offers another option for route calculation. This option is both, economic and eco-friendly in the sense that it minimizes fuel consumption and thereby also emissions.

---

<b>Note</b>	This topic explains the basic concepts of eco routing and describes the NDS data structures containing information for calculating eco routes.  For all information on how to calculate eco routes based on this data in the compiler and in the application, refer to <i>NDS – Compiler Interoperability Specification, 5 Routing Building Block</i> on page 79.
-------------	---

---

The following infrastructural factors influence the calculation of eco routes and have to be taken into account for eco routing in NDS:

- Consumption speed curve (CSC)

The CSC describes the speed-dependent fuel or energy consumption for a car driving at a constant height and at a distinct average speed. The CSC takes normal traffic speed fluctuations into consideration. Specific consumption effects, however, which could also be derived from a map, are modeled separately by means of the speed variation and slope factors.

- Speed variation

Transitions, intersections, and curves on a route cause speed variations and thus impact consumption. The speed variation factor for transitions describes how acceleration and deceleration affect consumption due to road class transitions, speed limits, right of way regulations, and sharp bends at nodes and intersections. The speed variation factor for curves describes acceleration and deceleration effects on consumption due to speed reduction at curves along links.

- Slope

Road slope above a specific threshold causes excessive consumption due to a higher energy effort during uphill driving and an energy waste due to braking when driving downhill.

NDS provides structures to store data for these infrastructural factors in the database, which can then be used in the calculation. Flexible attributes, which can be assigned to links on all levels, carry consumption information for excess slope and speed variation.

---

**Caution**

The flexible consumption attributes in an NDS database are based on the road infrastructure, and are, thus, universally valid for all vehicles. Vehicle- and driver-specific properties can be added at runtime and multiplied with the infrastructure-related consumption factors from the database. The Compiler Interoperability Specification describes how the compiler and/or the application use this information to calculate the routes, and also how additional, vehicle-specific information can be added to get more precise results, see *NDS – Compiler Interoperability Specification, 6 Eco Routing* on page 101.

The following sections give an overview on the basic concepts of eco routing and the data structures that are available in NDS to store the required information for eco route calculation.

## Eco Routing Factors and Calculation – Overview

Figure 9-21 gives an overview of the energy consumption factors used for eco route calculation and shows how these consumption factors can be handled on aggregated levels.

---

**Note**

The table at this point only provides an overview; the parameters and the derivation of the equations are explained in detail in the Compiler Interoperability Specification, see *NDS – Compiler Interoperability Specification, 6 Eco Routing* on page 101.

Figure 9-21 Fuel/energy consumption effects - Overview

Factor	Effect	Physics	Map attributes	Parameter	Eco attribute for level 13	Eco attribute for aggregated links
$c_{speed}$	- Air and roll resistance - Engine loss - Drive train loss		<ul style="list-style-type: none"> <li>• Speed category</li> <li>• Speed patterns</li> <li>• Real-time traffic speed</li> </ul>	CSC (vehicle dependent ... ) consumption speed dependency curve	Calculated at runtime from link attributes	$c_{speed}$ Grouped attribute: Speed classes by percentage of link
$c_{transition}$	Energy loss caused by - Breaking - Accelerating		<ul style="list-style-type: none"> <li>• Change of speed category</li> <li>• Change of speed limit</li> <li>• Intersection type</li> <li>• Traffic lights</li> <li>• Stop signs</li> <li>• Right of way</li> <li>• Manoeuvre angles</li> </ul>	Vehicle mass (m) Acceleration efficiency ( $k_+$ ) Recovery efficiency ( $k_-$ ) Probability of effect (p)	Calculated at runtime from link and intersection-attributes	$c_{transit+}$ $c_{transit-}$  $c_{var+}$ $c_{var-}$
$c_{curve}$	Energy loss caused by - Breaking - Accelerating in curves		<ul style="list-style-type: none"> <li>• Curvature</li> <li>• Shape</li> </ul>	Curve speed limit factor ( $v_{lim} = f(R)$ ) Vehicle mass (m) Acceleration efficiency ( $k_+$ ) Recovery efficiency ( $k_-$ )	Calculated at compile time from ADAS curvature or shape	$c_{curve+}$ $c_{curve-}$
$c_{slope}$	Energy loss caused by - Excess climbing - Breaking at steep slopes		<ul style="list-style-type: none"> <li>• Height / Δheight</li> <li>• Slope ρ</li> </ul>	Excess slope threshold (ρ) Vehicle mass (m) Climbing efficiency ( $k_+$ ) Recovery efficiency ( $k_-$ )	...calc. at compile time from ADAS slope or DTM data	$c_{slope+}$ $c_{slope-}$

### 9.9.1 Eco Routing Metadata

Table 9-12 shows the eco routing metadata in an NDS database.

DataScript location: `nds.overall.metadata > RoutingMetadata`

Table 9-12 Eco routing metadata

Metadata	Description
<code>hasEcoRoutingAttributes</code>	Indicates the availability of eco routing data in the database
<code>upExessSlopeThreshold</code>	Defines the threshold used to calculate the excess slope (uphill) if <code>hasEcoRoutingAttributes</code> is TRUE
<code>downExessSlopeThreshold</code>	Defines the threshold used to calculate the excess slope (downhill) if <code>hasEcoRoutingAttributes</code> is TRUE

### 9.9.2 Consumption Speed Curve

The consumption speed curve (CSC) describes the dependency of energy (fuel) consumption and the (average) driving speed of a vehicle ( $v$ ): The speed-related consumption factor  $c_{speed}$  is based upon this vehicle-specific CSC function, using the estimated speed data from the map links, such as estimated average speed and speed limits:

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

Figure 9-22 shows a typical example of a CSC for combustion engine vehicles (approximations base on published fleet data; source: *Emission Factors from the Model PHEM for the HBEFA Version 3; Report I-20/2009; Graz University of Technology*).

*Figure 9-22 Typical CSC representing a plurality of vehicles*

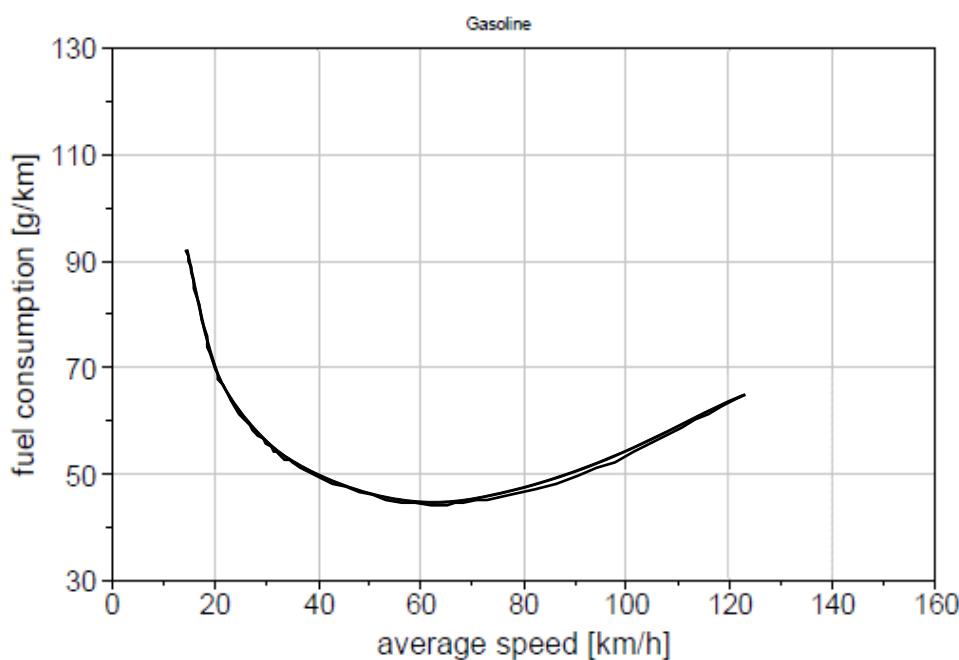


Figure 9-22 shows an example for a CSC representing average data over a whole class of vehicle models. The CSC used for eco routing calculation is specific to the individual vehicle model, since it summarizes the joint effects of aerodynamic resistance, roll resistance, and engine and transmission efficiency for the respective vehicle at different speed levels.

A simple lookup of the vehicle-specific CSC is sufficient to extract the expected consumption value for a given average speed of a link. For links on upper routing levels, however, it is not sufficient to use the overall average speed for a CSC lookup. Due to the extremely nonlinear form of the CSC, the fractional speed contributions from the lowest level must be propagated up to the aggregated link. For this, the NDS-specific feature of aggregated attributes is used (see Section 9.9.2 *Consumption Speed Curve* on page 154).

For more information, refer to *NDS – Compiler Interoperability Specification, 6.2 Speed Factor cspeed and Consumption Speed Curve (CSC)* on page 104.

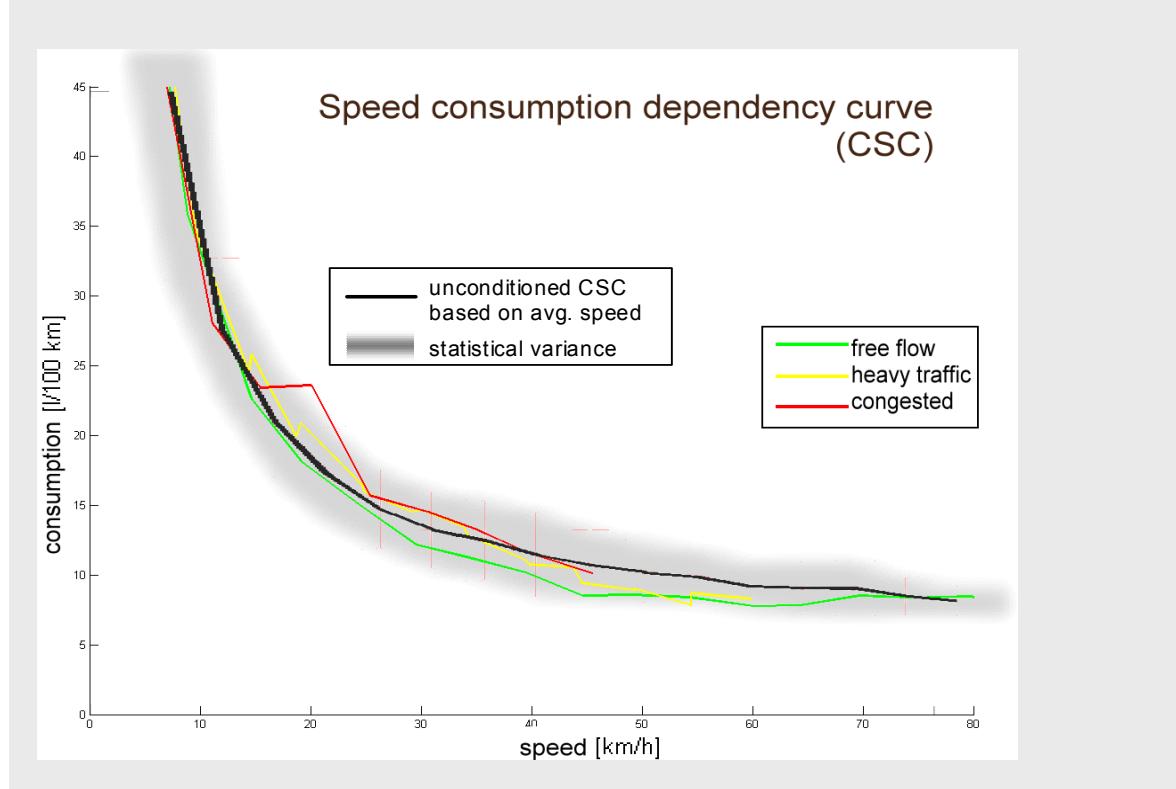
### Influences on Speed-dependant Fuel Consumption

Driving tests with normal driving behavior have shown that the influence of average speed on fuel consumption is almost independent of the conditions that cause the average speed:

- A high average speed results from low speed variation levels, which is also reflected by the moderate consumption increase observed at higher speeds (see Figure 9-23).
- A low average speed usually results from external influences, which force the driver to reduce his driving speed and may also cause occasional stops. A low average speed is the consequence of intermittent driving, as encountered in urban environments or congested traffic.

Figure 9-23 shows the statistical CSC (black curve; result of 600 kilometers driving under varying conditions) and statistical variance (gray shading). The colored curves show the measured speeds and consumption based on traffic conditions as judged by the driver. The figure shows that the traffic condition does not influence the consumption directly. According to this figure, the driven average speed is a decisive parameter that determines consumption regardless of the driving condition.

Figure 9-23 Statistical CSC and variances



As a consequence, NDS uses the CSC for modelling speed influences on the consumption without taking the source of different speed values into account. The following map data can, therefore, be used as a source for speed-related consumption estimations in NDS:

- Static speed categories or speed limits
- Speed profiles (by hour of day/weekday)
- Speed data from real time traffic feeds

### Data Structures for CSC

NDS models the consumption speed curve by means of the flexible attribute `CONSUMPTION_SPEED_DEPENDENCY`. The attribute stores an array of consumption speed values (`ConsumptionSpeedDependencyCurve`). Each consumption speed value (`ConsumptionSpeedValue`) is a tuple of an average speed value (`averageSpeed`) and the percentage of the link, to which the average speed applies (`LINK_PERCENTAGE`). The link percentage factor acts as a weighing factor for the CSC calculation.

DataScript location: `nds.common.flexibleattributes`

On level 13, each link has a dedicated average speed value assigned. The application can use this value for table lookup and to calculate the consumption value at runtime. For aggregated links on higher routing levels, the CSC is represented by a maximum of 256 value pairs of average speed values and their corresponding link percentage factors.

On the lowest routing level, the average speed attribute is assigned to each link and is valid for the link's entire length. The routing algorithm uses this average speed value at runtime for the consumption lookup.

On upper routing levels, aggregated links with different average speeds are represented by a maximum of  $n = 256$  value pairs of average speed values and their corresponding weighing factors.

For more information on the calculation and aggregation of CSC, see *NDS – Compiler Interoperability Specification, 6.2 Speed Factor cspeed and Consumption Speed Curve (CSC)* on page 104.

### 9.9.3 Speed Variation

*Speed variation* affects fuel consumption due to changes in kinetic energy, especially caused by braking and accelerating at curves, transitions, and intersections. The kinetic energy of an object of a mass  $m$  moving with a speed  $v$  is defined by the following formula:

$$e_{kin} = \frac{m}{2} v^2$$

The object gains this energy during acceleration and maintains the energy unless its speed changes. Following the kinetic energy definition above, speed changes cause a change in kinetic energy, which is proportional to the difference of the speeds squared.

The following factors influence consumption by causing speed variation and are taken into account for eco routing:

- Transitions

The consumption factor for transitions ( $c_{transition}$ ) is used to model effects caused by transitions between links, such as:

- Changed explicit attribution of speed from one link to the next link
- Speed reduction due to a change in direction from one link to the next (maneuver)
- Speed reduction due to a right of way rule at an intersection (depends only on entry link) by explicit attribution (for example, traffic lights, right of way) or implicit attribution (for example, priority model derived from road classes)

- Curves

The consumption factor for curves ( $c_{curve}$ ) is used to model effects caused by curves located along links: Before entering a curve, a driver reduces speed and accelerates again when leaving the curve.

The two factors are combined into a normalized common speed variation factor:  $C_{var}$ .

For more information on how to calculate these values, refer to *NDS – Compiler Interoperability Specification, 6.3 Speed Variation* on page 112.

### Data Structures for Speed Variation Factor

The speed variation factor is calculated at compile time based on the following information:

- The change of speed from one link to the next
- The speed drop due to direction changes
- The speed drop due to right of way rules such as traffic lights
- Deliberate speed reduction before curves

To reflect the energy dependency as stated above, the speed variation is measured in terms of squared speed differences. For practical reasons, this difference is not stored directly; the value is normalized per link length and scaled by 128.

To store the calculated consumption factor for speed variation, NDS provides the flexible attribute **CONSUMPTION\_SPEED\_VARIATION**. The flexible attribute may be assigned to links at intersections, transitions, or sharp turns.

DataScript location: `nds.common.flexibleattributes`

The calculation of the speed variation factor is described in *NDS – Compiler Interoperability Specification, 6.3 Speed Variation* on page 112.

---

**Example**      Original squared speed variation:  $(80^2 - 50^2) \text{ km}^2/\text{h}^2 = 3900 \text{ km}^2/\text{h}^2 \sim 301 \text{ m}^2/\text{s}^2$

Link length: 20,000 cm = 200 m

Stored value for speed valuation:  $301 / 200 * 128 = 193 \text{ m}^2/\text{s}^2$

---

The speed variation factors for transitions belong to an intersection and involve the link and its relations (bending angles, previous and consecutive speed levels, etc.). For this reason, the flexible attribute is not used on lower Routing levels. The application uses the link and intersection attributes to calculate the consumption values at runtime. The routing algorithm derives the consumption values for lower-level links from the calculated values.

For aggregated links on higher levels, the energy effects due to acceleration and deceleration are collected and summed up for the aggregated link. The resulting speed variation is stored in the flexible attribute **CONSUMPTION\_SPEED\_VARIATION**. For more information, refer to *NDS – Compiler Interoperability Specification, 6.3 Speed Variation* on page 112.

#### 9.9.4 Slope

In the same way as the kinetic energy, the changes of potential energy influence consumption. The potential energy of an object of a mass  $m$  depends directly on its vertical (=height) coordinate  $h$ , as defined in the following formula:

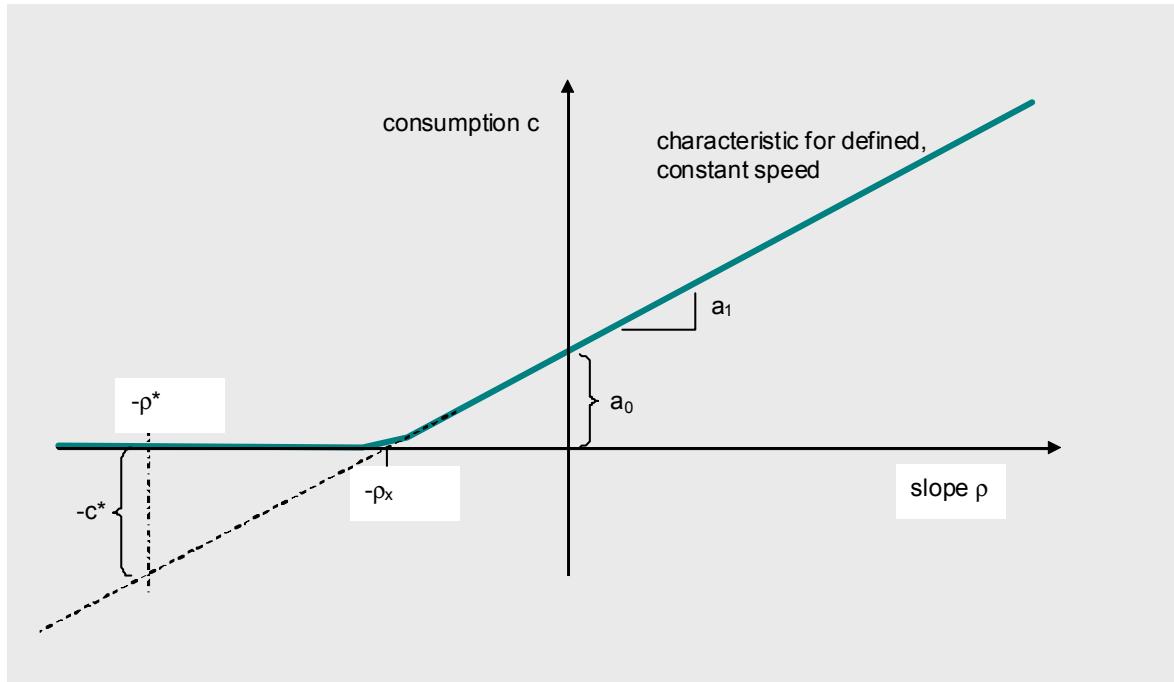
$$e_{pot} = m \cdot g \cdot h$$

Changing the height  $h$  by driving ascents or descents increases or reduces the potential energy correspondingly. Consequently, the consumption factor for slope ( $c_{slope}$ ) must consider the following two effects:

- Ascending a slope costs additional energy by increasing the potential energy.
- During descent, the existing potential energy may either be wasted by braking (including engine braking), or by driving without using motor energy (idling). That means the consumption figures may be very low or even negative, reflecting eventual energy recovery for electrical drives.

The typical consumption behavior of combustion and electric engines shows a close linear dependency between slope and consumption over a large operating range, that is, roughly within the range of  $\pm 4\%$ .

Figure 9-24 Simplified curve for dependency between slope and consumption for combustion engine vehicles



Three considerations result from this fact:

- As the eco routing function compares energy consumption for route alternatives to the same destination, the height difference between start and destination are equal for all route alternatives. That is, also the difference of potential energy from start to destination is equal for all routes.
- If a route has only moderate up and down slopes that are about balanced, the extra energy effort along the ascending slopes is typically compensated by a reduced energy consumption in corresponding downhill phases.
- Especially with extreme down slopes below  $-4\%$ , part of the potential energy cannot be recovered, that is, the energy is wasted due to braking in order to keep a constant speed.

That leads to the conclusion that the consumption difference between alternative routes caused by uphill and downhill driving is primarily determined by segments of excessive slope. For eco routing it is therefore not necessary to consider slope values below a defined magnitude. Thus, the eco routing attribute for storing consumption information for excess slope only need to be assigned to links with excess slope magnitudes. This applies to about 8% of the links in normal terrain and up to 30% in mountain regions. For more information, refer to *NDS – Compiler Interoperability Specification, 6.4 Slope* on page 125.

Range calculation applications must estimate the absolute consumption for every point along the route. To do this, the slope average value is required for all links of the road network. The **AVERAGE\_SLOPE** attribute is designed for this as it allows applications to estimate the actual consumption and the remaining energy for every position along the route.

## Data Structures for Slope

A slope is defined as the tangent of the angle of ascent, usually expressed by a dimensionless percentage value.

The flexible attribute `AVERAGE_SLOPE` stores the value for the average slope between the start and the end node of a link. The value is calculated as the height difference between the start and end node, divided by the total link length (according to the tangent definition). The average slope attribute is essential for range estimation purposes.

DataScript location `nds.common.flexibleattributes`

For eco routing applications, NDS provides the attribute type `excessSlope` that defines the value, by which the up and down slope exceeds a defined threshold. Routing applications use this attribute type to decide about the lowest consumption route alternative to a given destination (see *NDS – Compiler Interoperability Specification*, 6.4.3 *Excess Slope Attribute* on page 131).

To store the consumption factors for excess slope, NDS provides the flexible attributes `CONSUMPTION_UP_EXCESS_SLOPE` and `CONSUMPTION_DOWN_EXCESS_SLOPE` (DataScript location: `nds.common.flexibleattributes`). In addition to the excess slope value itself, the metadata contains the slope threshold that has been used for the excess slope calculation at compile time (`upExessSlopeThreshold` and `downExessSlopeThreshold`, see Table 9-12).



## 10 Name Building Block

The Name building block contains the name data for the Routing and Basic Map Display building blocks. Instead of storing their own names, these building blocks make use of the name features centrally stored and managed in the Name building block. Names can thus be shared between the Routing and Basic Map Display building blocks and can be maintained separately in the Name building block.

The name data for the POI and TMC building block are subject to more frequent changes and, therefore, not stored in the Name building block.

The Name building block is a mandatory building block and is the only one of its type within a database complying with Navigation Data Standard.

Name data is used for:

- Location input by hierarchical address refinement
- Location input by proximity search
- Labeling maps
- Route guidance, route lists or speech output
- Address retrieval
- Speech input

Phonetic transcriptions used for speech output are stored in the Speech building block (see Chapter 14 *Speech Building Block* on page 281).

The structure of the Name building block balances low data redundancy and efficient access to name data. By assigning IDs to named objects according to Morton order, efficient local clustering of named objects is reached.

### Multilingual Support

Navigation Data Standard supports the management of names in different languages. Not only is it possible to store a name in multiple languages, but also to store several names that are used alternatively for a place or a road with the same language identifier.

For character encoding, UTF-8 is used. By using this format, databases are open for future language enhancements.

### Fast Identification of Locations

Navigation Data Standard offers additional support for navigation applications. So-called *selection graphs* can be defined that flexibly lay out the structures for location input. They are complemented by an indexing mechanism, so-called *next-valid-character trees*, allowing for a fast identification and selection of locations.

## 10.1 Building Block Structure and Content

Table 10-1 lists the data contained in the Name building block.

*Table 10-1 Data in Name building block*

Type of data	Description
Name features	<p>Feature class definitions are provided in the database for named object features representing real-world objects like roads and cities. The name strings representing the names of the named object features are stored with the named objects.</p> <p>For a detailed overview on the named object feature class, refer to Section 10.2 <i>Named Object Features</i> on page 167.</p>
NVC trees	<p>Next-valid-character trees can be defined to facilitate the input of strings. For more information, refer to Section 10.6 <i>Next-valid-character Trees</i> on page 186.</p>
Metadata	<p>The metadata of the Name building block comprise the following:</p> <ul style="list-style-type: none"> <li>- Selection graph definition</li> <li>- Selection criteria definition</li> <li>- Class-related metadata</li> <li>- Name-specific metadata in overall metadata</li> </ul> <p>For more information, refer to Section <i>Name Metadata</i> on page 165.</p> <p>The following metadata is closely related to names, but also used in other building blocks, such as the POI or the Traffic Information building block. This metadata is, therefore, stored in the common metadata:</p> <ul style="list-style-type: none"> <li>- Language definition: Languages supported in an update region, see Section 4.5.6 <i>Language Table</i> on page 70</li> <li>- Rotation metadata: Definition of rotation marker and rotation separation string, see Section 20.2.1 <i>Rotation Metadata</i> on page 353</li> <li>- Maximum string length, see Section 20.2.2 <i>Maximum Length for Name Strings</i> on page 354</li> </ul>

### Location Input

Navigation Data Standard supports location input by the following means: means of selection criteria, selection graphs, next-valid-character trees and a number of relations defined for named object features.

- Selection graphs (see Section 10.4 *Selection Graphs* on page 182)
- Selection criteria (see Section 10.5 *Selection Criteria* on page 184)
- Next-valid-character trees (see Section 10.6 *Next-valid-character Trees* on page 186)
- Relations between named object features (see Section 10.2.2 *Relations between Named Object Features* on page 173)

Navigation Data Standard also supports speech input and speech output for location input.

<b>Note</b>	Phonetic transcriptions are stored in the Speech building block (see Chapter 14 <i>Speech Building Block</i> on page 281).
-------------	--

## Use of Levels

The Name building block has no explicit leveling, meaning that name data is not physically stored in tiles as, for example, the data in the Routing building block. Named objects are stored as BLOBs together with their name strings in the `NamedObjectTable`.

## Name Metadata

Table 10-2 gives an overview of the metadata of the Name building block.

*Table 10-2 Metadata of the Name building block*

Metadata	Description
Name metadata (DataScript location: <code>nds.name.metadata</code> )	
Selection graph ( <code>SelectionGraphDefinition</code> )	Defines the structure for location input For more information, refer to Section 10.4 <i>Selection Graphs</i> on page 182.
Selection criteria ( <code>SelectionCriteriaDefinition</code> )	Defines the selection criteria that can be used in selection graphs and next-valid-character tree definitions For more information, refer to Section 10.5 <i>Selection Criteria</i> on page 184.
Named object class definition ( <code>NamedObjectClassDefinition</code> )	Specifies the definitions of available named object classes and their properties For more information, refer to Section <i>Class-specific Metadata in Name Building Block</i> on page 170.
Global metadata for Names (DataScript location: <code>nds.overall.metadata &gt; NameMetadata</code> )	
Mixed-case characters ( <code>isMixedCaseSupported</code> )	Defines whether mixed case is supported in NVC trees and whether name strings in mixed case are available
Name strings in upper or lower cases ( <code>areNameStringsInUpperCase</code> )	Defines whether name strings are in upper <b>or</b> lower cases: <ul style="list-style-type: none"><li>- If mixed case is supported, it defines in which case next-valid-character (NVC) strings are stored.</li><li>- If mixed case is not supported, it defines in which case NVC strings and name strings are stored.</li></ul>
Mixed case support ( <code>mixedCaseDescriptionList</code> )	Defines the mixed case support for several languages

Metadata	Description
Name string length (maximumLocationInputStringLength, maximumSignpostStringLength, maximumBasicMapDisplayStringLength, maximumRoutingStringLength)	Defines the maximum length for name strings For more information, refer to 20.2.2 <i>Maximum Length for Name Strings</i> on page 354.
Availability of NVC tree node attribute types (nvcTreeNodeAttributeTypeAvailability)	Defines the NVC tree node attribute types, which are available for an update region See also <i>Flexible Attributes in NVC Trees</i> on page 191.
Availability of named object attribute types (namedObjectAttributeTypeAvailability)	Defines the named object attribute types available for this update region

**Note**

The following metadata related to names is stored in the common metadata as it is also used in other building blocks, such as the POI or Traffic Information building block:

- Language definition (see Section 4.5.6 *Language Table* on page 70)
- Rotation metadata (see Section 20.2.1 *Rotation Metadata* on page 353)
- Maximum string length (see Section 20.2.2 *Maximum Length for Name Strings* on page 354)

## Dependencies to Basic Map Display Building Block

References are defined from the Basic Map Display building block to the Name building block. The references are needed for labeling maps.

For more information, refer to Section 10.8 *References of Named Object Features* on page 198.

## Dependencies to Routing Building Block

References are defined from the Routing building block to the Name building block. The references are needed for route guidance, route lists, and reverse geocoding, as well as for speech output to identify the features with the corresponding names.

For more information, refer to Section 10.8 *References of Named Object Features* on page 198.

## Dependencies to Speech Building Block

The speech data associated with name strings in the Name building block is stored in the Speech building block. Speech data can be phonetic transcriptions or prerecorded voice fragments. The Speech building block also offers the possibility to match the user's speech input with database items (Automatic Speech Recognition, ASR) which can be used, for example, for location input.

The Speech building block is an optional building block. Its availability is indicated in the `urBuildingBlockVersionTable`.

Phonetic transcriptions and prerecorded voice records reference the associated name string. This reference direction decouples the Speech building block from the Name, Traffic Information, Routing, and POI building blocks . There is an n-to-m relation between phonetic transcriptions and the name string they refer to.

The phonetic transcriptions for named objects are stored in the `NamedObjectPhonemeTable` (DataScript location `nds.speech.phoneticrepresentation`) together with a unique ID. This ID in the `namedObjectPhonemeTable` corresponds to the ID of the respective named object (`NamedObjectId`). The prerecorded voice data for named objects (identified by named object IDs) is stored in the `NamedObjectToPrerecordedVTable` and `NamedObjectPrerecordedVTable` (DataScript location `nds.speech.prerecordedvoice`).

For more information, refer to Chapter 14 *Speech Building Block* on page 281.

## 10.2 Named Object Features

A named object represents an object of the real world, for example, a country, a city, or a road. Different names that describe the same real-world object are related to each other. This makes it possible to identify an object via any of its names.

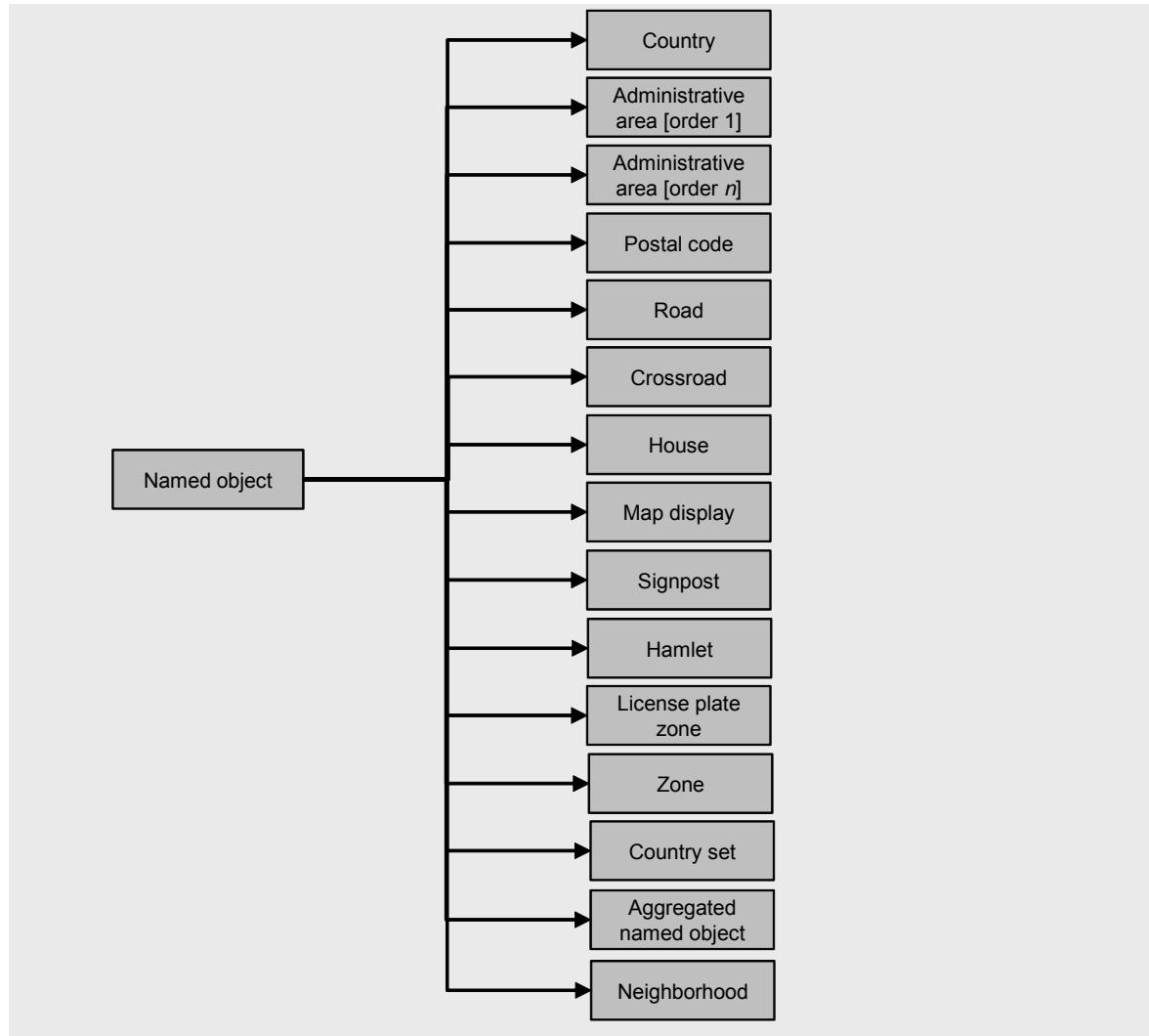
Named objects are stored as BLOBs in the `namedObjectTable`. Each named object in the table is uniquely identified by means of a `namedObjectId`.

Each named object can have one or more name strings. A name string is a textual description of a real-world object. This can be, for example, the name of a country, a city or a road, a telephone number, or a road number. Name strings are stored together with the respective named object in the `NamedObject` BLOB.

For more information on name strings, refer to Section 10.3 *Name String Features* on page 177.

To distinguish the manifold types of named objects, the named object feature classes depicted in Figure 10-1 are provided. The classes are defined in the metadata of the building block. The class structure can be extended by adding new classes if necessary.

Figure 10-1 Named object feature classes



## Country

Named country features are used to represent individual countries, like Germany, France, or Belgium.

## Administrative Area

Named administrative area features are used to represent administrative, geographical, or political units, for example:

- A collection of countries like the European Union or Benelux
- Federal states like Bavaria or Ohio
- A city like London or Hannover
- A city district like Soho (in London)

- A municipality like Unterhaching
- A postal area like SW1A 1AA or 80337

The different administrative area classes are distinguished by their ordinal number. Generally speaking, higher ordinal numbers characterize larger units, lower numbers smaller units. The highest order number is 1. However, the assignment of order numbers is country-specific and may vary in different databases. For Germany, for example, the assignment could be defined as follows: 1 = Bundesland, 2 = Landkreis, 8 = Gemeinde. For the US, it could be defined as follows: 1 = State and 6 = County.

### **Postal Code**

Named postal code features are used to represent postal areas and can be used as a selection criterion for destination input.

### **Road**

Named road features are used to represent links or a collection of links that are continuously interconnected and share a name, for example, Portobello Road or Champs Élysées.

A traffic place that is modeled as a link in the Routing building block is also represented by a named road feature. The same applies to numbered exits of freeways.

### **Crossroad**

Named crossroad features are used to represent the intersections of two or more roads. Usually, crossroads can be retrieved at runtime by the applications. In the following cases, however, crossroad features are needed:

- For crossroads that have a name of their own and that represent traffic places in the Routing building block.

This type of crossroad is typical, for example, in Japanese road networks. Another example of this type of crossroad in the US is Twelve Corners in Brighton, part of Rochester, NY.

- For crossroads that have no name of their own (for example, junction of Berlin Road and Main Road) if a next-valid-character tree is to be provided.

This is typically the case for input paths where the selection of crossroads follows that of a road without the context of a city, for example, Country > Road > Crossroad. Calculating crossroads at runtime for a road with a large number of links and crossing links can be very time-consuming. By modelling the crossroads as features, the corresponding objects can be loaded and presented for selection very fast.

For this type of crossroad feature, all roads that make up the crossroad must be connected to the feature using the crossroad-to-road relation. See Section *Crossroad-to-road Relation* on page 176.

### **House**

Named house features are used to represent buildings that can be identified by a name, a telephone number, a house number, or a house number range.

## Map Display

Named map display features are used to identify a point, a line, or an area displayed on a map, for example, a mountain peak, a river, a railway line, or a lake.

## Signpost

The named object feature class **SIGNPOST** is used for signpost names. For more information, refer to Section *NDS – Compiler Interoperability Specification*, 17.4 *Using Flexible Attributes for Signposts* on page 234.

## License Plate Zone

The named object feature class **LICENSE\_PLATE\_ZONE** is used for areas with a designated code which will be used for license plates. This code can be used as a selection criterion for destination input.

## Additional Classes

For named object features not covered by the feature classes introduced above, new classes may be added.

This could be, for example, a class for tourist areas or geographic regions like the Black Forest or the Rocky Mountains.

## Class-specific Metadata in Name Building Block

The metadata of the Name building block contains the definitions of the named object classes (DataScript location: `nds.name.metadata > NamedObjectClassDefinition`). Table 10-3 gives an overview of the properties of named object feature classes.

Table 10-3 Properties of named object feature classes

Property	Description
<code>iconSetId</code>	References the icon set, whose icons are used to visualize the named object feature, for example, in search masks. The use of this property is optional.
<code>namedObjectClassAttributeTypes</code>	Specifies which flexible attribute types are available for all or part of the named object features. This enables applications to access and evaluate the corresponding information. Flexible attributes include, for example, floor numbers of houses and the position of named object features.

## Properties of Named Object Features

Named object features are characterized by the following main properties:

- Attributes
- Relations between named object features
- References to next-valid-character trees
- Name string features

### 10.2.1 Attributes of Named Object Features

The following sections describe the attributes for named object features. The attributes are defined in DataScript at `nds.name.object`.

#### A Named Object Feature can be an Aggregation Feature

As a rule, exactly one real-world object is represented by a named object feature. There is, however, an exception to this rule: Real-world objects that share a name in the same language may be represented in the database by an aggregation feature. An aggregation feature is an umbrella object representing all features that have the same name.

The city Neustadt, for example, can be found in Germany repeatedly. Another example are roads for which a common base name exists, like the name Washington for Washington Road or Washington Square.

This property of named object features is modeled by means of the attribute `isAggregationFeature` (in `NamedObjectAttributes`).

Aggregation features are generated in the process of compilation. They can only be used for location input. For further selection of concrete features, one or more NVC trees are used that enable the user to disambiguate the aggregation named object (see Section 10.6.3 *Disambiguation in NVC Trees* on page 192).

#### A Named Object Feature can be a Destination

The attribute `isDestination` (in `NamedObjectAttributes`) specifies whether a feature is selectable as a destination. If set to TRUE, this can be an indication for applications that a further location refinement is not required.

By setting this attribute value to TRUE, the named object feature is marked as referencing at least one routing feature being a possible destination to which a route can be calculated.

Typically, the attribute value is set to FALSE if a named object feature references too many routing features or if a named object feature represents an area with a high number of links, for example, Bavaria.

For aggregation features, this attribute is set to FALSE.

#### A Named Object Feature Has a Reference Position

The attribute `hasPosition` (in `noAggregationFeatureAttributes`) can be used to display the named object feature on a map.

If there are, for example, two cities with the same name and both named city features are assigned this attribute, the corresponding features can be made visible on a map for disambiguation.

This attribute is not relevant for aggregation features.

## A Named Object Feature Has Access Location(s)

The attribute `hasAccessLocationList` (in `noAggregationFeatureAttributes`) is used to specify where a feature, for example, a house, can be accessed. A feature can have several access locations. The access locations are stored in the `accessLocationList` (DataScript location: `nds.name.object`) by means of location references.

A location reference may refer to a base or route link, a road geometry line, an intersection, a global or local position, or an AGORA or OPENLR reference. If a link or line is used, then one of the following can be referenced: a position on it, a range on it or the full link or line. The location reference also gives the side of the reference (left, right or both).

For location references to a different tile, a tile ID may be used (see also `referringToExternalFeature`).

This attribute is not relevant for aggregation features.

## A Named Object Feature Has a Geographical Extent

A named object feature can be defined as having a geographical extent. The geographical extent of a feature can be either specified by defining a bounding box or by listing the intersected tiles.

---

**Note** The number of tiles in the list of intersected tiles shall not exceed 255. For detailed information on how to handle cases with a higher number of intersected tiles, refer to *NDS – Compiler Interoperability Specification*, 7.5.3 *Geographical Extent* on page 150.

---

A geographical extent can be defined for the following purposes:

- To apply an automatic zoom for two-dimensional named object features. Using this attribute, features like cities or similar can be fully displayed on the monitor of a navigation system.
- To retrieve location information for places with a limited number of possible destinations, for example, for a small village.
- To retrieve the features related to a selected named object feature for location input or for map display. This is done by checking whether the selected feature references routing or basic map display features.

Thus, it is possible, for example, to retrieve the related features of a building represented by a named house feature on a map, or to find all links that are referenced by a road name feature. The list of links can then be passed to the Routing building block to calculate the route to the corresponding road.

The geographical extent can be set for all named object features that reference a feature in the Routing building block or in the Basic Map Display building block. The geographical extent is modeled by means of the attributes `hasBoundingRectangle` and `boundingRectangle`, as well as `hasTileList` and `extTileIdList` (in `NoAggregationFeatureAttributes`). To make it possible to retrieve location information for places with a limited number of possible destinations, the attribute `useBoundingRectangle` (in `NamedObjectAttributes`) is used.

The corresponding attributes are not relevant for aggregation features.

### A Named Object Feature Has a Class

The attribute `namedObjectClass` (in `NamedObject`) contains the class code of the named object feature class that this named object belongs to.

For more information, refer to Section 10.2 *Named Object Features* on page 167.

### Flexible Attributes

A named object may be assigned feature class-specific flexible attributes, for example, the population number for city named objects. The attributes are stored directly in the named object BLOB (DataScript location: `nds.name.object > NamedObject > attributeList`).

## 10.2.2 Relations between Named Object Features

Named object features can reference each other using different relation types. Table 10-4 shows the available relation types (DataScript location: `nds.name.object > NamedObjectRelationType`).

*Table 10-4 Relations between named object features*

Relation	Description
ACCESS_AT_ROAD	Connects a named house feature with a named road feature See Section <i>Access-at-road Relation</i> on page 174
CAPITAL_RELATION	Connects a named city feature with a named region feature for which a capital can be defined See Section <i>Capital Relation</i> on page 174
CONTAINED_IN	Connects a named object feature with a named region feature See Section <i>Contained-in Relation</i> on page 174
CROSSROAD_TO_ROAD	Connects a named crossroad feature with a named road feature See Section <i>Crossroad-to-road Relation</i> on page 176
ROAD_TO_HOUSE	Connects a named road feature with a named house feature See Section <i>Road-to-house Relation</i> on page 176
PARENT_NVC	Connects a named region object with another named region object See Section <i>Parent NVC Relation</i> on page 176

The following sections give detailed information on the different types of relations.

### Access-at-road Relation

The ACCESS\_AT\_ROAD relation connects a named house feature with a named road feature. It is used for houses with alternate addresses. This means, the same road has two (or more) different names and the same houses in this road have alternate numbers, depending on the road name they belong to.

---

**Example** House No. 1 on Washington Road has the alternate address House No. 77 on Road A 200.

---

In the database, the house is mapped to two named house features and the road is mapped to two named road features accordingly. By means of the access-to-road relation, the named house features are connected to the named road features they belong to. This relation is necessary to uniquely identify which house belongs to which road, but is not relevant for aggregation features.

### Capital Relation

The CAPITAL\_RELATION relation connects a named city feature with a named region feature for which a capital can be defined, for example, a country or a state. It can be evaluated to retrieve capital information which can then be used for map rendering: Different types of capitals can be displayed on a map using different styles.

This relation is not relevant for aggregation features.

---

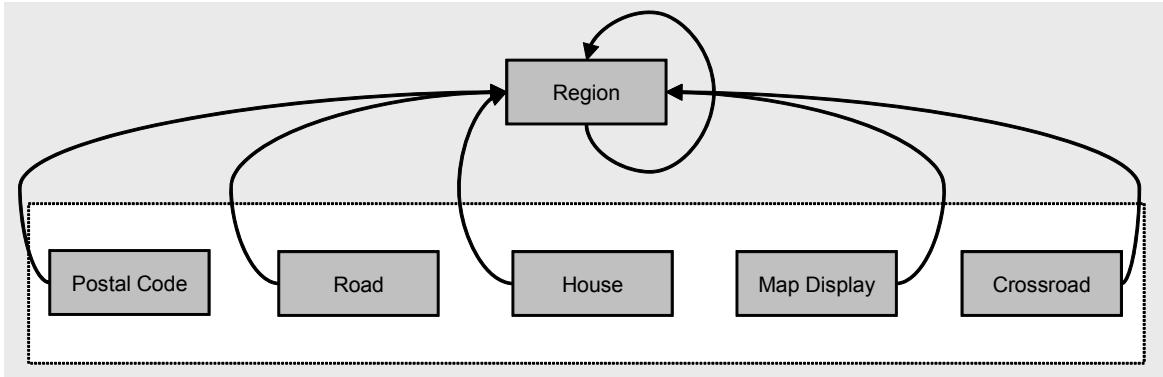
**Example** The following list gives examples for the capital relation:

- Paris is the capital of France.
  - London is the capital of the United Kingdom and the capital of England.
  - Munich is the capital of Bavaria.
- 

### Contained-in Relation

The CONTAINED\_IN relation is a mandatory relation that connects a named object feature to one or more named region features. It is used to model dependencies between named object features. This includes hierarchical dependencies, that is, political and administrative dependencies, as well as other dependencies like the allocation of postal areas. This type of relation is needed for retrieving address information. It can also be used for location input if no next-valid-character tree is defined for a specific named object feature.

Figure 10-2 Contained-in relation between named object features



A named object feature is usually connected with its direct parent. For disambiguation purposes, however, it can also be connected with named region features that are superordinate to its parent.

The relation may also be used if a feature is only partially contained in another feature. The relation is not relevant for aggregation features, and also not relevant for named country objects and artificial regions that are not connected to an administrative hierarchy, such as the Black Forest.

**Example** For Germany:

- Wendenstraße (road) is contained in Hammerbrook (city district), direct relation
- Wendenstraße (road) is contained in Hamm-Süd (city district), direct relation
- Wendenstraße (road) is contained in 20097 (postal area), direct relation
- Wendenstraße (road) is contained in 20537 (postal area), direct relation
- Wendenstraße (road) is contained in Hamburg (city), indirect relation
- Hamburg (city) is contained in Hamburg (federal state), direct relation
- Hamburg (federal state) is contained in Germany (country), direct relation

For Japan:

- A specific house is contained in a block (block), direct relation
- The block is contained in a city (city), direct relation

### Crossroad-to-road Relation

The CROSSROAD\_TO\_ROAD relation connects a named crossroad feature with the named road features that form the crossroad. The relation is mandatory for crossroads which are not determined at runtime but modeled as features. For more information, refer to Section *Crossroad* on page 169.

### Road-to-house Relation

The ROAD\_TO\_HOUSE relation connects a named road feature with a named house feature. This relation facilitates quick access from roads to houses. Using the ACCESS\_AT\_ROAD relation for retrieving the houses at a specific road would result in time-consuming search operations.

### Parent NVC Relation

The parent NVC relation connects two named region objects. This relation can be used to connect a named region object without NVC tree with a named region object with NVC tree. For searching an NVC tree in the `NvcTreeTable`, the named object ID of the related named region object with NVC tree is used.

A build-up area „Center“, for example, has no NVC tree. It uses the NVC tree of the related named region object „Munich“.

This relation makes it possible to enter city districts without related roads in a navigation system. Thus, the number of city districts available for destination input in the navigation system can be increased without increasing the size and reducing the usability. A typical effect of this method is that more roads than the user expects may be found for the city district. This is acceptable, however, as all available roads can be found, too.

### 10.2.3 References to Next-valid-character Trees

To provide a next-valid-character tree for a named object, a reference has to be established to the corresponding next-valid-character tree. A named object feature may reference more than one next-valid-character tree.

Available selection criteria are defined in the `SelectionCriteriaDefinition` structure (DataScript location: `nds.name.metadata > SelectionCriteriaDefinition`). Each selection criterion is mapped to one or more named object classes. This makes it possible for the abstract selection criteria to be broken down to concrete named objects. The selection criterion `place`, for example, is mapped to the named object classes `order 8` and `order 9`. The mapping is done to give information about the relation between the different concepts for selection criteria and named object classification.

`namedObjectTable.NvcSelectionCriteriaOptions` list the NVC tree availability for all selection criteria for a particular named object. The selection criteria referred to here are the same selection criteria as defined for the update region (in `nds.name.metadata > SelectionCriteriaDefinition`). The NVC tree availability (`nds.name.object > NvcTreeAvailabilityType`) indicates whether an NVC tree for the respective selection criterion is available.

The following types exist:

- LOAD: There is an NVC tree for each selection criterion (default case).
- GENERATE: There are no NVC trees for the selection criteria. The NVC tree must be calculated based on the existing relations between the named object features. This is required, for example, if the geographical extent is used.
- USE\_ALTERNATE: The NVC for this named object is the same as for another named object, which is identified in `alternateNamedObjectId`. Therefore, the NVC of the other named object shall be used. This can occur, for example, if CITY and POSTAL\_AREA share the same street list.
- NOT\_AVAILABLE: There is no NVC tree available. The NVC tree cannot be generated, either.

### Example

For an order 8 named object, the selection criterion „Road“ can be set to LOAD. For a road named object, the selection criteria „House“ and „Crossroad“ can be set to GENERATE. For another road, for which there are no house numbers stored in the database, the NVC tree availability is set to NOT\_AVAILABLE.

For more information on next-valid-character trees, refer to Section 10.6 *Next-valid-character Trees* on page 186.

## 10.3 Name String Features

A name string feature is the textual description of a real-world object represented by a character string or a number or a mixture of both. Name strings are features of the Name building block, and are modeled as `NamedObjectString` in `nds.name.namestring`.

One place can be identifiable by more than one name. For example, the capital of Belgium has three official names according to the three languages spoken in Belgium: Bruxelles, Brussel and Brüssel. In addition, there are translations for the name of the city into several languages, for example, Brussels (English), Bruselas (Spanish).

A named object can contain one or more name strings. The name strings identify the names available for the named objects including their attributes.

For more information, refer to *NDS – Compiler Interoperability Specification*, 7.7 *Name Strings* on page 155.

### 10.3.1 Attributes of Name String Features

Name strings are characterized by the following fixed attributes:

- Use type (`usageType`)
- Language (`languageCode`)
- Name format (`format`)

- Relation type (`relationType`)
- Non-printable control and separation characters (`hasControlCharacters`)

## Use Type

Table 10-5 shows available attributes for name string use types (DataScript location: `nds.name.object > usageType`).

*Table 10-5 Use types for name strings*

Use type attribute	Description
OFFICIAL_NAME	Defines a name string feature as the official name of the named object. An object can have several official names. The city of Brussels, for example, has three official names: Bruxelles, Brussel, and Brüssel.
ALTERNATE_NAME	Identifies a name string as a transliteration, exonym , or synonym name for a named object feature. For example, Leningrad instead of St. Petersburg.
DEFAULT_OFFICIAL_NAME	Identifies a name string as the default name string for a named object that is to be used for map display. In case of named object features with one or more official names, such as Bruxelles and Brussel, one of the official names shall be marked as default.
PREFERRED_ALTERNATE_NAME	Identifies the name string for a named object that is to be used for map display In case of named object features with one or more alternate names, one of the alternate names shall be marked as default. There are, for example, two alternate names for the Russian capital, an exonym and a transliteration of the Cyrillic name Москва: Moscow [RU_eng] or Moskva [RU_eng]

## Language

The attribute `languageCode` is assigned to name strings to specify their language (DataScript location: `nds.name.namestring > NamedObjectString`). For language-independent names like house numbers, telephone numbers or zip codes, the attribute is set to `UNDEFINED_LANGUAGE_CODE`.

The language attribute can be evaluated by the applications, for example,

- To display maps in official languages
- To display exonyms and/or transliterations only if they match the user-selected language

All names are stored in accordance to the Unicode code point order. For more information, refer to *NDS – Compiler Interoperability Specification, 7.3.4 Collation Rules* on page 139.

## Name Format

Names are strings consisting of letters, numbers, and/or special characters. They are categorized by the attribute `format` (DataScript location: `nds.name.namestring`). This makes it possible to apply format-specific input methods for the corresponding features.

Table 10-6 gives an overview on the available name formats.

*Table 10-6 Name formats*

Name format	Description
NAME (standard format)	<p>Standard names are used to identify any type of named object features consisting of letters and/or numbers, unless it belongs to one of the formats below.</p> <p>Standard names can contain rotation markers (for example “^”). The markers describe possible rotations of compound names.</p> <p>Example: The marker for “Robert-Bosch-Road” is set as follows: “Robert-^Bosch-Road”.</p> <p>For location input, the following input variants are supported and lead to the same result: “Bosch-Road, Robert-” (rotated name) and “Robert-Bosch-Road” (original name).</p>
HOUSE_NUMBER	House numbers are used to identify single houses.
HOUSE_NUMBER_RANGE	<p>House number ranges are used to encode sets of house numbers in an efficient way.</p> <p>House number ranges are optional in an NDS database. If available, they are stored with road geometry lines and base links. They may also be stored with route links. For route links extending over more than one tile, the house number range is split at tile borders.</p> <p>House number ranges are modeled as a name format and, thus, the same rules as for other name string data apply: Language code and transliterations for house number ranges are, therefore, stored in the same way as transliterations of other name formats.</p>
TELEPHONE_NUMBER	<p>Telephone numbers can be used to identify house named objects. They can, for example, be evaluated for direct dialing or for fast and straightforward location input.</p> <p>In addition, the area codes of telephone numbers can be used to identify countries, cities, areas, or other regions.</p>
ROAD_NUMBER	Road numbers can be used for administrative road classification. Defining names representing road numbers allows applications to display administrative road classes together with an icon on a map.
	<p><b>Note</b> This type of information cannot be derived from the corresponding links, because a link can be assigned multiple times or because of overlapping road classifications (a road can have, for example, a national and a European classification).</p>

Name format	Description
OFFICIAL_ABBREVIATION	This name string defines an official code or abbreviation for any named object feature class. This name format is used to speed up the location input in general and helps to uniquely identify countries. Hence, for country named object, this name string must be used to store an identifier for a real-world administrative structure. Examples: ISO country code, prefecture codes in Japan, or state codes in the USA.
MOTORWAY_INTERSECTION_N AME	Identifies the name of a motorway intersection Motorway intersection names can be used as an alternative to road names.
EXIT_NUMBER	Identifies motorway and other exit numbers Indicates the availability of a specific exit icon with a reference to the respective iconSetId If no icon is defined (hasIcon = FALSE), the default icon defined in the exitIconSetId in the RoadNumberClassPrefixTable is used.
ZIP_CODE_RANGE	Zip code ranges are used to store precalculated zip codes for city and city district named objects. They enable the application to display the zip code when the user selects a city. For a zip code range, all zip codes of a city or city district are collected. The identical numbers are kept, whereas the non-identical numbers are replaced by an "x" or another notation for arbitrary numbers. Example: The zip codes for Hattingen are: 45525, 45527, and 45529. This results in the following zip code range: 4552x.
MOTORWAY_EXIT_NAME	Identifies the exit name of a motorway exit. This name format makes it possible <ul style="list-style-type: none"> <li>– to use motorway exit names of road named objects for display instead of road names</li> <li>– to use motorway exit names of signpost named objects for display in context of signposts.</li> </ul>

## Relation Types

The relation type attribute is assigned to name strings to define the type of the string. The available relation types are described in Table 10-7 (DataScript location: nds.name.namestring > relationType).

Table 10-7 Relation type attribute for name strings

Name string relation	Description
BASE_NAME	<p>The name string is a meaningful part of a complete name.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>- Main [USA_eng]</li> <li>- Washington [USA_eng]</li> </ul> <p>Washington, for example, is the base name of the following complete road names: "Washington Road", "Washington Square", "South Washington Street", and so on.</p> <p>For named object features that share a base name, an aggregation feature is created in the database. For more information on aggregation features, refer to <i>A Named Object Feature can be an Aggregation Feature</i> on page 171.</p>
SYNONYM	<p>The name string is an alternate name in the official language of the region the feature belongs to.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>- Frankfurt [DEU_deu] for Frankfurt am Main</li> <li>- Stachus [DEU_deu] for Karlsplatz</li> <li>- 's-Gravenhage [NLD_nld] for Den Haag</li> </ul>
TRANSLITERATION	<p>The name string is an alternate name in a language that represents a non-official alphabet of the region the feature belongs to.</p> <p>This includes transcriptions and transliterations. The following would be the transliteration of Москва [RUS_rus] from the Cyrillic into the Latin alphabet: Moskva [RUS_rul].</p>
FIRST LETTER INPUT	<p>The name string is an abbreviated name.</p> <p>Example: WR for Washington Road</p> <p>Abbreviations are also common in countries like China where long name strings are shortened to the first letters of the string allowing for fast matches.</p>
EXONYM	<p>Alternate name string for a named object, for example, for a city or a country, that is not used locally, neither in an official nor in any local language.</p>
ALTERNATE SPELLING	<p>In some languages, one or more characters can be replaced by other characters, such as the umlauts in German: Muenchen, for example, is an alternate spelling for München. The same can be applied to diacritic characters, which are usually removed to create the alternate spelling; a, for example, would be the alternate spelling of â.</p>
NO_RELATION	<p>Name string has no relation to another string.</p> <p>A 1:1 relation exists between name strings. Therefore, the relation type must be set to NO_RELATION if no other relation exists.</p>

---

<b>Note</b>	Exonyms are stored with the name string use type for alternate names and marked as a non-official language. The following would be exonyms for München in Germany:
	<ul style="list-style-type: none"><li>■ Monaco [ITA_ita]</li><li>■ Munich [GBR_eng]</li></ul>

---

### Non-printable Control and Separation Characters

A name string can be passed on to the application without changes if the fixed attribute `hasControlCharacters` is set to FALSE. If the attribute is set to TRUE, the application has to filter out control characters for displaying the name string. DataScript location:  
`nds.name.namestring > hasControlCharacters`.

The control characters refer to:

- Special characters redefined in NDS (rotation marker, escape byte, rotation separation character). Those characters are defined as constants; DataScript location:  
`nds.overall.metadata`.
- Other non-printable control or separation characters like soft hyphen or line break

---

<b>Note</b>	The rotation marker is not allowed in name strings of NVC trees. The same is true for all other control and separation characters, except for the escape byte and the rotation separation marker.
-------------	---

---

## 10.4 Selection Graphs

A selection graph is a directed acyclic graph which defines the index structures and the input paths available for location input. Input paths describe the sequence of location types, such as countries, cities, or roads, during location input. A sample input path could be: `country > postal code > street > house number`. Different types of locations are represented by different selection criteria.

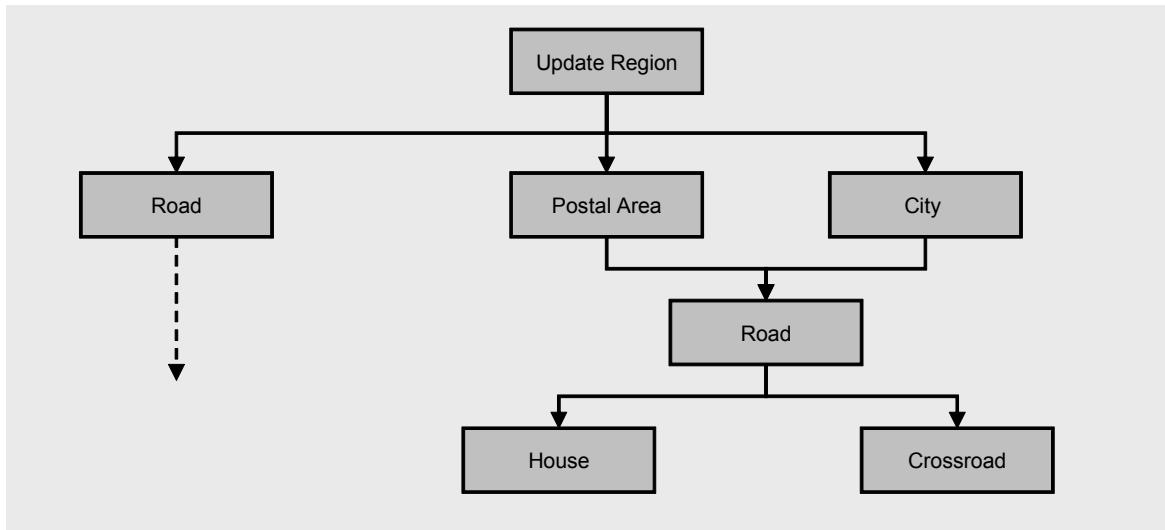
Selection graphs can be country-specific or region-specific, as well as specific to a customer. The definition of a selection graph can be based on administrative structures of a region or country, or it can focus on the search habits of the consumers of a country. Whereas in the United States the selection of a road immediately follows the selection of a federal state, Europeans usually select a city before selecting a road. A selection graph may provide alternative input paths: One can be adapted to the administrative structures of a region, while a second path reflects the search habits of the consumers.

For an update region, one or more selection graphs can be defined. A selection graph is usually available per region named object identifying its scope. These region named objects also define the global entry to the database. In the update region Benelux, for example, there can either be three selection graphs, one for each country (Belgium, Netherlands, Luxemburg), or there is just one selection graph for the complete update region Benelux.

The selection graph definition is stored in the metadata of the Name building block. 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.

Figure 10-3 shows a selection graph as it could be defined for a European update region, for example Germany.

*Figure 10-3 Selection graph, example*



The nodes (for example `city`) of the graph represent the selection criteria, the arrows (edges) describe the path from one selection criterion to the next criterion or criteria. In the example above, the path from `Update Region` branches to the nodes `Road`, `Postal Area` and `City`.

### Disambiguation in Selection Graphs

Named object features that match a selection criterion in a graph can be ambiguous. This is generally the case for aggregation features. Therefore, provisions must be taken to ensure that real-world objects are uniquely identified. To this end, the definition of a selection graph also contains disambiguation information for selection criteria (see Section 10.6.3 *Disambiguation in NVC Trees* on page 192).

## Selection Graphs and Name Formats

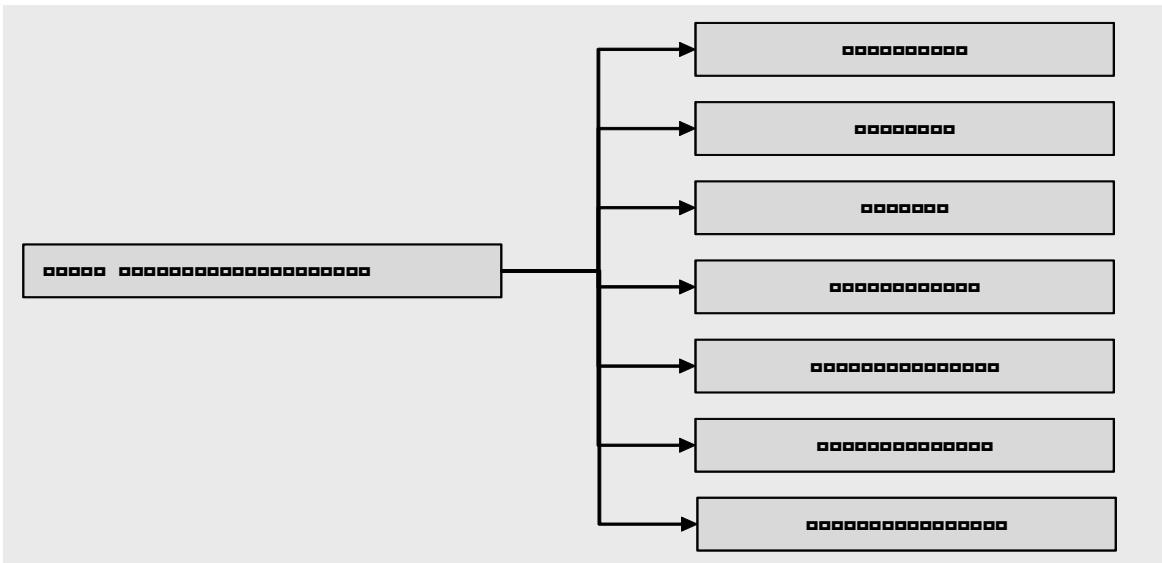
Selection graphs can also contain information about name formats. By this means, it is possible to handle, for example, road numbers or official codes in separate next-valid-character trees. By default, it is assumed that a next-valid-character tree uses standard names.

For more information on name formats, refer to Section 10.3.1 *Attributes of Name String Features* on page 177.

## 10.5 Selection Criteria

Selection criteria are used in selection graphs to present named object features for selection during location input. Each named object feature class can be mapped to one or more selection criteria. For example, the named object feature classes ORDER\_8 and ORDER\_9 are mapped to the selection criterion “Place” which groups together all named features of type ORDER\_8 and ORDER\_9 of a specific country.

Figure 10-4 Predefined selection criteria, overview



Selection criteria are used for the precompilation of selection options. However, applications may choose to overrule the precompiled options.

Selection criteria definitions are stored as metadata 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 selection metadata table has exactly one row, which is identified by an arbitrary artificial ID.

Table 10-8 shows the properties of selection criteria (DataScript location: `nds.name.metadata > SelectionCriteriaDefinition`).

Table 10-8 Properties of selection criteria

Property	Description
nameList	<p>Defines region-specific names for a selection criterion</p> <p>Depending on the region for which a selection criterion is used, its name can differ. For example, a federal state in the USA is called "State", whereas in Canada it is called "Province". The name can be stored in different languages. To identify the language, the language code is stored with the region-specific name.</p> <p>The following applies:</p> <ul style="list-style-type: none"> <li>- If the names are identical for the complete update region, the selection criterion names are defined in the metadata of the Name building block.</li> <li>- If different names are used for the same type of areas within the same update region, selection criterion names are defined in the global metadata (see Section 20.2 <i>Common Data</i> on page 352). In this case, the attribute <code>hasRegionSpecificNames</code> must be set to TRUE.</li> </ul>
namedObjectClassList	<p>Creates a relation between a selection criterion and a named object feature class</p> <p>A selection criterion can be mapped to one or more feature classes (1:n relation).</p>
iconSetId	References the icon set, whose icons are used to visualize the named object feature, for example, in search masks. The use of this property is optional.

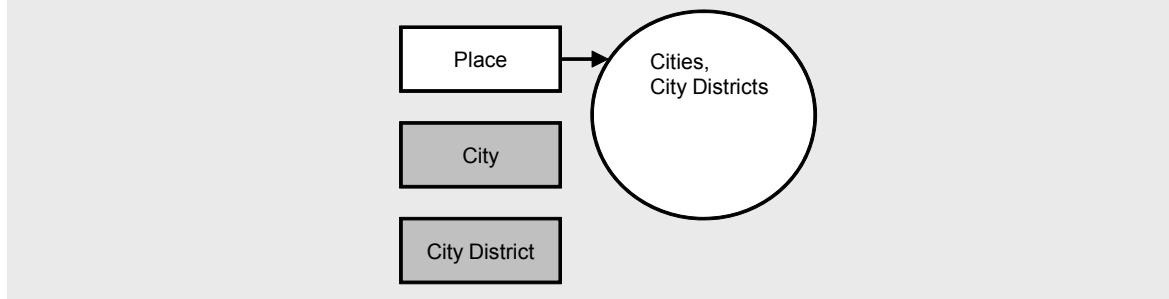
### Combining Named Object Feature Classes to a Selection Criterion

Named object feature classes form the basis for the majority of selection criteria used in selection graphs. Usually one named object feature class is represented by one selection criterion. However, it is sometimes useful to define selection criteria that group together several named object feature classes to facilitate location input and to allow shortcuts.

Typically, the administrative area feature classes City and City District are combined to one criterion. By grouping them to a new selection criterion Place, end users of a navigation system can look for the name of a place without further knowledge about administrative structures.

For selection criteria that are the result of a grouping, common indices can be prepared and used in next-valid-character trees.

Figure 10-5 Example: Mapping between selection criteria and named object feature classes



## 10.6 Next-valid-character Trees

Next-valid-character trees (NVC tree) facilitate fast selection of names from large lists that have been precompiled on the basis of the selection criteria defined in the metadata. Instead of browsing through a long list of names, users can traverse the tree stepwise. With each step, only the valid continuations (given by the tree nodes) are offered for selection. For valid continuations, the number of remaining matches is given.

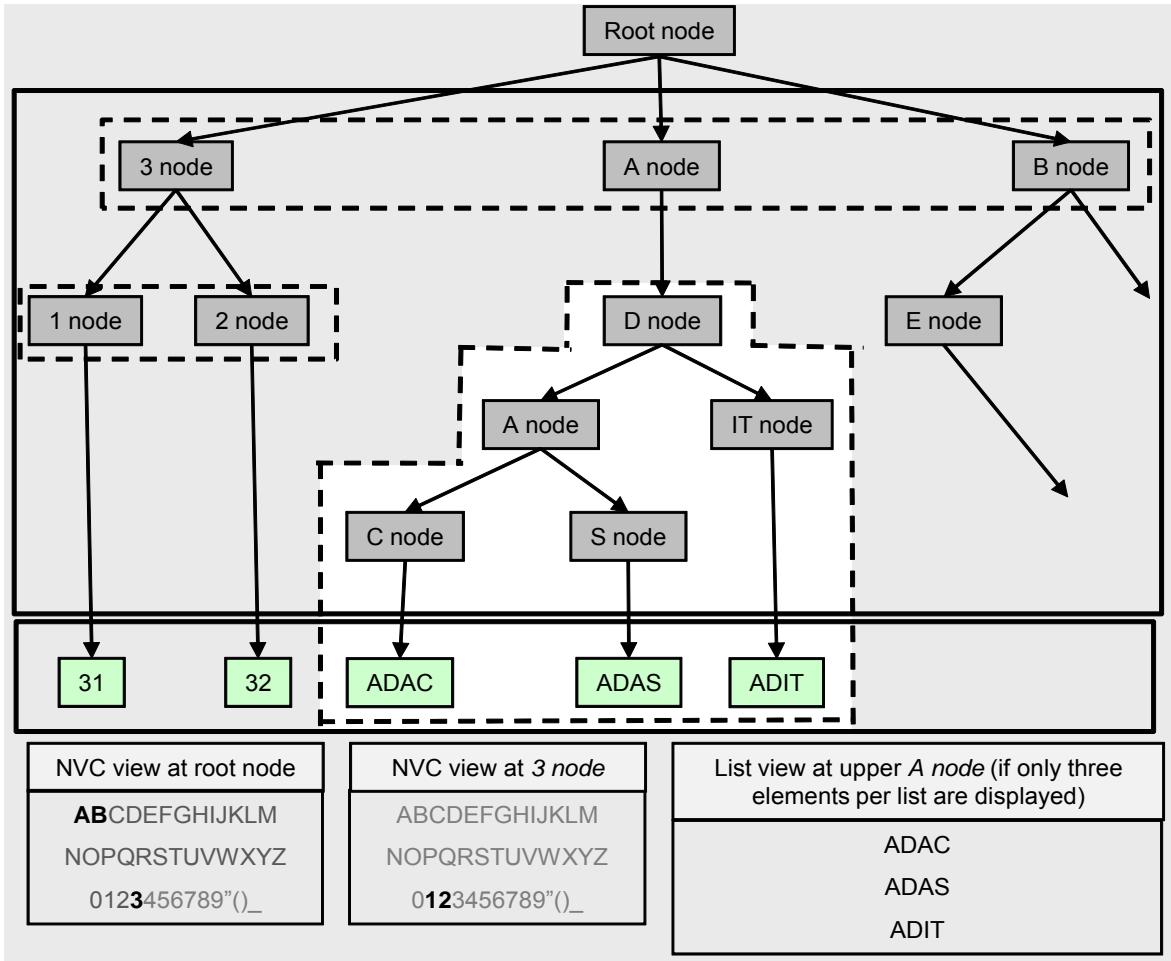
Most of the input paths in a selection graph will be supported by NVC trees. NVC trees are stored in the Name building block (DataScript location: `nds.name.nvctree`).

**Note**

NVC trees can also be created for POIs. POI NVC trees are stored in the POI building block, see Section 13.6.4 *Next-valid Character Trees for POIs* on page 278.

Figure 10-6 shows an example of an NVC tree referring named object features.

Figure 10-6 Next-valid-character tree, example



### Related Topics:

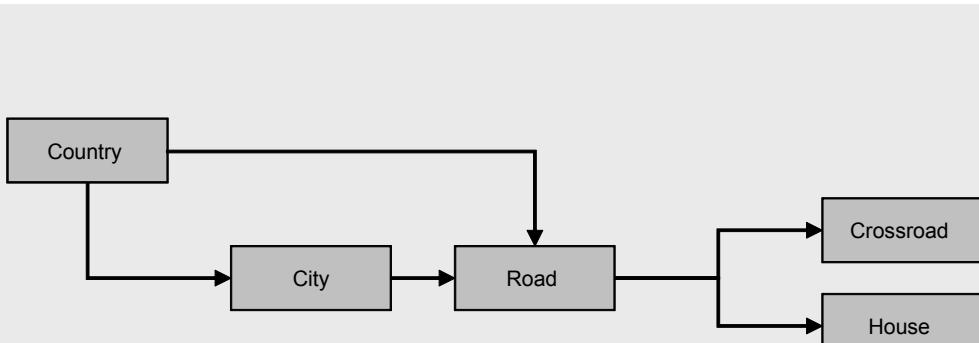
- *NDS – Compiler Interoperability Specification, 7.5.2 Next-valid-character Trees on page 141*

#### 10.6.1 Selection Graphs and NVC Trees

To support location input, several NVC trees are needed. Their exact number and type depend on the structure of the selection graph. Generally, for each edge of the selection graph an NVC tree can be defined. The definition contains the information to which edge of the input path the NVC tree refers. An NVC tree can be compiled for either an individual selection criterion or for a group of selection criteria.

In the following example, a road can be selected either by directly using the input path **Country > Road > ...** or by first choosing a city using the input path **Country > City > Road > ...**

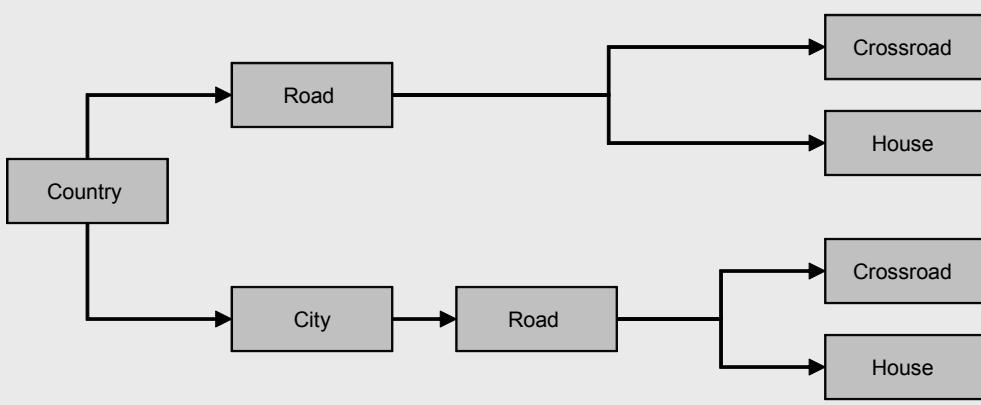
Figure 10-7 Selection graphs, example



For the two edges on the first level of the graph, two different NVC trees are provided: One containing all road names of the country, a so-called road tree, and a second containing all city names, a so-called city tree. Selecting a city will lead the user to a new NVC tree, containing all roads of the selected city only.

The following figure illustrates the physical representation of NVC trees associated with the selection graphs.

Figure 10-8 Selection graphs and NVC trees, example



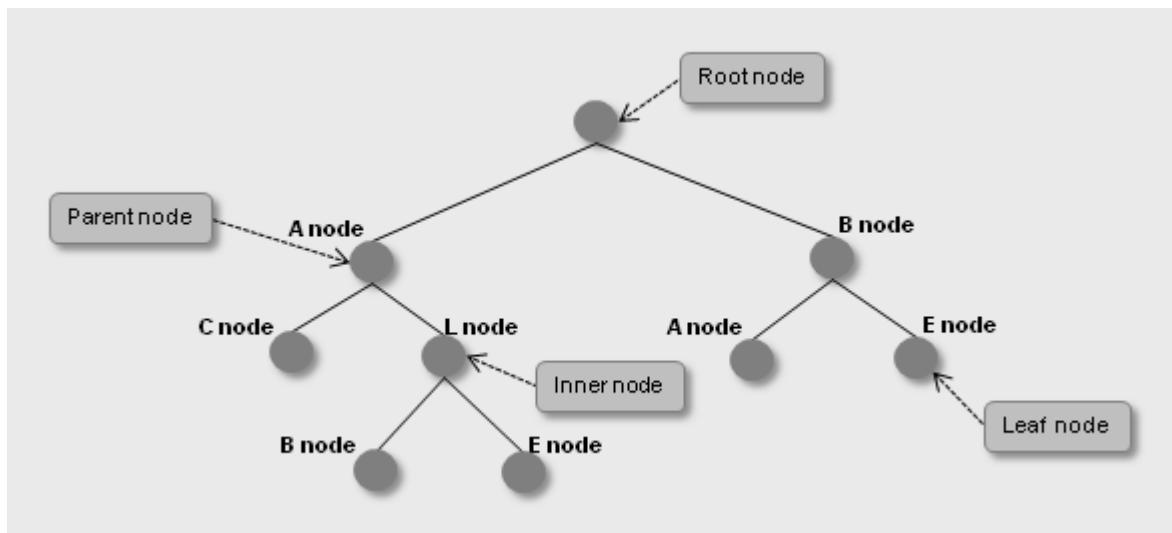
## 10.6.2 Structure of NVC Trees

An NVC tree is referred to by one or more named object features. It is made up of nodes and edges. The nodes represent characters that can be entered. For a given node, the edges lead to the node's child nodes. The child nodes represent possible continuations after selecting previous characters.

There are the following node types:

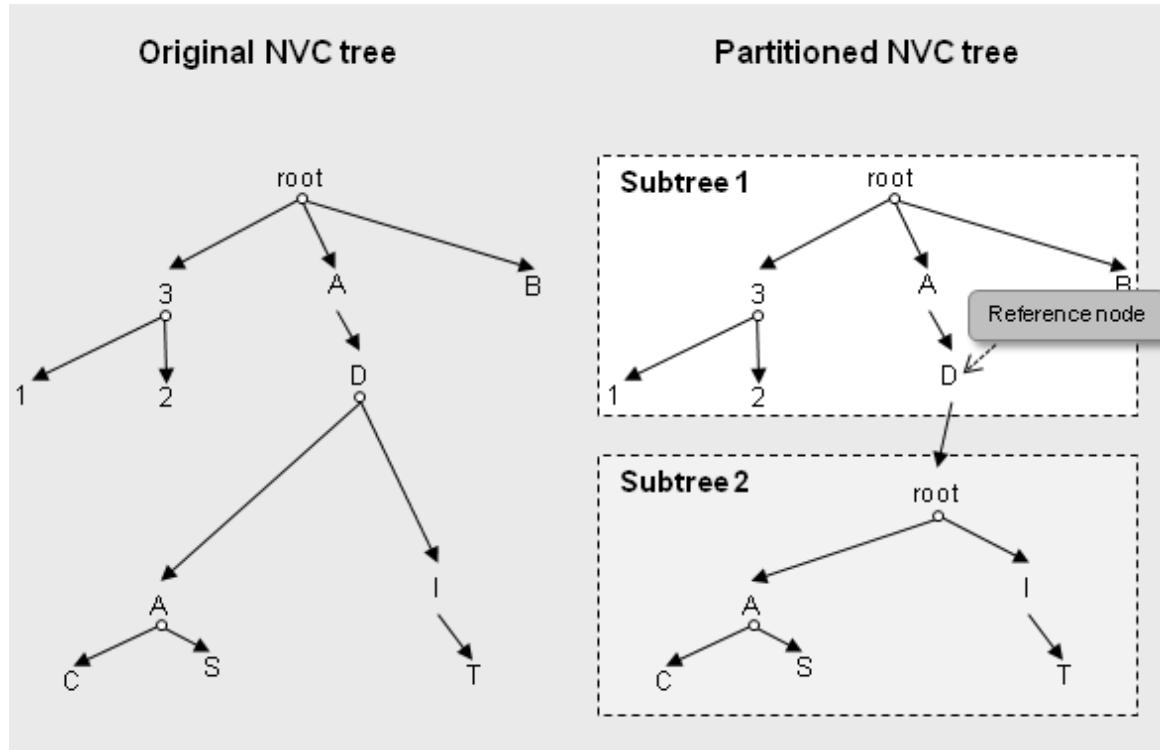
- The topmost node in the tree is called *root node*.
  - A node with subordinate nodes is called *parent node*. The subordinate nodes are called *child nodes*.
- Child nodes with the same parent node are referred to as *sibling nodes*. Sibling nodes are sorted according to the Unicode code point order (see Section 10.3.1 *Attributes of Name String Features* on page 177).
- Nodes that have both a parent and child nodes are called *inner nodes*.
  - A node without child nodes is called *leaf node*. The path leading from the root node to a leaf node represents typically one name string. The leaf node then has n references to named object features with this name string.

Figure 10-9 Next-valid-character tree, terms



NVC trees may be partitioned to enable partial loading of NVCs by applications. During partitioning of an NVC tree, a *reference node* is created pointing to the subtree. For more information about partitioning, see *NDS – Compiler Interoperability Specification, Size and Partitioning* on page 144.

Figure 10-10 Partitioning of NVC trees, example



The different node types are represented as follows in DataScript (location: `nds.name.nvctree > TreeNodeType`).

Table 10-9 Data structures for NVC tree node types

Data	Description
<code>TreeInnerNode</code>	<p>Represents an inner node of an NVC tree. Inner nodes must have at least one child node. The root node of the NVC tree also has this type; the root node must be the first one in the list of nodes.</p> <p>Information, such as the number of contained names or biggest city attributes, is only stored in reference nodes and leaf nodes. The information for inner nodes is calculated online based on the information stored in the reference and leaf nodes.</p> <p>For more information, refer to <i>Reconstructing Information from Leaf and Reference Nodes</i> on page 191.</p>
<code>TreeLeafNode</code>	Represents a leaf node and thus a path name, which is the concatenation of all prefix strings on the path from the NVC root node to this node
<code>TreeShortLeafNode</code>	The short leaf node is an optimized leaf node with a reference to one named object or POI with exactly one name. For POIs, it has exactly one category. The short leaf node makes the standard cases more space efficient: It is not necessary to store, for example, the number of contained names and categories as they are always one.

Data	Description
TreeRefNode	Represents a reference node to another NVC subtree

<b>Note</b>	Instead of storing complete named object references, an NVC tree stores in the NVC tree nodes one base named object reference (DataScript location: <code>NvcSubTreeBlob &gt; namedObjectBaseRef</code> ) and relative named object references (DataScript location: <code>namedObjectRelativeRef</code> in <code>NvcNamedObjectReference</code> and <code>TreeShortLeafNode</code> ). Based on this information, the complete named object ID can be reconstructed as follows: <code>namedObjectId = namedObjectBaseRef + namedObjectRelativeRef</code> .
<b>Note</b>	The rotation marker is not allowed in name strings of NVC trees. The same is true for all other control and separation characters, except the escape byte and the rotation separation marker.

## Reconstructing Information from Leaf and Reference Nodes

To facilitate compact storage, only minimal information is stored in the NVC tree. The application has to reconstruct the missing information at runtime. The number of contained names of a node, for example, is stored in leaf and reference nodes only. To get this information also for the inner nodes, the application has to use the following algorithm recursively:

- The number of contained names of a node is the sum of contained names of its child nodes.
- The root node of the NVC is the starting point.
- For nodes, which are no leaf or reference nodes, determine the number of contained names from the sum of contained names stored in its child nodes.
- For leaf or reference nodes, determine the number of contained names from the attribute `numContainedNames`.

The same applies to POI categories, languages, or flexible attributes.

## Flexible Attributes in NVC Trees

NDS provides additional flexible attributes for NVC nodes. One example is the `EXTENDED_POSTAL_CODE_POSITION` attribute, which contains the relative geographic position for an extended postal code. Another example are the attributes for storing the names of big cities.

Flexible attributes for NVC nodes are provided in a simple attribute list. The availability of a simple attribute list is indicated by a flag at the respective node.

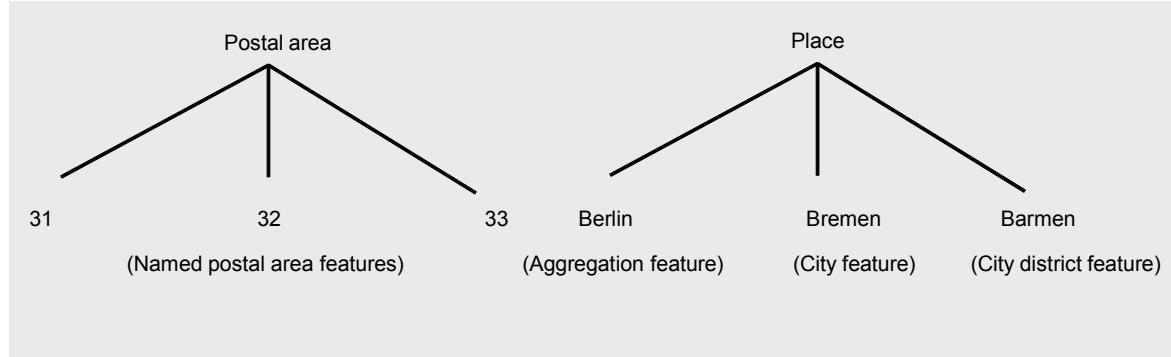
See also:

- Overview of flexible attributes in Appendix D *List of Flexible Attributes* on page 381
- *NDS – Compiler Interoperability Specification, Big Cities in NVC Trees* on page 148

### 10.6.3 Disambiguation in NVC Trees

In the following example, two NVC trees are available on the first level of the selection graph: One tree contains all postal area codes of Germany, and the other tree contains the names of all cities and city districts for which navigation data is available.

Figure 10-11 Disambiguation, example 1



The codes of the postal areas are unambiguous: Each postal area code exists only once. In contrast to this, the names of places can be ambiguous. Therefore, provisions must be taken to uniquely identify real-world objects. To this end, the corresponding selection criteria are assigned one or more further criteria for disambiguation. This is done in the selection graph definition.

In this example, the following selection criteria are defined for disambiguation:

Figure 10-12 Disambiguation, example 2

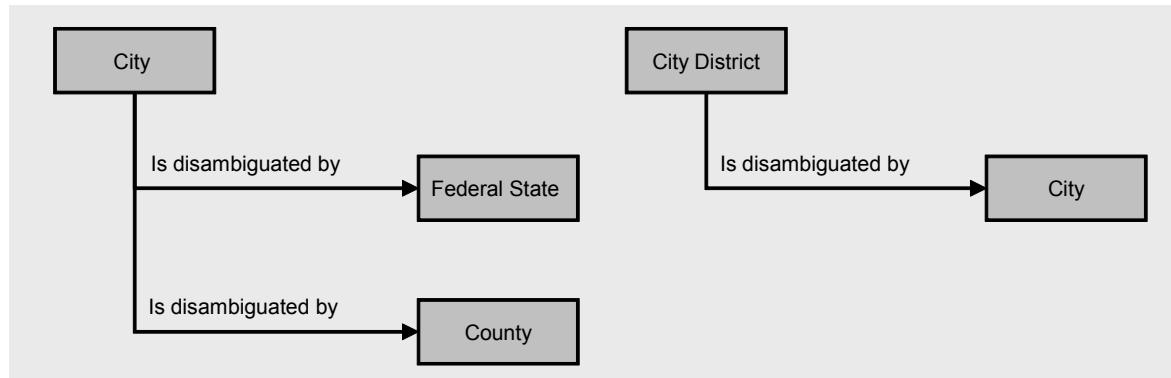
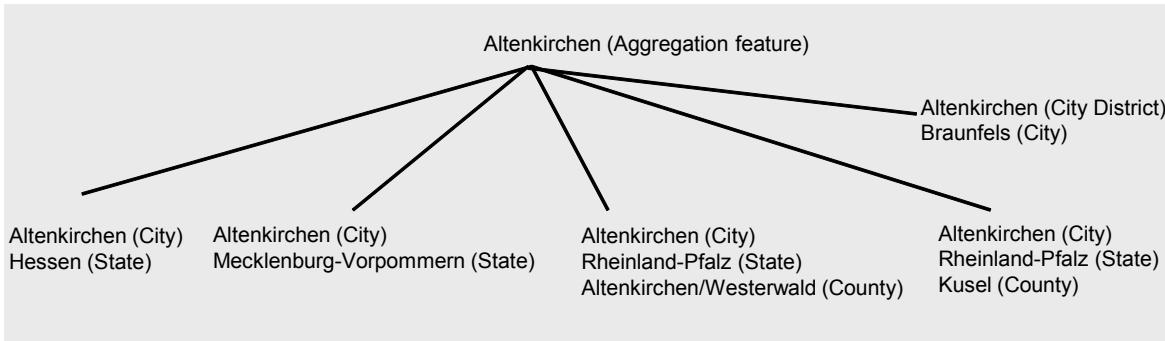


Figure 10-13 shows an example where all defined disambiguation criteria are needed to uniquely identify the different places that belong to the aggregation feature Altenkirchen.

Figure 10-13 Disambiguation, example 3



There are two different ways to store the features retrieved from an aggregation feature. The storage methods can be specified during compilation:

- In one NVC tree. Displaying ambiguous features using one NVC tree is advantageous if the number of retrieved features is large (see Figure 10-13).
- In several NVC trees, one tree for each disambiguation criterion. The ambiguous features could be disambiguated, for example, using the criteria **State** and **Municipality**.

#### Related Topics:

- *NDS – Compiler Interoperability Specification, Disambiguation in NVC Trees* on page 146

#### 10.6.4 Languages in NVC Trees

NVC trees may contain names in more than one language. In order to support displaying the official language and the language selected by the user, but not any other languages, NVC trees contain language filter information.

The number of remaining matches in an NVC node can be derived for each language.

Table 10-10 lists the language information that is available for NVC trees.

Table 10-10 Language information in NVC trees

Data	DataScript location	Description
defaultLanguageCode	nds.name.nvctree > NvcSubTreeBlob	Defines the default language code for name strings referenced by leave nodes or reference codes
useDefaultLanguage	nds.name.nvctree > TreeRefNode TreeShortLeafNode and NvcGeneralPoiOrNameObjectReference	Indicates that only the default language must be used. If set to FALSE, the language code list is stored in the respective structure.

Data	DataScript location	Description
numNamesInDefaultLanguage	nds.name.nvctree > TreeRefNode and NvcGeneralPoiOrNamedObjectReference	Defines the number of name strings in the default language if useDefaultLanguage is set to TRUE
numLanguages	nds.name.nvctree > TreeRefNode and NvcGeneralPoiOrNamedObjectReference	Defines the number of language descriptions for the referenced subtree, POI or named object reference if useDefaultLanguage is set to FALSE
languageDescription	nds.name.nvctree > TreeRefNode and NvcGeneralPoiOrNamedObjectReference	Defines a list of languages of name strings in the referenced subtree, POI or named object reference with the number of name strings per language if useDefaultLanguage is set to FALSE
languageCode	nds.name.nvctree > LanguageDescription and TreeShortLeafNode	Defines that a related node, POI or named object reference refers to one or more name strings of the POI or named object of subsequent nodes in the language specified by the language code
numNames	nds.name.nvctree > LanguageDescription	Defines the number of names in the specified language referred in the referenced subtree (including recursion into other subtrees) or in the referenced named object

#### Related Topics:

- *NDS – Compiler Interoperability Specification, Ideograms in NVC Trees* on page 147
- *NDS – Compiler Interoperability Specification, Filtering of Names in Different Languages in NVC Trees* on page 147

#### 10.6.5 Mixed-case Characters in NVC Trees

In the NVC list view, users normally expect that names are represented in mixed-case characters, for example, New York instead of NEW YORK. However, users do not want to switch between upper and lower cases when entering names for location input and using the next-valid-character tree. In general, this requirement can be fulfilled by storing NVC strings in either upper or lower case only and using mixed-case characters in name strings. Low-performance applications, however, cannot retrieve name strings from the named objects referenced by the NVC nodes for the NVC list view in an acceptable time.

In order to support an application to generate mixed-case characters for NVC list views, NVC strings shall be stored in one case only.

The Name metadata (DataScript location: `nds.overall.metadata > NameMetadata`) contains the information whether upper or lower case is used. The metadata also contains the information that applications require to change cases: change case markers and change case rules.

- Change case markers: To indicate the parts of an NVC string that require a case change, additional markers shall be added. These markers enable applications to change the string to a different case at the marked point. Storing such markers is more size-efficient than duplicating the complete string.  
NDS provides two ways to mark what cases to change:
  - Separation characters
  - Case change masks
- Change case rules: The metadata of the Name building block also defines rules how to change the marked cases. The rules define a source character (for example “a”) and a target character (for example “A”) for the conversion.

For additional information on the metadata of the Name building block, see Section *Name Metadata* on page 165, Table 10-2.

### Separation Characters

The application can use separation characters to indicate that the case of the character following the separation marker has to be changed. Space (ASCII code 0x20) or the start of text (ASCII code 0x02), for example, may be defined as separation characters.

For many languages and names, applications can use general rules based on separation characters to create mixed-case strings from single-case strings. In these cases, additional markers are not required.

---

**Note**      NDS recommends to use separation characters for marking case changes.

---

**Example:** The following rules for changing characters are stored in the metadata of the Name building block: r > R, b > B, s > S. As separation characters “-” and the start of the text are defined. Applying these rules to the NVC name string “robert-bosch-straße”, results in the correct spelling “Robert-Bosch-Straße”.

### Change Case Mask

While the use of separation characters is recommended to mark case changes, some NVC name strings cannot be correctly transformed using separation characters.

The NVC name string “an der waldquelle” requires a special marker for the characters “a” and “w”. A general rule in the application specifying the blank as separation character results in “An Der Waldquelle” instead of the correct spelling “An der Waldquelle”.

To change cases correctly, some strings require a change case mask. As change case masks are not as size-efficient as separation characters they shall only be used when no correct transformation with separation characters is possible.

---

**Example** The NVC name string “an der inn” would require change case markers for the characters “a” and “i”. (“der” is an article and therefore written in lower case.) The blanks are defined by separation characters. For this example, the ChangeCaseMask is filled as follows:

```
ChangeCaseMask
{
    uint8 numCharacters = 10
    Bool changeCaseOfCharacter[numCharacters] =
    {TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE};
}
```

---

For more information, refer to `nds.overall.metadata > NameMetadata` in the *NDS – Physical Model Description*.

### 10.6.6 Extended Postal Codes in NVC Trees

Extended postal codes support location input via long postal codes which provide a point location for a postal code. They consist of a base postal code for a postal code area and a suffix. Extended postal codes are stored in NVC trees by means of storing the following information at leaf nodes:

- Reference to a postal code named object for the postal code area of the base code
- Relative geographic position

#### Related Topics:

- *NDS – Compiler Interoperability Specification, Extended Postal Codes* on page 148

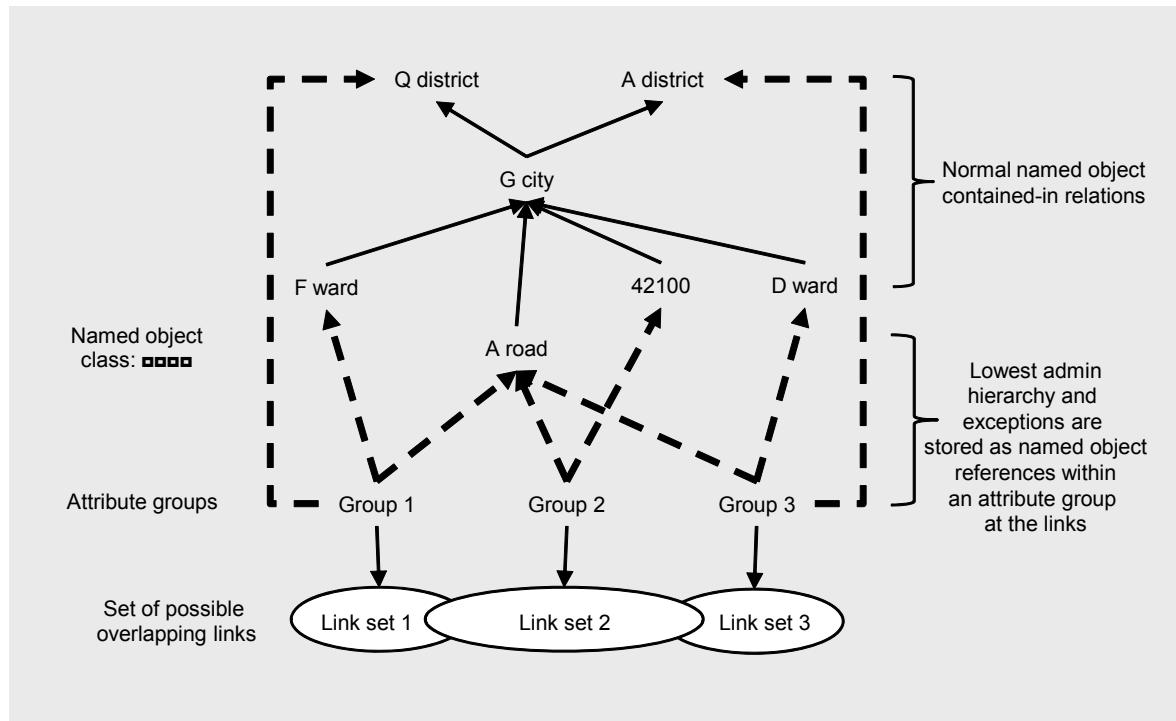
## 10.7 Address Retrieval

Navigation Data Standard supports the retrieval of addresses. For each update region, an address format can be specified in the global metadata. This format defines the address fields used in the corresponding region. An application can collect the elements of an address by following the contained-in relation starting with a routing or basic map display feature.

For more information on the contained-in relation, refer to Section 10.2.2 *Relations between Named Object Features* on page 173.

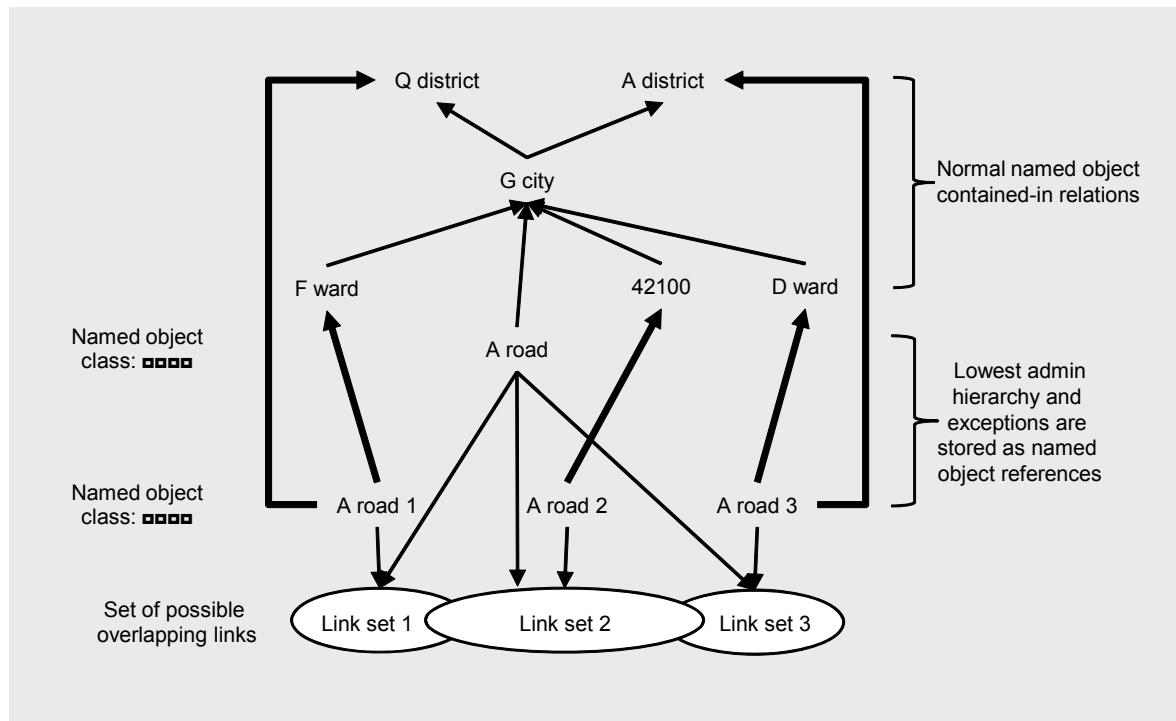
Contained-in relations are used to retrieve the unique administrative hierarchy of road named objects. Road named objects can, however, span over several administrative areas, for example, several city districts and postal areas. Only a part of the assigned links of the respective road named object is part of the spanning administrative areas. The references from respective links to these administrative areas can be stored in form of attribute groups in the Routing building block (see Figure 10-14).

Figure 10-14 Name hierarchy for address retrieval, roads and grouping



Alternatively, the road can be split into several road named objects, which all have the same administrative area hierarchy (see Figure 10-15). In this case, more than one road named object with the same name can be assigned to a link. It is, therefore, necessary to indicate, which of the road named objects shall be used for map display. This is done by means of the flexible attribute `CONCRETE_NAMED_OBJECT`.

Figure 10-15 Name hierarchy for address retrieval, roads only



The two approaches may be combined or used alone.

## 10.8 References of Named Object Features

Named object features are referenced by basic map display features and by routing features. For more information, refer to *NDS – Compiler Interoperability Specification, 7.4 Allocation of Named Objects to Basic Map Display and Routing Levels* on page 140

### References from Basic Map Display Features to Named Object Features

The Name building block provides the name data for features of the Basic Map Display building block. For more information on the features provided in the Basic Map Display building block, refer to Chapter 11 *Basic Map Display Building Block* on page 201.

Table 10-11 shows the name references that can be established from features of the Basic Map Display building block to named object features.

Table 10-11 References from the Basic Map Display to the Name building block

Basic map display feature	Named object feature
Area, line, point	Map display
Area, line, point	Region
Line	Road

## References from Routing Features to Named Object Features

The Name building block provides the name data for features of the Routing building block. These names are needed for providing route lists, map labeling, address retrieval, signposts and speech input/output. For more information on the features of the Routing building block, refer to Chapter 9 *Routing Building Block* on page 107.

Table 10-12 shows the name references that can be established from the features of the Routing building block to named object features. References from routing features to named objects are stored in the name attribute layer.

*Table 10-12 References from the Routing to the Name building block*

Routing feature	Named object feature
Base link, route link, road geometry line, intersection	House
Base link, route link, road geometry line, intersection	Road
Base link, route link, road geometry line, intersection	Region (administrative area)
Base link, route link, road geometry line, intersection	Signpost
Intersection	Crossroad

## 10.9 Address Points

From an administrative point of view, a house may belong to a specific street, but not be located at that street. Also, it is possible that the actual point of access to the house is on a third street.

---

**Example** House numbers in Russia: Moscow, ulitsa Geroyev Panfiolvtsev, 49 KORP 1, 141 KORP STR 6

---

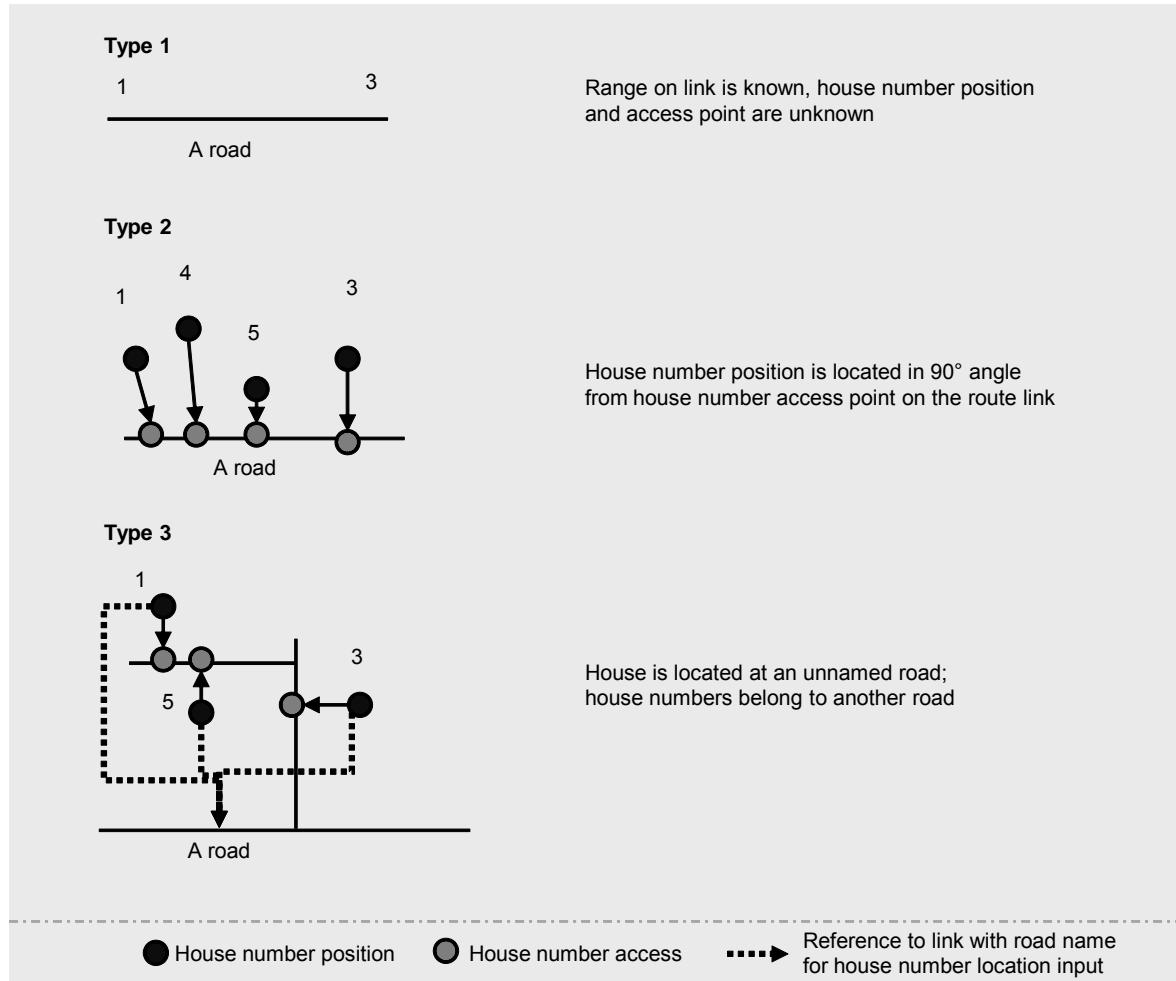
Address points are used to model the correct location of the house number on the map as well as the correct access point for routing to the address. Address points are assigned to house number ranges and stored in the name attribute layer.

---

**Note** Although address points are stored in the Routing building block, they logically belong to the Name building block.

---

Figure 10-16 Types of address points



Address points carry the following information:

- House number (mandatory): Name string for the house number
- House position (optional): Defines the location of the house by means of the `CoordXYOffset` position. The offset is relative to the reference point of a tile containing the link which is related to the point address.
- Position of an access to a house (optional): Defines the position of the access point for the house number on a line feature
- Link reference (optional): Gives an internal or external directed link reference

**Note**

Address points may have transliterations, see DataScript  
`nds.name.namestring > AddressPointHouseNumber`

## 11 Basic Map Display Building Block

The Basic Map Display building block groups together the features for map display applications. It contains the map features necessary for rendering two-dimensional maps together with their geometrical data.

The Basic Map Display building block is an optional building block. If the Basic Map Display building block is available in a database, it must contain at least all road geometries that are available in the upper levels of the Routing building block.

---

**Note** If a World map is only represented by the Basic Map Display building block, it does not contain roads.

---

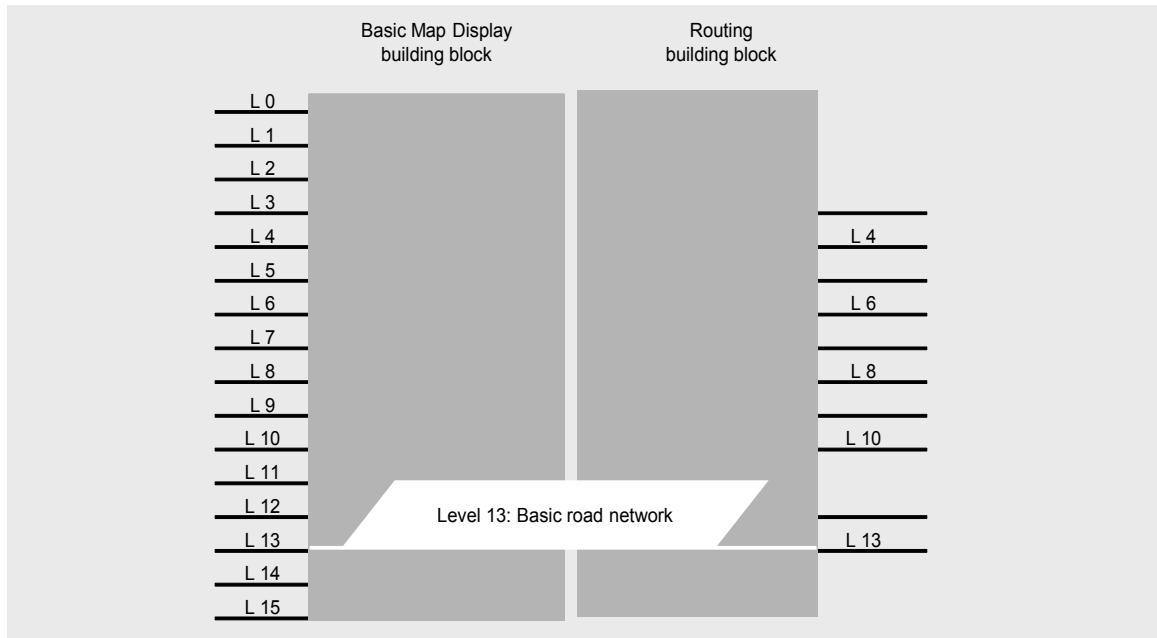
Instead of storing their own names for labeling maps, the features of the Basic Map Display building block make use of the name features stored in the Name building block via references. Names can thus be shared between building blocks and can be maintained separately in the Name building block.

For more information on name features, refer to Chapter 10 *Name Building Block* on page 163.

## 11.1 Building Block Structure and Content

The Basic Map Display building block consists of up to 16 levels. It contains the definitions of the map display features as well as building block-specific metadata. On level 13, the Basic Map Display building block shares data with the Routing building block, meaning that road geometry, which is also used for map display, is physically stored in the Routing building block.

*Figure 11-1 Shared level between Basic Map Display and Routing building blocks*



The Basic Map Display building block contains the following data.

*Table 11-1 Data in Basic Map Display building block*

Type of data	Description
Basic map display features	<p>Feature class definitions are provided for all basic map display features. The following main classes are available:</p> <ul style="list-style-type: none"> <li>- Point features</li> <li>- Line features</li> <li>- Area features</li> </ul> <p>For more information, refer to Section 11.2 <i>Overview of Basic Map Display Features</i> on page 207.</p>
Metadata	<p>The metadata of the Basic Map Display building block comprises the following:</p> <ul style="list-style-type: none"> <li>- BMD-specific metadata in overall metadata</li> <li>- BMD-specific metadata in level metadata</li> <li>- Feature class hierarchy.</li> </ul> <p>For more information, refer to Section <i>Metadata</i> on page 205.</p>

## Levels

The Basic Map Display building block may make use of up to 16 levels. The levels are optional and may be omitted. On level 13, the Basic Map Display building block shares data with the Routing building block. The levels of the highest resolution can contain detailed city maps.

Data is stored partly redundant on different levels with decreasing spatial density of content on higher levels, that is, on levels with lower numbers. The reduction of spatial density can be achieved through the following generalization or post-processing methods during compilation: Filtering, simplification, adjustment, and aggregation.

For more information, refer to *NDS – Compiler Interoperability Specification, 8.2 Use of Levels* on page 160.

## Levels and Map Scales

In order to allow flexible zooming, recommendations are provided for maximum scale levels for each level in the Basic Map Display building block. By assigning scale levels to building block levels, it is defined up to which scale level the building block elements are visible on the map. The information on recommended minimum and maximum map scale denominators is stored in the global metadata (see Section 5.1 *Level Metadata* on page 73).

Table 11-2 shows sample assignments between building block levels and maximum map scales.

*Table 11-2 Example: Assignment of max. map scale denominators to building block levels*

Tiling level	Tile width [km]	Recommended max. map scale denominator [1:X]
0	20000	80,000,000
1	10000	40,000,000
2	5000	20,000,000
3	2500	10,000.,00
4	1250	5,000,000
5	625	2,500,000
6	312.5	1,250,000
7	156.25	625,000
8	78.13	320,000
9	39.06	160,000
10	19.53	80,000
11	9.77	40,000
12	4.88	20,000
13	2.44	10,000
14	1.22	5,000
15	0.61	2,500

## Scale Sublevels

A map scale range can be further divided into smaller intervals by means of scale sublevels. These intervals enable an application to flexibly hide elements that are located on the same building block level at different zoom factors. The use of sublevels does not depend on the number of levels used. However, if the database is compiled with less than the maximum number of 16 levels, scale sublevels can be used to divide the map scales assigned to the remaining levels in order to facilitate graduated zooming.

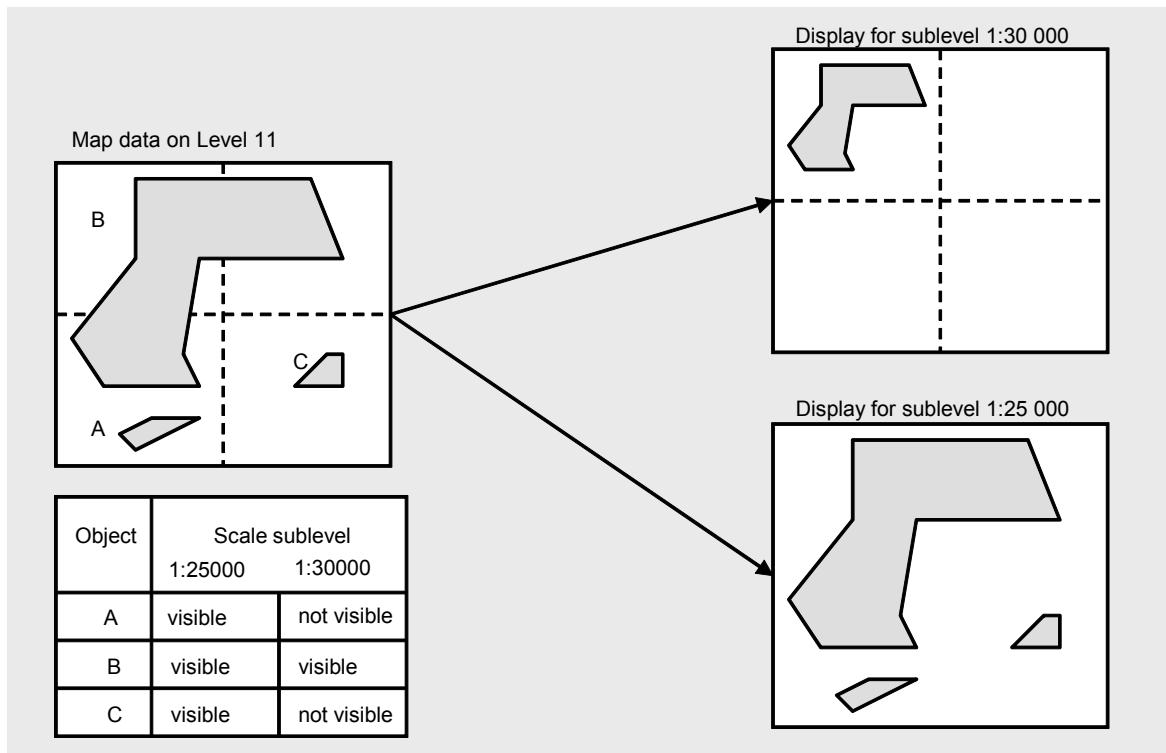
The definition of scale sublevels is customer-specific. The information on the number of scale sublevels is stored in the overall metadata (see Section 5.1 *Level Metadata* on page 73).

Scale sublevels are assigned to basic map display features by means of the subscale attribute. The sublevel values for a given level are between the minimum and maximum scale denominators assigned to that level (the minimum scale denominator is identical to the maximum scale denominator of the previous level). The subscale values must be in descending order. If a feature has the subscale attribute *i* in the range from 0 to <no. of scale sublevels>, the rendering engine should suppress this feature if the map scale is larger than sublevel [*i*-1]. Thus, a feature with sublevel index 0 will never be suppressed.

- 
- Example** The scale sublevel attribute indicates intermediate scales of 1:25000, 1:30000 and 1:35000 between the minimum scale of 1:20000 and the maximum scale of 1:40000. When drawing a map with scale 1:27000, the application will only read and display the features labeled with 1:30000 or 1:35000 to reduce the feature density in the rendered map.
- 

Figure 11-2 illustrates how elements are shown and hidden on scale sublevels.

Figure 11-2 Element display in scale sublevels



## Metadata

The following table gives an overview of the metadata of the Basic Map Display building block.

Table 11-3 Metadata of the Basic Map Display building block

Metadata	Description
Levels ( <code>hasBmdTiles</code> )	Defines how many and which levels are provided in the Basic Map Display building block  DataScript location: <code>nds.overall.metadata &gt; LevelMetadata</code>
Feature class hierarchy ( <code>bmdFeatureClassHierarchyTable</code> )	Defines the class hierarchy in the Basic Map Display building block  DataScript location: <code>nds.bmd.metadata</code> For more information, refer to Section 11.2 <i>Overview of Basic Map Display Features</i> on page 207.
Number of feature classes ( <code>numFeatureClasses</code> )	Defines the number of available feature classes for the Basic Map Display building block  DataScript location: <code>nds.overall.metadata &gt; bmdMetadata</code>

Metadata	Description
Feature class list (bmdFeatureClasses)	<p>Lists the available feature classes of the Basic Map Display building block for a specific update region</p> <p>The application can use this information for providing the corresponding styles. The list of feature classes can also be used for release notes and hints for testing the database content.</p> <p>DataScript location: <code>nds.overall.metadata &gt; bmdMetadata</code></p>
Polygon triangulation (polygonsAreTriangulated)	<p>Specifies whether the polygons of a specific update region are represented by triangle strips or triangle fans</p> <p>The application can use this information to determine whether it needs to triangulate the polygons depending on its GPU. This information is especially interesting for low-end systems with 2D GPU as they would slow down significantly when they get triangle strips.</p> <p>DataScript location: <code>nds.overall.metadata &gt; bmdMetadata</code></p>
Attribute types (bmdAttributeTypeAvailability)	<p>Specifies the available attribute types of the Basic Map Display building block for a specific update region</p> <p>With this information, an application can estimate the storage space needed for attributes. This information can also be used for release notes and hints for testing the database content.</p> <p>DataScript location: <code>nds.overall.metadata &gt; bmdMetadata</code></p>
Detailed city model (containsDetailedCityModel)	<p>Defines whether the Basic Map Display building block contains a detailed city model</p> <p>DataScript location: <code>nds.common &gt; BuildingBlockDetailedType</code>.</p>

## Dependencies to Name Building Block

For labeling maps, references are defined from the Basic Map Display building block to the Name building block.

Names of map display features are retrieved from the Name building block. They are usually positioned dynamically on the display.

For more information, refer to Section 10.8 *References of Named Object Features* on page 198.

## Dependencies to Routing Building Block

For displaying routes on a map, references are defined from the Routing building block to the Basic Map Display building block.

For more information, refer to Section 9.5 *References of Routing Features* on page 128.

## Dependencies to Traffic Information Building Block

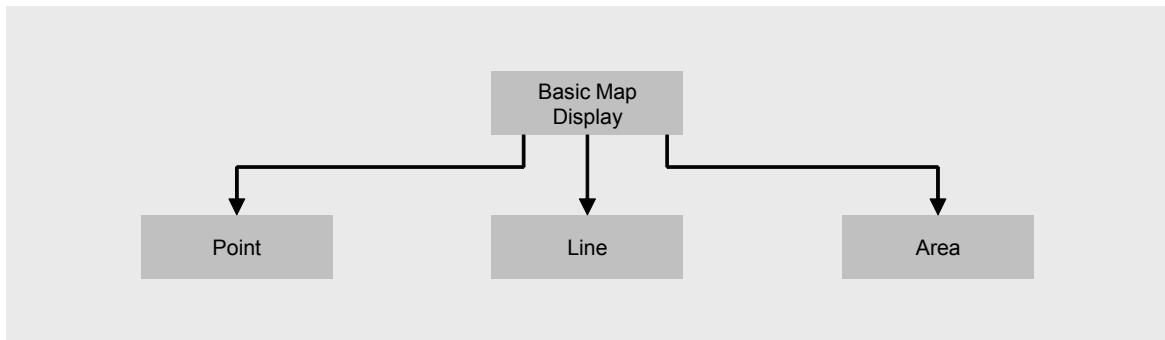
To enable the display of traffic-related information in maps, references are defined to all levels of data that are used in the Basic Map Display building block. Local lists of flexible attributes, organized in tiles on different levels, set up references between location codes (coming from messages with information about traffic events) and NDS features in the Basic Map Display building block.

For more information, refer to Section 12.2.5 *Linking TMC Information to NDS Features* on page 237.

## 11.2 Overview of Basic Map Display Features

The figure below gives an overview of the main feature classes available in the Basic Map Display building block. This is followed by conceptual information on the individual features. For more information on the feature classes used to represent the map features in the database as well as on their properties, refer to Sections 11.3 to 11.5 (pages 208 ff.).

*Figure 11-3 Overview of basic map display features*



The feature classes of the Basic Map Display building block form a hierarchy. This hierarchy is defined in the metadata of the building block (see Section *Metadata* on page 205) and can be extended by adding new classes and assigning them to a more general parent class. This enables existing applications to handle new classes.

### Points

Point features are used for rendering single points on a map. Subclasses are provided for the different types of point features. This includes city centers, city district centers, and others.

**Note** Points of Interest (POI) are not part of the Basic Map Display building block. This is due to the fact that POIs are subject to frequent changes. In addition, they have a number of attributes that are not relevant for map display.

For more information, refer to Chapter 13 *POI Building Block* on page 253.

## Lines

Line features are used for rendering lines on a map. Subclasses are provided for the different types of line features. This includes roads, ferry connections, water lines, railway lines, borderlines, and others.

## Areas

Area features are used for rendering area shapes on a map. Subclasses are provided for the different types of area features. This includes water areas, land use and others.

For more information, refer to [Section 11.5 Area Features on page 211](#) and [NDS – Compiler Interoperability Specification, 8.1.3 Area Features on page 158](#).

## 11.3 Point Features

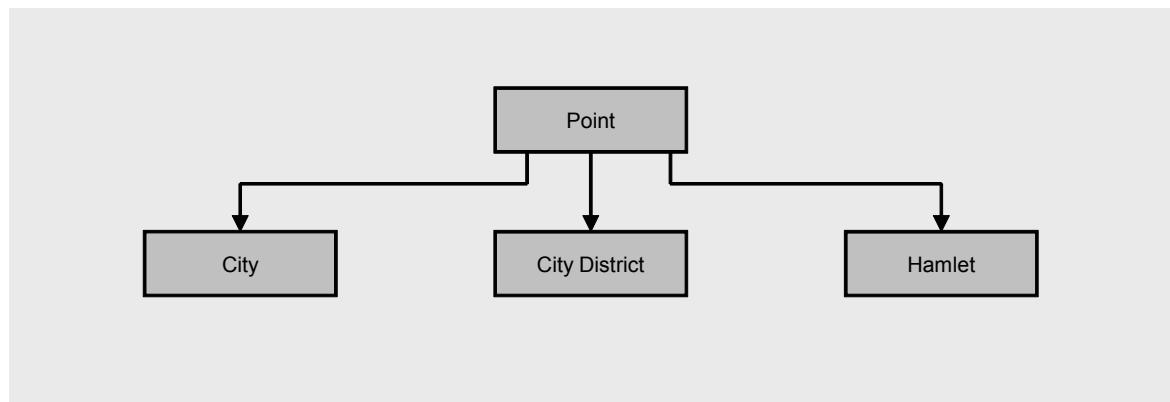
The following list gives examples of subclasses, that are provided for basic map display features of the point feature class:

- City and its center
- City district and its center
- Hamlet
- Mountain peak

More point feature classes can be defined as enhancements in future versions of the standard. For a complete list of point features defined in NDS, refer to [DataScript nds.bmd.common](#).

The feature classes of the Basic Map Display building block form a hierarchy. This hierarchy is defined in the metadata of the building block and may be extended by adding new classes and assigning them to a more general parent class. This enables existing applications to handle new classes. Figure 11-4 gives an example.

*Figure 11-4 Point feature classes, example*



## Properties of Point Features

The Point feature classes are characterized by the following fixed attributes (future subclasses may provide different attributes).

*Table 11-4 Fixed attributes of point features*

Attribute	Description
coord	Coordinates of the feature's shape points
scaleSublevel	Specifies the recommended scale sublevel for rendering the feature For more information, refer to Section <i>Scale Sublevels</i> on page 204.

Point features are not visible if rendered as a single point or pixel on a map. Therefore, they are usually represented by icons. For more information on icons, refer to Chapter 20 *Icons* on page 355.

The following flexible attributes are available for point features of the subclass *City*. They are assigned to individual features if required:

- POPULATION (DataScript location: `nds.common.flexibleattributes`): This attribute is used to specify the population for a place.
- IS\_COUNTRY\_CAPITAL: This attribute indicates whether the city is a capital.

## 11.4 Line Features

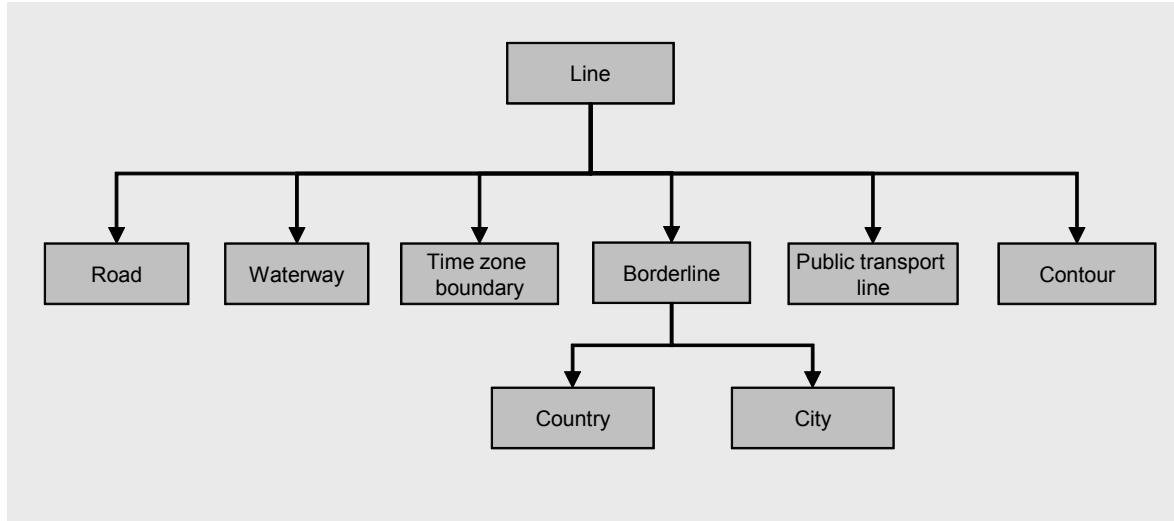
To represent the manifold number of line-shaped map elements, a large number of subclasses is provided for basic map display features of the line feature class. For example:

- Roads (includes ferries)
- Waterways
- Time zone boundaries
- Borderlines
- Public transport lines, for example, railway lines or subways
- Contour lines

More line feature classes can be defined as enhancements in future versions of the standard. The feature classes of the Basic Map Display building block form a hierarchy. This hierarchy is defined in the metadata of the building block (see Section *Metadata* on page 205) and may be extended by adding new classes and assigning them to a more general parent class. This enables existing applications to handle new classes.

Figure 11-5 gives an example.

Figure 11-5 Line feature classes, example



## Properties of Line Features

The line feature classes are characterized by the following fixed attributes (future subclasses may provide different attributes).

Table 11-5 Fixed attributes of line features

Attribute	Description
coord	Coordinates of the feature's shape points
scaleSublevel	Specifies the recommended scale sublevel for rendering the feature For more information, refer to Section <i>Scale Sublevels</i> on page 204.

On level 13, the Basic Map Display building block does not contain lines representing roads. The corresponding information is stored in the Routing building block only. For roads stored on all other levels of the Basic Map Display building block, additional road geometry line attributes are provided.

For more information on road attributes, refer to Section *9.3 Road Geometry Line Feature Class* on page 122.

The following **flexible attributes** are available for line features of the road subclass (DataScript location of attributes: `nds.common.flexibleattributes`). In contrast to the fixed attributes above, they are only applied to individual features of the line feature class, if required. Examples:

- `DRAWING_ORDER`
- `ROAD_Z_LEVEL`

## 11.5 Area Features

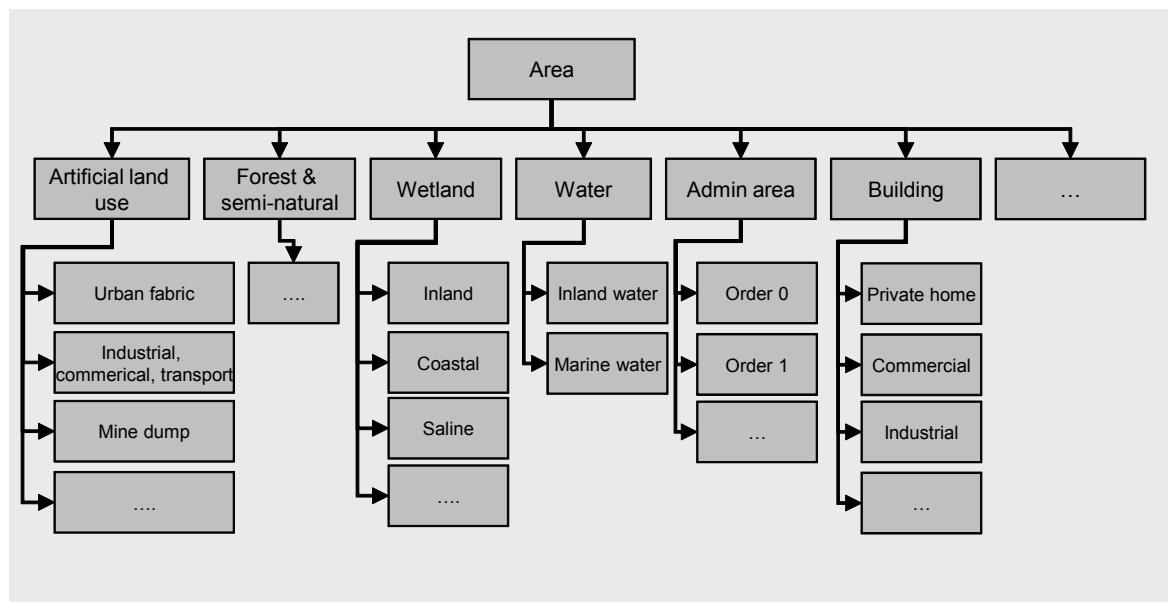
Areas that represent regions with different characteristics can be classified accordingly. The following area types are provided:

- Water, for example, lakes or the sea
- Land use, for example, built-up areas or woodland
- Buildings, for example, hospitals or industrial buildings

More area feature classes can be defined as enhancements in future versions of the standard. The feature classes of the Basic Map Display building block form a hierarchy. This hierarchy is defined in the building block metadata (see Section *Metadata* on page 205) and may be extended by adding new classes and assigning them to a more general parent class. This enables existing applications to handle new classes.

Figure 11-6 gives an example.

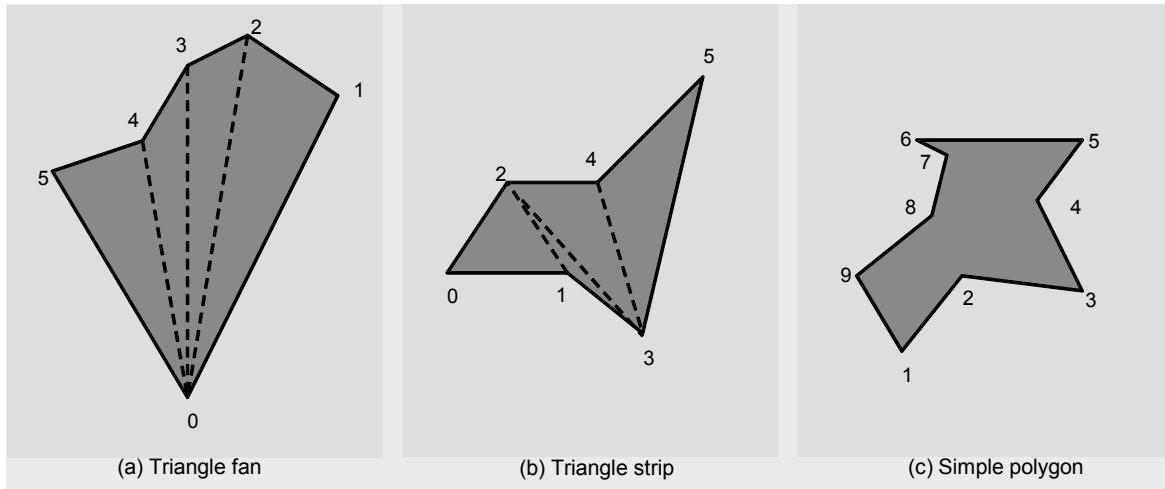
*Figure 11-6 Area feature classes, example*



Area features can have complex shapes. An area shape can be approximated by a polygon, that is, a sequence of points joined by straight line segments with an additional segment joining the last point to the first one. Area definitions include polygons with self-intersections and non-convex polygons, which are usually problematic for drawing and filling algorithms.

To handle all of these special cases, and to simplify area rendering for the application, NDS introduces the following polygon types: Triangle fan, triangle strip, and simple polygon (see Figure 11-7).

Figure 11-7 Polygon types



For effective rendering by 3D engines, polygons are segmented into polygon primitives. Triangulated polygons are stored as triangle fan and triangle strip. In case there is no 3D engine, simple polygons are used for area features. Simple polygons in NDS are connected and have no self-intersections or holes to avoid problems for drawing and filling algorithms.

## Properties of Area Features

The area feature classes are characterized by the following fixed attributes (future subclasses may provide different attributes).

Table 11-6 Fixed attributes of area features

Attribute	Description
coord	Describes the geometry of the feature
scaleSublevel	Specifies the recommended scale sublevel for rendering the feature For more information, refer to Section <i>Scale Sublevels</i> on page 204.

If an area covers several tiles, tile patterns may be provided. For more information, refer to Section 11.8 *Tile Patterns* on page 214.

## Multipolygons

An area may consist of several polygons of the same feature class. This may be useful, because drawing several areas together with the same style is faster than drawing each area separately.

An area is encoded as a list of vertices with certain properties. The polygon flag indicates the end of a polygon. Thus, the next vertex value in the list is the start vertex for the following polygon. Since polygons must have more than two vertices, the polygon flags for the first two vertices are not needed to indicate the end of the polygon. As a physical optimization, NDS uses the combination of the first two polygon flags to indicate the polygon type.

DataScript location: `nds .bmd .areas > AreaCoordXYDiff > polygonFlag`

For more information, refer to *NDS – Compiler Interoperability Specification, 8.1.3 Area Features* on page 158.

### Pseudo Edges

For encoding triangle strips, triangle fans and connecting polygons on different places of a tile, or for polygons, which are cut by tile borders, NDS uses the pseudo edge flag. The flag indicates that the edge shall not be drawn, either because it is inside the polygon, or connects two remote polygons.

DataScript location: `nds .bmd .areas > AreaCoordXYDiff > isPseudo`

For more information, refer to *NDS – Compiler Interoperability Specification, 8.4 Relation between Source Features and Compiled Features* on page 163

## 11.6 Icons

Features of the Basic Map Display building block can be represented by icons, for example, if they would not be visible when rendered as a single point or pixel. Icons can be used to display, for example, point features and road numbers.

---

<b>Note</b>	Basic map display icons may be associated with line features in such a way that it is possible to display a rotated icon image. The icon reference and the angle value for the rotation are stored in the flexible attribute <code>rotatedIconRef</code> assigned to the line feature.
-------------	--

---

For more information on icons, refer to Chapter 20 *Icons* on page 355.

## 11.7 Drawing Styles

The drawing style of a map display feature, which includes attributes like color, border style, or font size, is application-specific. It is, therefore, not predefined in the database. Usually, the drawing style not only depends on the class a feature belongs to, but also on additional attributes. For instance, road features can be drawn as lines of different color and thickness, depending on the functional road class. Cities in overview maps can be drawn as circles or squares of different sizes, depending on population.

As a dedicated drawing style attribute would constrain the application, NDS provides no drawing style attribute. Therefore, additional attributes can be specified for each feature class, and the application can select a drawing style based on the complete set of attribute values.

---

**Example** An application wants to highlight road features with speed limits. This is not possible with a drawing style attribute for road features based on functional road classes only.

---

**Note** For rendering road number strings, the drawing style may be influenced by specifying a predefined font in the icon table (DataScript location: `nds.overall.icondata > IconTable`). Font-related information required for rendering text is stored in the **FontTable** (DataScript location: `nds.overall.metadata`). For detailed information on icons, refer to Section 20 *Icons* on page 355.

---

## 11.8 Tile Patterns

Tile patterns can be defined for map elements that extend over the whole area of several tiles. The provision of such patterns can significantly save storage space because identical tiles need to be stored only once. This is useful for larger areas with identical map content, such as large lakes, oceans, forests, or deserts.

Patterns are assigned to tiles via reference tables (DataScript location: `nds.bmd.main > bmdTilePatternTable`). In Figure 11-8, the same pattern can be used for all tiles which are completely covered by the polygon (represented by the gray layer), for example, tile 18 and tile 109. For all other tiles intersected by the polygon, the corresponding area features have to be defined (for example, tile 120, tile 22), as these tiles differ in their content.

Figure 11-8 Tile pattern, example

42	43	46	47	58	59	62	63	106	107	110	111	122	123	126	127
40	41	44	45	56	57	60	61	104	105	108	109	120	121	124	125
34	35	38	39	50	51	54	55	98	99	102	103	114	115	118	119
32	33	36	37	48	49	52	53	96	97	100	101	112	113	116	117
10	11	14	15	26	27	30	31	74	75	78	79	90	91	94	95
8	9	12	13	24	25	28	29	72	73	76	77	88	89	92	93
2	3	6	7	18	19	22	23	66	67	70	71	82	83	86	87
0	1	4	5	16	17	20	21	64	65	68	69	80	81	84	85

## 11.9 Drawing Order

For map rendering, the drawing order is important. In general, area features are drawn first, line features are drawn on top of the area features, and point features are drawn last.

Within the features of the same class, the general drawing order is determined by the physical order of the data. This includes ordering of features according to their subclasses. Land use or waterway areas, for example, are drawn before surface use areas like buildings or sports grounds.

Deviations from this general drawing order may be required for incremental updates, as new features are always added to the end of the feature lists. In this case, the following information shall be evaluated in the given sequence to determine the drawing order:

1. the flexible attribute ROAD\_Z\_LEVEL (DataScript location:  
`nds.common.flexibleattributes`; this attribute is used to cluster features in height levels)
2. the flexible attribute DRAWING\_ORDER (DataScript location:  
`nds.common.flexibleattributes`).

Both attributes override the general drawing order.

Moreover, features of the same class can be sorted by increasing importance to support on-the-fly map generalization from the same map level. On-the-fly generalization is also supported by assigning scale sublevels (see Section *Scale Sublevels* on page 204).

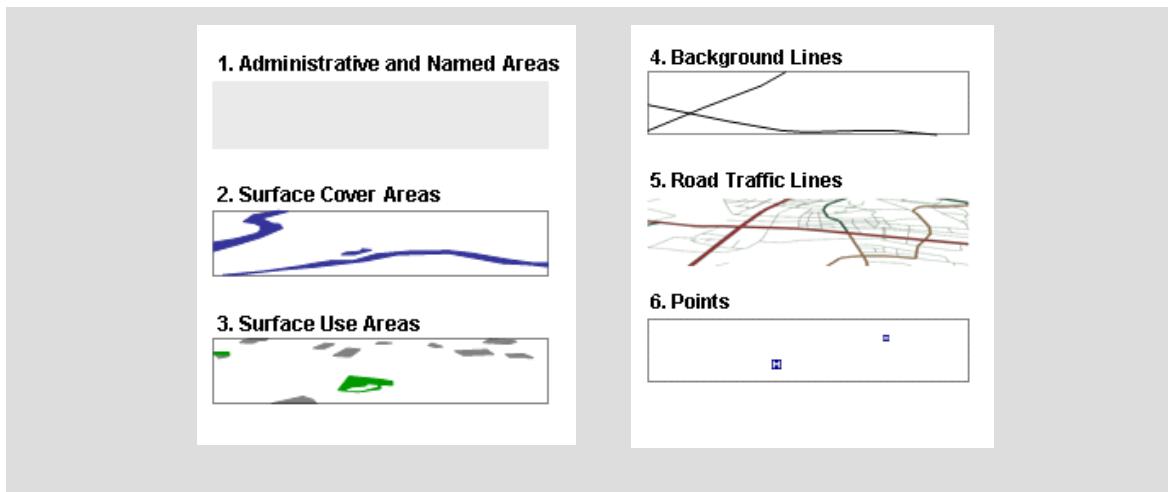
For more information, refer to *NDS – Compiler Interoperability Specification*, 8.3 Drawing Order on page 162.

Figure 11-9 and Figure 11-10 illustrate the drawing order of features.

Figure 11-9 Drawing order, example 1



Figure 11-10 Drawing order, example 2



## 11.10 Using Z Levels for Map Rendering

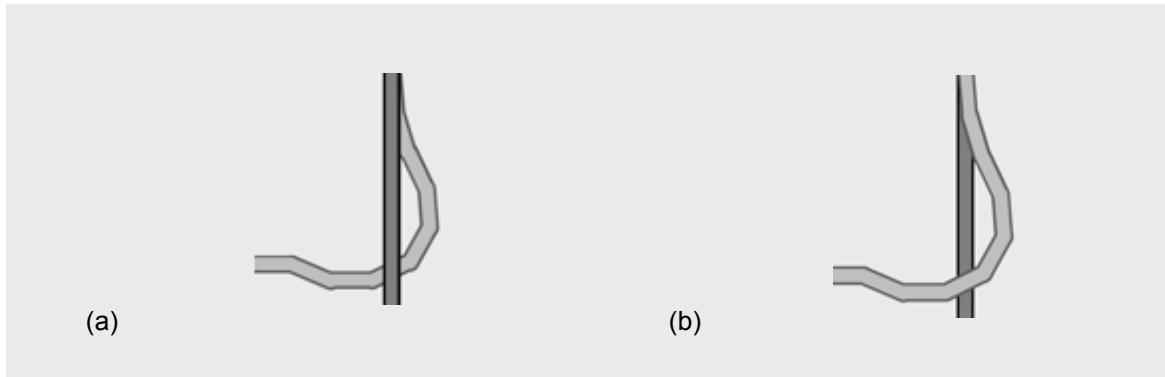
The Z level concept of Navigation Data Standard provides more flexibility in rendering ramps, tunnels (underpasses), and bridges. Databases without Z level information can only use the drawing order information to ensure that more important roads are rendered above less important roads, and to avoid that important roads are interrupted.

For more information on map rendering with Z levels, refer to *NDS – Compiler Interoperability Specification*, 8.5 Using Z Level Attributes for Map Rendering on page 164.

### 11.10.1 Advantages of Z Levels

The following figures show examples for map rendering with and without Z levels. The figures show a ramp that leads across a main road and then merges into the main road.

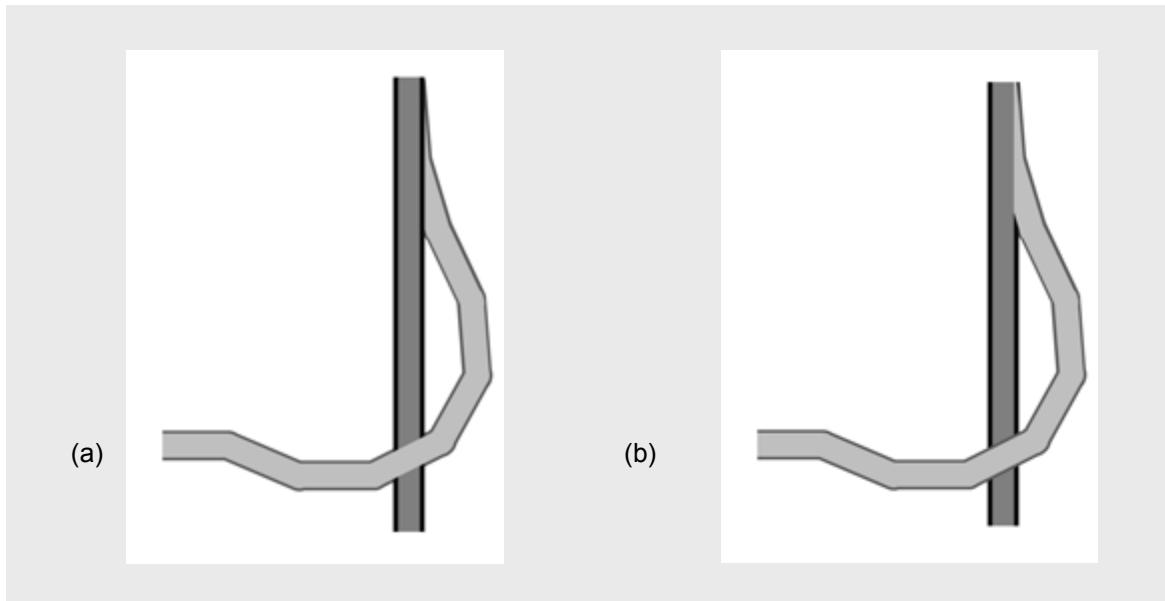
Figure 11-11 illustrates map rendering without Z levels.

*Figure 11-11 Map rendering without Z levels*

In Figure 11-11 (a), the more important road (dark grey) is rendered last and lies completely above the curved ramp (light grey). The ramp seems to pass under the main road. However, this does not reflect the real topology, as the ramp crosses the more important road.

In Figure 11-11 (b), the ramp including the merging section at the top of the picture is drawn above the main road. This is misleading because usually the more important road has to be rendered above the less important road. Therefore, the map rendering shown in Figure 11-11 (b) would not be acceptable for customers of navigation databases.

Figure 11-12 depicts two possibilities of map rendering with Z levels.

*Figure 11-12 Map rendering with Z levels*

In both figures, the ramp crosses the main road correctly. In Figure 11-12 (a), the merging section is not drawn. The ramp lies under the main road. In Figure 11-12 (b), also the merging section is drawn correctly.

## 11.10.2 Data Structures for Z Level Information

Navigation Data Standard implements the Z level by means of a flexible attribute called `ROAD_Z_LEVEL` (DataScript location: `nds.common.flexibleattributes`). A validity range is assigned to the Z level attribute.

The following information is specified in the attribute:

- `roadHeightLevel`: Height information for height points
- `DETACHED_FROM_TERRAIN`: Indicates whether a road is detached from the terrain or follows the terrain

The Z level attribute can be changed for every coordinate for the following NDS features:

- For level 13: Base links, road geometry lines, and intersections (Routing building block)
- For all other levels: Lines (Basic Map Display building block)

For more information on the Z level attribute, refer to the *NDS – Physical Model Description*.

Z level values are assigned to a range of shape points of a route link. Between two Z level ranges, there can be a gap of several shape points without Z level. This gap defines the ramp (3D) or the overlap section (2D). If no Z level attribute is set for a link, the Z level value for the complete link is 0.

Z level values may be positive, negative, and zero numbers (signed integer values). A Z level value equal to zero defines the ground level. Roads with a Z level of 1 and higher are detached from the ground. Negative Z level values are necessary for roads that underpass a road on ground level (for example, an undercrossing).

The following figures illustrate the concept of Z level ranges.

*Figure 11-13 Example 1: Link without Z level attribute*

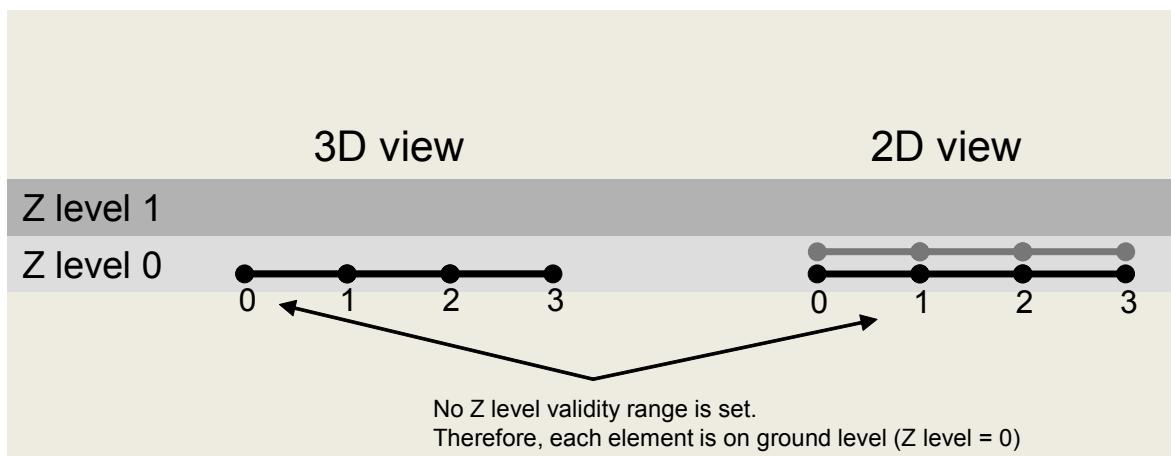


Figure 11-14 Example 2: Link with Z level = 1 for the complete range

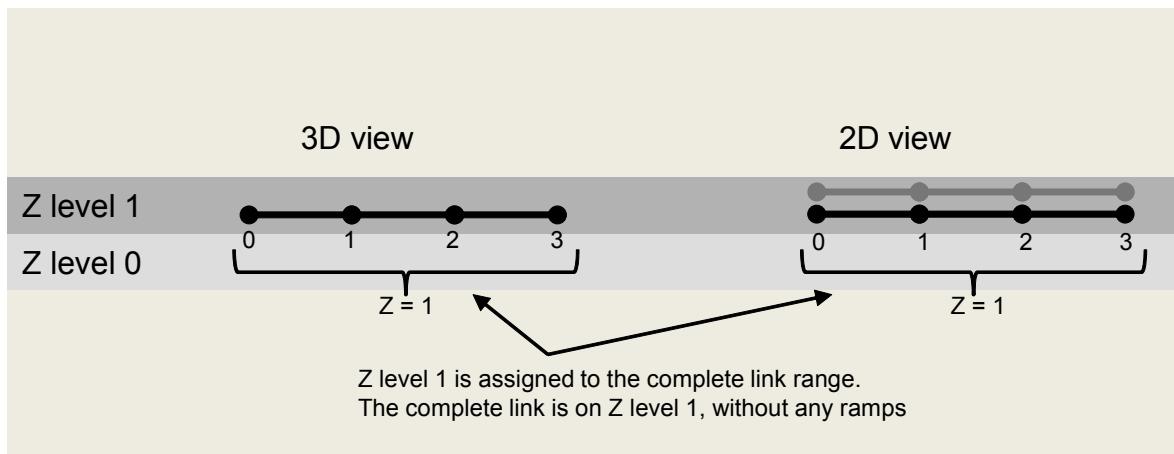


Figure 11-15 Example 3a: Link with constant Z level

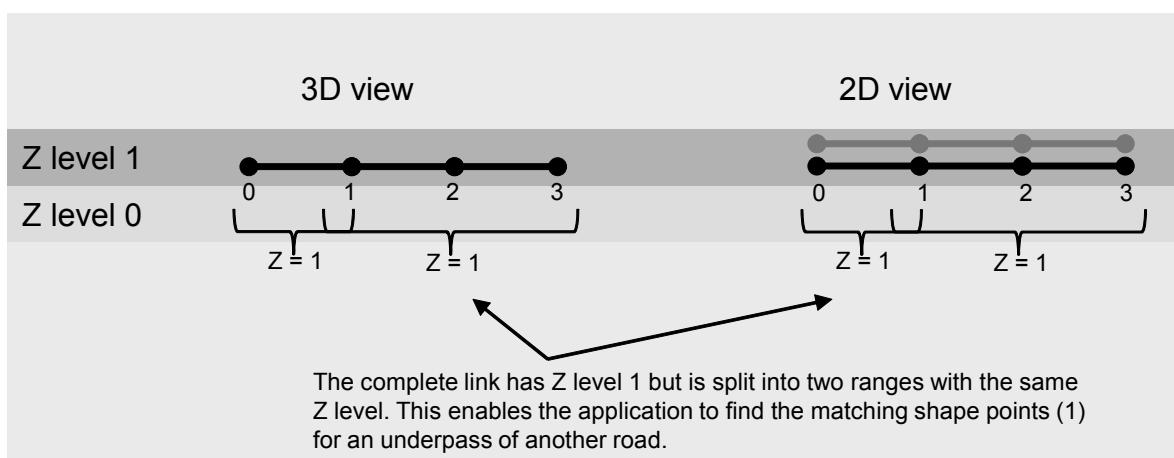


Figure 11-16 Example 3b: Link with constant Z level

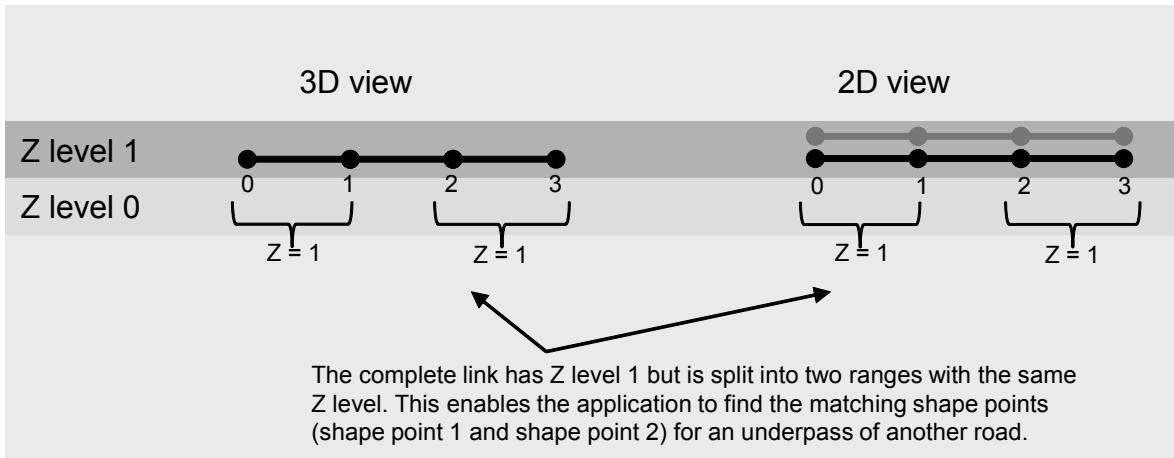


Figure 11-17 Example 4: Complete link is a ramp

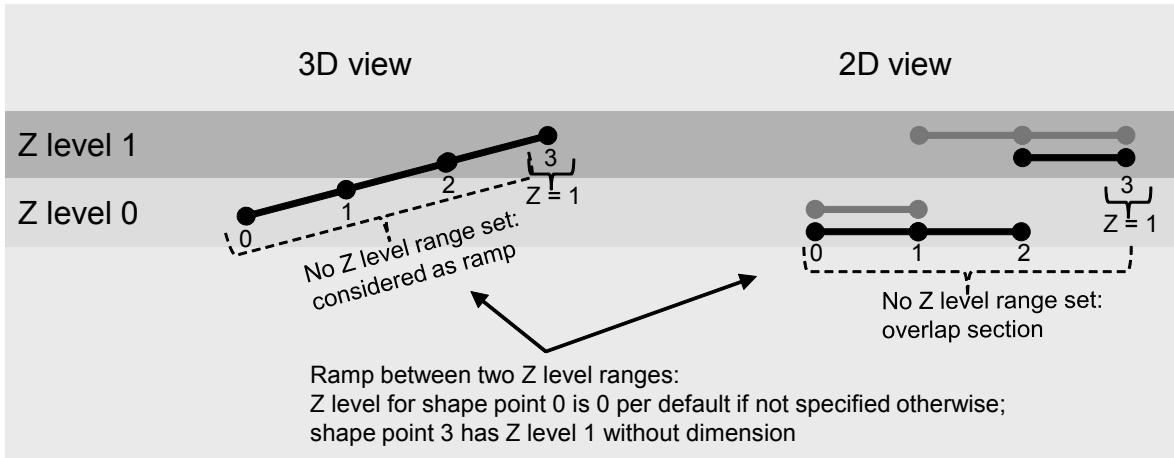


Figure 11-18 Example 5: Link has a start and an end Z level range

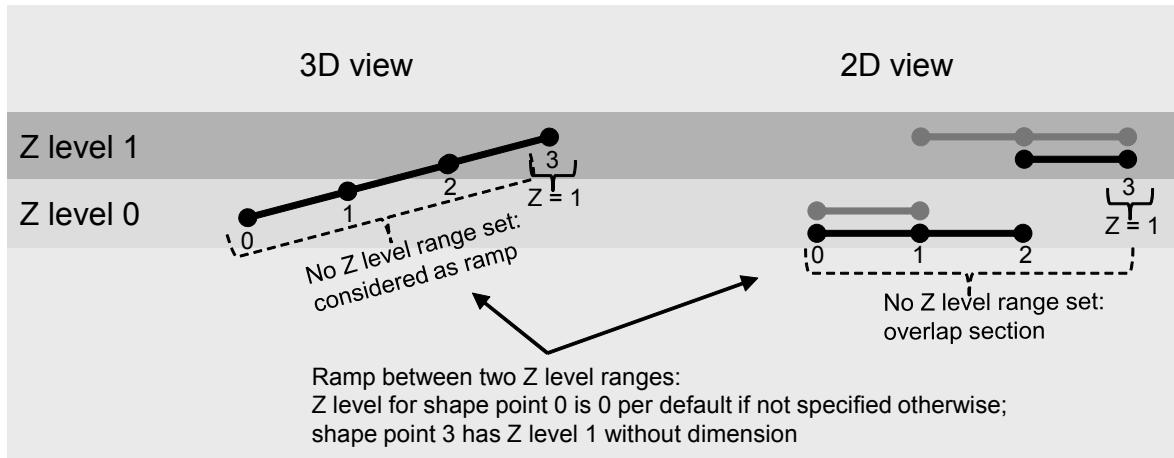


Figure 11-19 Example 6: Link ends with a ramp

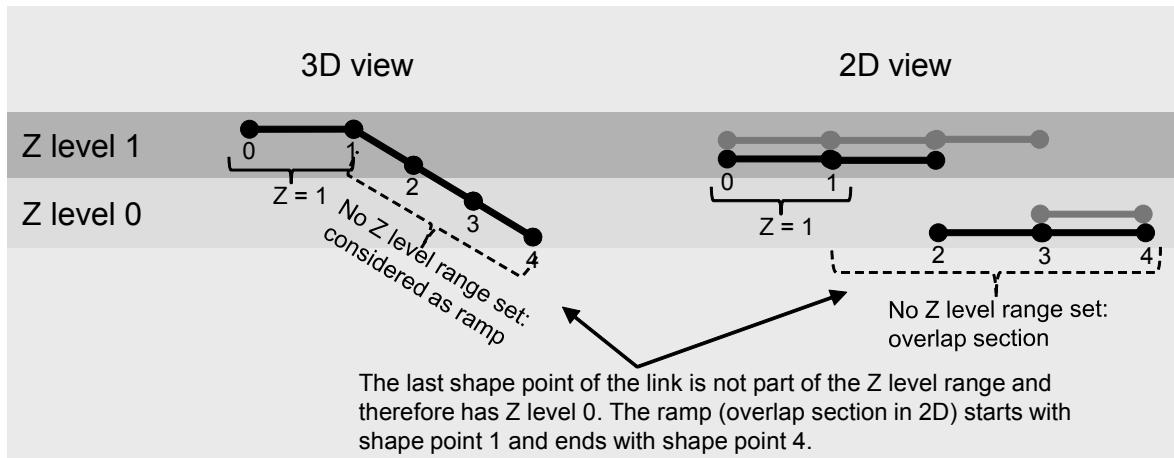
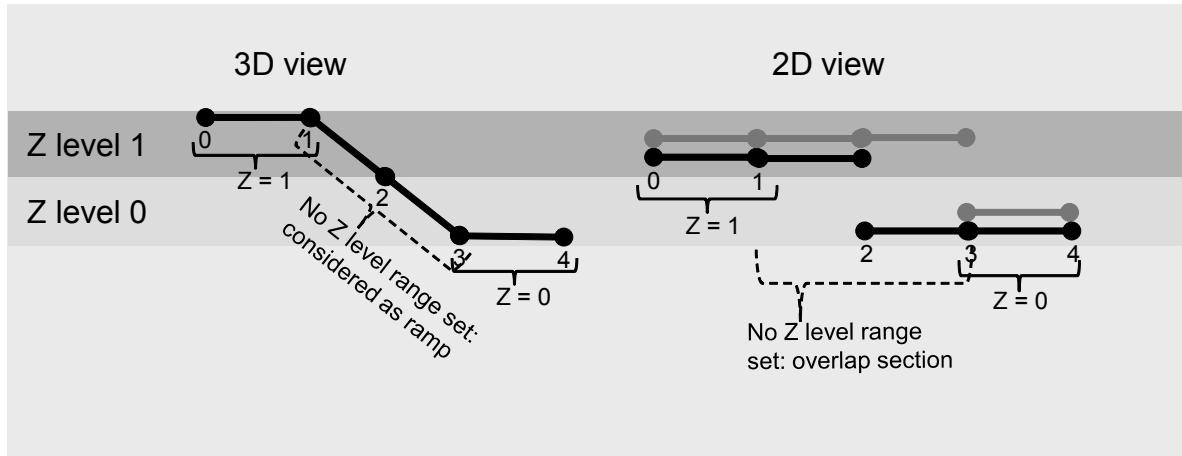


Figure 11-20 Example 7: Link with defined start and end Z level range and ramp in between



NDS does not allow to have nested shape point ranges, for example, 0..3 and 1..2, or 0..2 and 1..2; touching ranges (for example, 0..1 and 1..2) are allowed for equal Z levels and are used for marking the touching point as underpasses or overpasses.

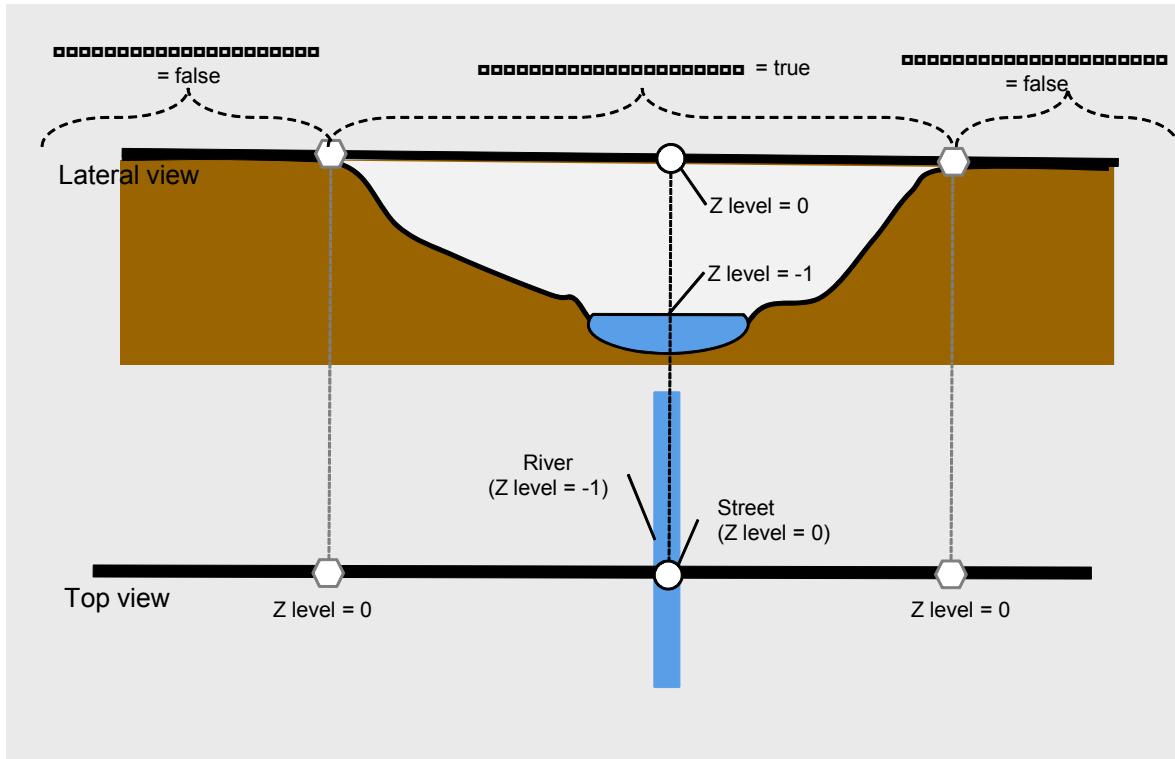
Z level ranges of different values can have shape points without Z level in between. The gap is necessary to render the ramp (3D) or for finding a suitable pair of shape points for the overlap section (2D).

For more information, refer to *NDS – Compiler Interoperability Specification, 8.5 Using Z Level Attributes for Map Rendering* on page 164.

### Detached from Terrain Attribute

In addition to the height information given in the Z level attribute, it is possible to define whether a road is detached from the terrain or follows the terrain contour. This is depicted in Figure 11-21.

Figure 11-21 Road across a river rendered by means of detached from terrain attribute



Although the river bank terrain is lower than the road, the Z level attribute is set to 0, and only changes from 0 to  $-1$  when the road crosses the river. This makes it possible to draw the river below the road. The DETACHED\_FROM\_TERRAIN attribute is used to ensure that the road stays on the same height when it crosses the river bank terrain. A map viewer application uses this information to draw an interpolated, straight elevation between the two points where the road crosses the rivervalley.

If the road needs to rise, for example, because the river is on the same level as the banks, the height information property can be used.

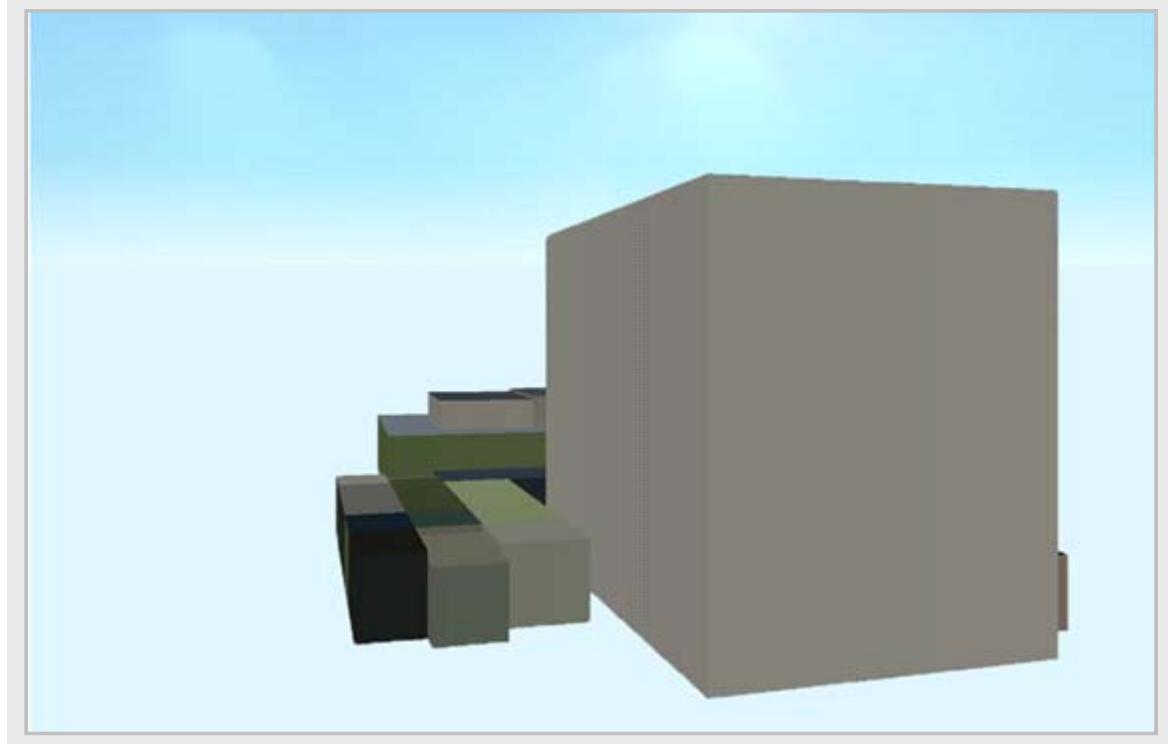
## 11.11 2.5D City Models

NDS supports the 2.5D representation for buildings, that is a 3D looking visualization of city buildings generated from elevated building footprints represented as polygons with height attributes. These models are called 2.5D city models. Figure 11-22 shows an example.

The recommended basic map display level for 2.5D city models is level 13. NDS uses area features of type **Building**, so-called *footprints*, as the basis for 2.5D city models. The application can decide whether to display the 2D footprints only, or the 2.5D city models.

- 
- Note** The input data required for 2.5D city models is not available for all parts of the world. The application has to be able to deal with mixed data.
- 

*Figure 11-22 2.5D city model, example*



### 11.11.1 Flexible Attributes for 2.5D City Models

NDS uses the underlying area features of type “building” as a footprint for the 2.5D representation. The third dimension is stored by means of flexible height attributes. For the illustration of visual properties, such as roof and wall colors, flexible color attributes are provided.

Table 11-7 lists the flexible attributes for 2.5D city models (DataScript location: `nds.common.flexibleattributes`).

*Table 11-7 Flexible attributes for 2.5D city models*

Flexible attribute	Description
<code>GROUND_HEIGHT</code>	Height of the building or building part above ground level The default value is <code>0</code> , and in this case it is not necessary to store the value. Buildings with a <code>GROUND_HEIGHT</code> value larger than <code>0</code> are located above the ground, that is, on top of another building. The value is always positive.

Flexible attribute	Description
BUILDING_HEIGHT	<p>Height of the building or building part above GROUND_HEIGHT</p> <p>The BUILDING_HEIGHT value is assigned to each area feature of type „building“ that is to be represented as a 2.5D city model. The value is always positive.</p>
WALL_COLOR, wallColorIdRef	<p>Color of all walls of a building or building part</p> <p>The wall color is specified as 24-bit RGB value. The colors are stored in a color table. The value of the attribute WALL_COLOR has a reference to the ID of the color in the color table.</p>
	<p><b>Note</b> The color table is stored in the product database (see Table 4-3).</p>
ROOF_COLOR, roofColorIdRef	<p>Roof color for 2.5D city models</p> <p>The roof color is specified as 24-bit RGB value. The colors are stored in a color table. The value of the attribute ROOF_COLOR has a reference to the ID of the color in the color table.</p>
BASE_ELEVATION	<p>Height of a building or building part above the geoid (EGM96).</p> <p>The BASE_ELEVATION attribute describes the ground level height. The GROUND_HEIGHT and ELEVATION_DELTA attributes, therefore, are always defined relative to BASE_ELEVATION.</p> <p>The elevation value can be positive or negative. A negative value is, for example, used for the Dead Sea area. This attribute allows the rendering of 2.5D city map objects together with a digital terrain model. The quality of the combined display is subject to the quality and resolution of the digital terrain modell.</p> <p>For more information on the Digital Terrain Model, refer to Chapter 15 <i>Digital Terrain Model Building Block</i> on page 289.</p>

Figure 11-23 shows an example of a composed building in 2.5D representation. Table 11-8 gives the respective attribute values for the two buildings.

Figure 11-23 Composed 2.5D representation of a building

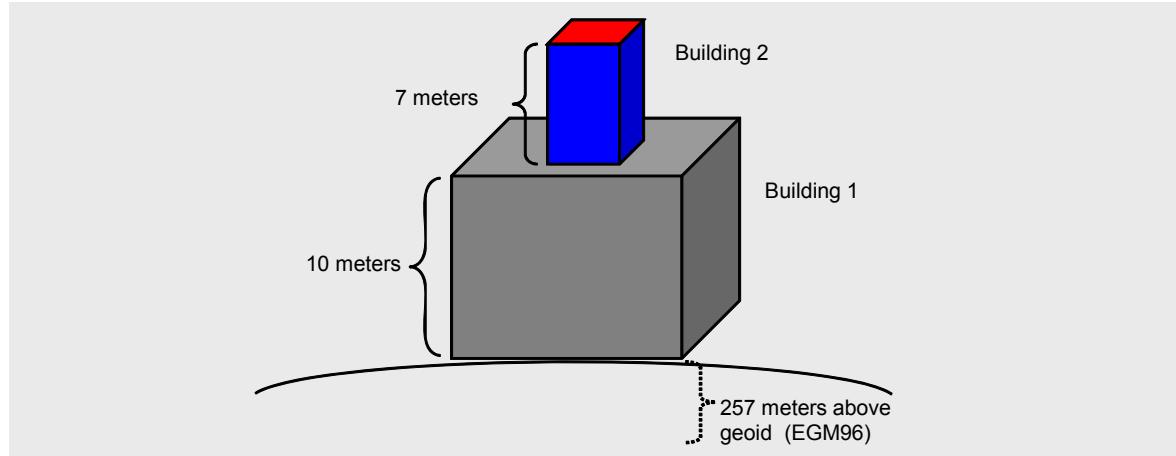


Table 11-8 Attribute values for the buildings in Figure 8-26

Flexible attribute	Values for building 1	Values for building 2
GROUND_HEIGHT	Default (0 meters)	10 meters
BUILDING_HEIGHT	10 meters	7 meters
WALL_COLOR	#0f0f0f	#0000ff
ROOF_COLOR	#0f0f0f	#ff0000
BASE_ELEVATION (if provided)	257 meters	257 meters

## 11.12 3D Objects Representing Building Footprints

BMD areas representing 2D footprints may have an additional flexible attribute **HAS\_3D\_OBJECT**. This flexible attribute indicates that a 3D model of a building is available and can replace a 2D building footprint or 2.5D city model (see 11.11 2.5D City Models on page 223).

If the flexible attribute **HAS\_3D\_OBJECT** is available in the raw data, the application can decide whether it displays the 2D building footprint, the 2.5D city model, or the 3D building model.

## 11.13 Labeling Hints

Labeling hints facilitate placing names for features in a displayed map. For area features, for example, it is important to curve the label, and to increase character spacing in order to make it possible to discern different area names on the map. For line features, it is useful to adapt the label to the curve.

This section describes how NDS supports label placement for line and area features. The following use cases are supported:

- Labeling line features with curved text
- Labeling line features with tangential text (can be implemented by means of curved labeling hints)
- Labeling line features with horizontal text
- Labeling area features with curved text
- Labeling area features with horizontal text

For more information, refer to *NDS – Compiler Interoperability Specification, 8.6 Labeling Hints* on page 175

### 11.13.1 Labeling of Line Features

There are different technical approaches for labeling line features. By now, NDS supports curved labeling for line features, which can also be used for tangential labels. Other approaches will be implemented in a later version of NDS.

#### Labeling Line Features with Curved Text

To label line features with curved text, the flexible attribute `LABELING_HINT` is assigned to features in Basic Map Display, or to base links and road geometry lines in Routing. The Boolean `hasCurvedLabelingLink` indicates that the line feature has a curved labeling hint.

The flexible attribute `ANTI_LABELING_HINT` can be used to specify points along a line feature or a road geometry line where no labels must be placed.

### 11.13.2 Labeling of Area Features

There are two different technical approaches for labeling area features, one using splines, and one using circles. By now, NDS supports horizontal labeling for area features by means of circles. The spline approach will be implemented in a later version of NDS. The database supplier may then decide on one approach or provide data for both approaches in the database. If the database contains data for both approaches, the compiler can select the labeling method based on the shape of the area feature.

#### Horizontal Text Labels based on Circles

For this approach, labeling points are provided for area features. An application can use these points to quickly determine the label position for the area feature. Labeling points are defined by the following data:

- Position (x,y)
- Maximum radius of a circle at (x,y) lying completely inside the polygon describing the area feature
- Highest level of visibility, meaning the least detailed map level in which the labeling point is contained

The best position for the label is determined by the largest incircle of the polygon. During compilation, further possible positions for labels in the polygon are determined. The radius of the resulting circles is defined by the minimal distance to the polygon's outline. Thus, the radii of different labeling points of an area feature provide a prioritization of the labeling points within one area feature. Depending on the spatial extension of the rendered text, the application can also decide on the appropriate labeling point for the rendered text.

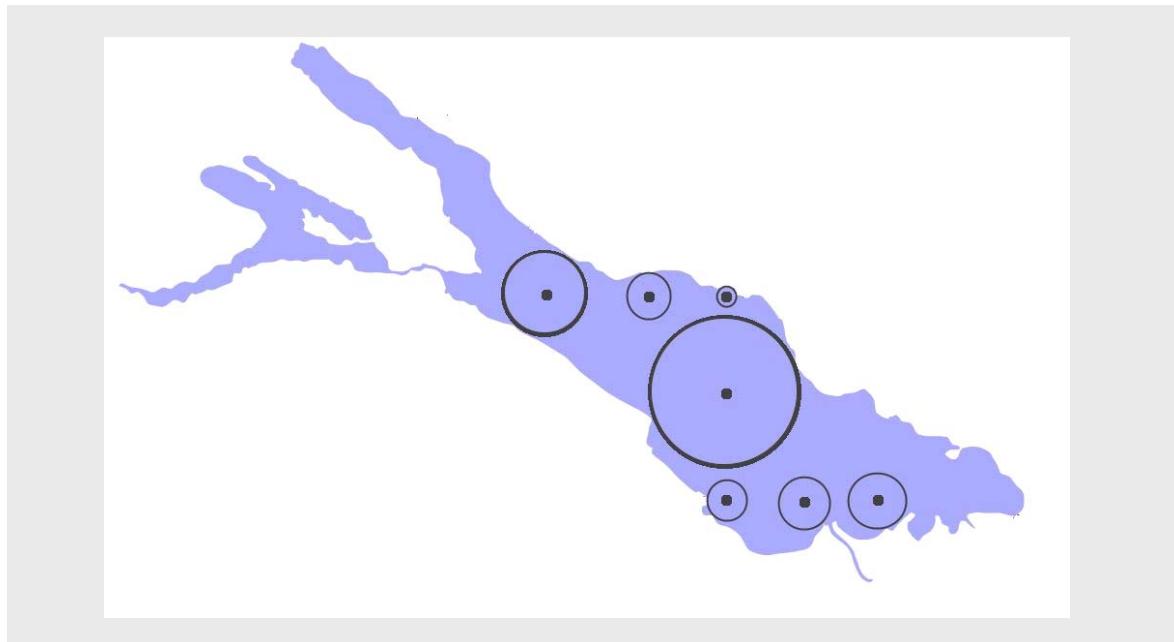
In order to provide a criterion for selecting the most suitable labeling point of an area feature, even if their radii match, the labeling points additionally contain the least detailed map level in which they are contained. This information makes it possible for detailed map levels to select the labeling points which can be used longest when zooming out to less detailed views. This reduces the side-effect of jumping labels while zooming out to a minimum.

The sequence in which labeling points are stored in the database is defined by the following criteria:

- Highest level of visibility, that is labeling points which are visible on the least detailed layers (lowest layer index) are stored first
- Radius; the labeling point with the largest radius is stored first

The displayed label depends on the zoom level. The application decides at runtime which label position is applicable by matching the available radii with the length of the name for the area feature. The definition of a threshold makes sure that circles are ignored if they are too small for a label. The circle line defined by the position (x,y) and the radius is not necessarily contained in the corresponding basic map display tile. Furthermore, the circle lines of different labeling points may intersect.

Figure 11-24 Labeling area features with horizontal text



## 12 Traffic Information Building Block

The Traffic Information (TI) building block enables applications to process traffic information data of various standards. Currently, Navigation Data Standard (NDS) supports the following protocol definitions:

- Traffic Message Channel (TMC)
  - Transport Protocol Experts Group (TPEG)
- 

**Note** Other traffic information protocols, such as Vehicle Information and Communication System (VICS), may be added in future versions of the standard. Until then, they may be added to an NDS-compliant product as a proprietary add-on (see Section 3.5 *Extending the Navigation Data Standard* on page 55).

To indicate that VICS information is available for a specific route or intersection, the flexible attribute `HAS_VICS` (`nds.common.flexibleattributes`) may be assigned to links and intersections.

---

The traffic information protocol definitions make it possible to deliver traffic and travel information in a language-independent binary encoding. Applications can look up the codes in a database and transform them into text and/or speech information in the language that has been selected by the user. References to other building blocks, such as Basic Map Display or Routing, allow applications to use the traffic information data for dynamic route calculation and map display.

---

**Note** Traffic information data can be transmitted by several means, such as Frequency Modulation (FM), Digital Audio Broadcasting (DAB), Internet, etc.. The transmission system is irrelevant for the database format specification. NDS only stores the information that is relevant for referencing the transmitted location information to NDS features in the Routing and Basic Map Display building blocks, as well as for producing language output from traffic information messages.

---

A traffic message typically consists of the following information:

- An event code describing the reason or effect of a traffic incident
- Predefined location codes describing the location affected by the traffic event

---

<b>Note</b>	Location coding can also be done by means of „on-the-fly“ referencing. In this case, existing features are used to uniquely reference a location on the map, such as road class, road name, or coordinates. Thus, on-the-fly referencing does not require specific data structures in the TI building block.
-------------	--

---

This chapter gives an overview of the data structure and concept of the Traffic Information building block, and explains how the traffic information data is linked to NDS features in the Routing and Basic Map Display building blocks.

---

<b>Note</b>	The Traffic Information building block is an optional building block.
-------------	---

---

## 12.1 Building Block Structure and Content

The Traffic Information building block combines the relational and BLOB-based approach for storing data:

- Relational tables, such as TMC and TPEG lookup tables, allow the application to only access the data that is actually needed, instead of loading a whole BLOB into the application's memory.
- BLOBs are, for example, used for the `TmcTileTable` (DataScript location: `nds.ti.tmc.location`). Thus, references between Routing and Basic Map Display features and the flexible attribute `LOCAL_TMC_LOCATION` are set in the same way as other flexible attributes in NDS.

Table 12-1 gives an overview of the data in the Traffic Information building block.

Table 12-1 Data in Traffic Information building block

Type of data	Description
TMC lookup tables	Relations containing descriptions of TMC locations and TMC events
Local lists of flexible attributes for TMC (BLOB)	Flexible attributes of the <code>TmcTileTable</code> , which are used to set up the references between TMC location codes and NDS features in the Routing and Basic Map Display building blocks To set these references, the TMC tiles must exist on the same level as the corresponding tiles in the Routing and Basic Map Display building blocks.

Type of data	Description
Metadata	<p>The following common metadata applies to the TI building block:</p> <ul style="list-style-type: none"> <li>- Version data for TMC events (DataScript location: <code>nds.all &gt; ProductDatabase &gt; bBlockCompVersionTable</code>)</li> <li>- Version data for TPEG events (DataScript location: <code>nds.all &gt; ProductDatabase &gt; bBlockCompVersionTable</code>)</li> <li>- Priority, road number prefix, and road icons for roads (DataScript location: <code>nds.overall.metadata &gt; RoadNumberClassPrefixTable</code>)</li> </ul> <p>The following building block-specific metadata is defined (DataScript location: <code>nds.overall.metadata &gt; TrafficInformationMetadata</code>):</p> <ul style="list-style-type: none"> <li>- Maximum string length of event names (<code>maximumEventNameStringLength</code>)</li> <li>- Maximum string length of location names (<code>maximumLocationNameStringLength</code>)</li> </ul>
TPEG lookup tables	<p>Relations containing descriptions of TPEG events.</p> <p>DataScript location: <code>nds.ti.tpeg &gt; events</code></p>

### Dependencies to Speech Building Block

The speech data associated with name strings in the Traffic Information building block is stored in the Speech building block. Speech data may be phonetic transcriptions or prerecorded voice fragments. The Speech building block also offers the possibility to match the user's speech input with database items (Automatic Speech Recognition, ASR), which can be used, for example, for location input.

The Speech building block is an optional building block. Its availability is indicated in the `UrBuildingBlockVersionTable`.

Phonetic transcriptions and prerecorded voice records reference the associated name string. This reference direction decouples the Speech building block from the Name, Traffic Information, Routing, and POI building blocks. There is an n-to-m relation between phonetic transcriptions and the name string they refer to.

Phonetic descriptions for TMC features and references from phonetic descriptions to TMC data are stored in the `TiPhoneticTraTable` of the Speech building block (DataScript location `nds.speech.phonetictranscription`).

The references from prerecorded voice data to Traffic Information names are stored in the following tables: `TmcEventPrerecordedVTable`, `TmcLocationPrerecordedVTable`, and `TpegEventPrerecordedVTable`. The prerecorded voice data itself is stored in the `TiPrerecordedVTable` of the Speech building block (DataScript location `nds.speech.prerecordedvoice`).

For more information, refer to Chapter 14 *Speech Building Block* on page 281.

## 12.2 Traffic Message Channel (TMC) – Introduction to Concept and Structure

The following section gives an overview of the concept of traffic information via TMC, and the structure for storing TMC data in an NDS database.

---

**Information** The TMC standard is described in EN ISO 14819.

---

TMC is transmitted via the Radio Data System (RDS), a digital sub-channel of standard Frequency Modulation (FM) radio broadcast. TMC bases on a system of predefined location codes, which are hierarchically grouped in location code tables and predefined event codes. The event information describes the reason (for example, accident) and/or effect (for example, traffic jam) of a traffic message. It is transmitted in the TMC message and refers to the coded location. Event lists stored in the application contain the descriptions of the traffic event for translation into text or speech output in the application.

### 12.2.1 TMC Event Information

TMC uses *event codes* to transport event information for TMC messages. The application uses event codes to find text or speech information in lookup tables. TMC bases on the ALERT-C standard, which contains event lists with descriptions of approximately 1500 events.

In addition to these event codes, supplementary information codes can be used to transmit additional information, which is displayed together with the information from the event codes. As the *supplementary information code* is coded with 8 bits within the TMC message, it is limited to 255 supplementary information codes.

The supplementary information is combined with other event information contained in a TMC message. The information transported by supplementary information codes can be categorized. Categories are, for example, vehicle type restrictions, diversion hints, and general instructions.

---

**Example** Supplementary information combined with event codes for generating text output in the application:

- Event code 202 (serious accident), supplementary information codes 12 (drive carefully) and 112 (approach with care): *Serious accident, drive carefully, approach with care.*
  - Event code 24 (bridge closed), supplementary information code 78 (for heavy lorries only): *Bridge closed, for heavy lorries only.*
- 

The TMC event information is divided into two categories:

- Language-independent information

The TMC standard defines for each event code a set of attributes. The application uses this so-called implicit information for message administration and analysis. For language-independent information, one entry for each event code is sufficient.

- Language-dependent information

Language-dependant information consists of text and/or speech information describing the meaning of the event code or supplementary information code. Each event code and each supplementary information code require a separate text and/or speech representation per language, which is available in the application.

## 12.2.2 TMC Location Information

TMC distinguishes three types of locations (in the following referred to as *TMC locations*):

- **Area locations** specify a region or a country and are mainly used for weather warning messages. They are not relevant for dynamic route calculation, but can be used for text and/or speech information and displayed on a map.

Area locations can be part of a location hierarchy (see Section 12.2.4 *Hierarchical Structure of TMC Locations* on page 236). Linear and point locations refer to the area location in which they are located.

- **Linear locations** specify road segments between two point locations. They are grouped hierarchically as described in Section 12.2.4 *Hierarchical Structure of TMC Locations* on page 236.
- **Point locations** specify freeway junctions and exits as well as important intersections of other major roads.

TMC messages provide information about a location (for example, a stretch of road, an intersection, a region) by means of a location reference, that is, an identifier that can be interpreted without ambiguity by the receiving system. This information is called *location code*. In TMC, locations are predefined and pre-coded, and the codes are stored in location code tables.

The number of location codes per TMC location table is limited due to the location code range of 16 bit as specified by the TMC standard. Thus, together with the location table number (6 bit) and the country code (4 bit), the maximum number of locations that can be assigned to a country is predefined.

### TMC Location Code Tables

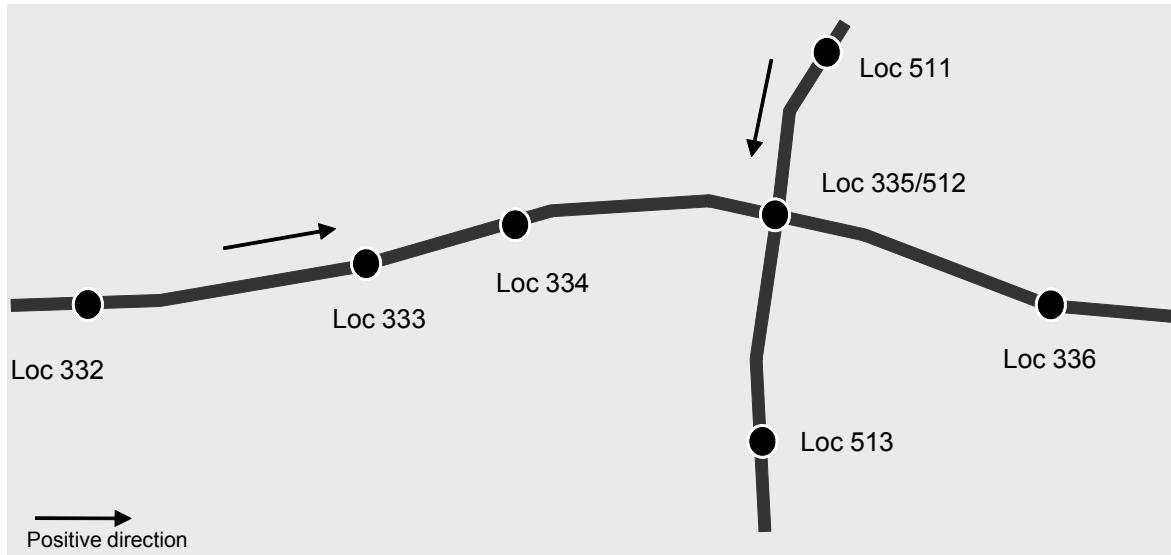
Location code tables are maintained on a national level. They are used to assign location codes to real locations on the road network. The receiving system of a TMC message must use the same location code table as the system that encodes the TMC message.

The rules for location reference coding are specified in part 3 of the TMC standard (EN ISO 14819-3). TMC uses the ALERT-C protocol for transmitting short messages, the so-called *TMC messages*. TMC messages contain information about events that might impact the road traffic, such as traffic flow, road blocks, and weather conditions.

## Sequence of Point Locations

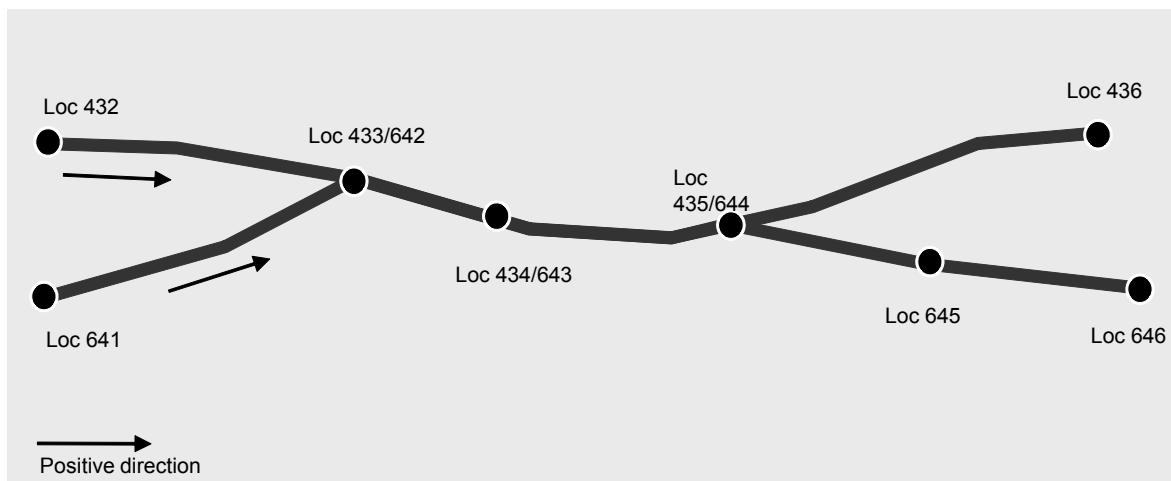
The sequence of point locations follows the roads. Point locations of a given road are usually numbered sequentially. Intersections of two roads with location codes carry two location codes: One along the first road, another along the second road. Figure 12-1 shows an example with the TMC locations LOC 335 and LOC 512.

*Figure 12-1 Sequence of point locations at intersections*



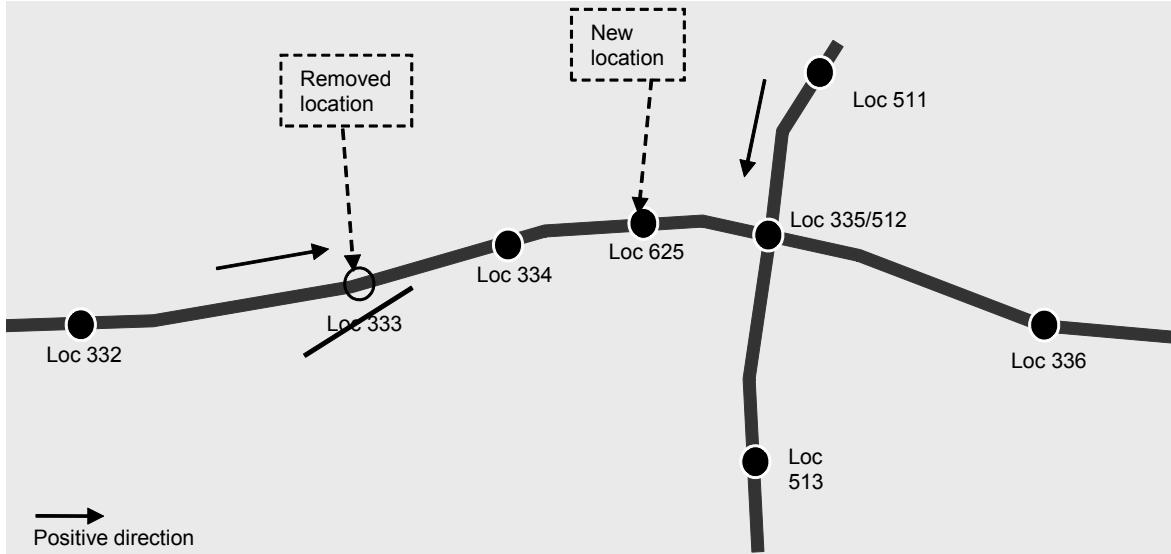
If roads share road sections, the point locations within the shared sections have independent location codes for the individual roads as shown in Figure 12-2.

*Figure 12-2 Location codes along roads with shared road sections*



If a TMC location is removed, for example, due to the closure of a freeway exit, the location code is not reused. The sequence of numbers will have a gap as shown in Figure 12-3 for the TMC location LOC 333. If a new TMC location is inserted for a new freeway exit, for example, a so far unused location code is assigned to the new TMC location. The sequence of location code numbering will thus no longer be subsequent. An example is shown in Figure 12-3 for the point location LOC 625.

Figure 12-3 Changes in the location code sequence



To enable applications to determine the sequence of point locations along a given road, each point location has a reference to its predecessor and successor along this road. The sequence of successor links defines the positive direction along a road. The opposite direction is called negative direction.

### 12.2.3 Determining TMC Locations for a Traffic Event

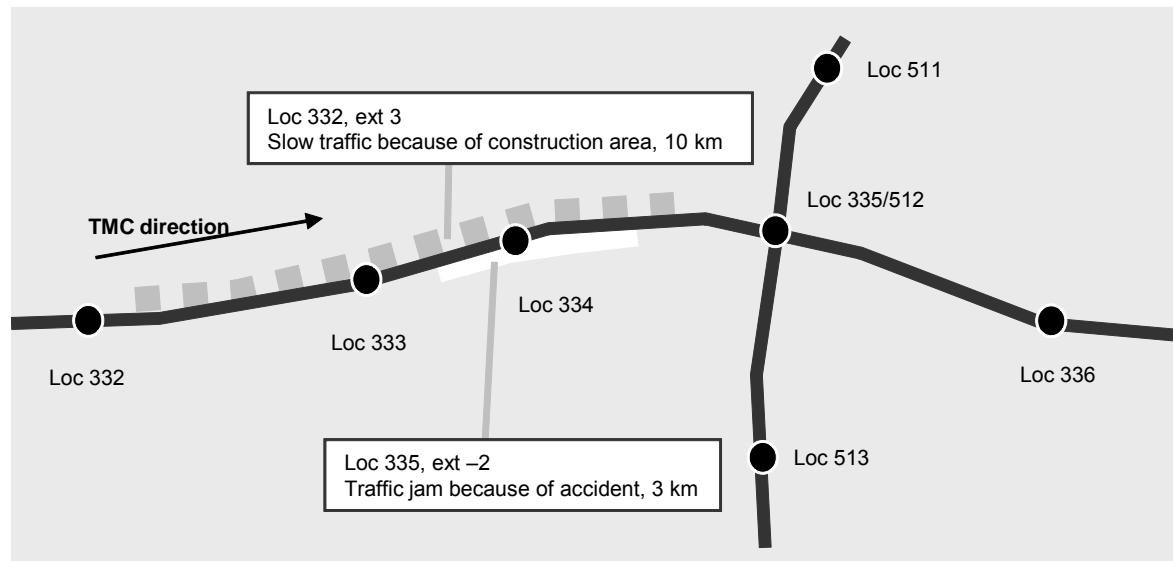
A navigation system can use the sequence of point locations following a road to determine the TMC location for a traffic event received through a TMC message. To enable applications to define the road section affected by the traffic event, TMC messages contain a point location code, a direction, and the extent.

The concept of primary and secondary location codes is used to indicate the extremities of the affected sections without having to list all the intervening places. The ALERT-C protocol defines such occurrences by addressing the TMC location of the accident as the primary location, and then identifying the secondary location (that is the tailback) by using the direction and extent fields.

The direction identifies the queue growth, and not the direction of the traffic flow, and is indicated by a positive or negative extent value (-/+). The extent identifies the number of locations along the road that are affected by the event. The extent is a number ranging from -7 to +7 and up to ±31 for multi-group messages.

Figure 12-4 shows an example of a traffic event consisting of 10 km of slow traffic between TMC locations LOC 335 and LOC 332 (eastbound). The length of the event is defined by the primary location with the location code LOC 332 and an extent of +3 (positive direction). The head of the traffic jam, where possibly the reason is to be found (for example, an accident), is located towards the primary location, and the tail of the traffic jam is located towards the secondary location (LOC 335). Figure 12-4 shows a second event between TMC locations LOC 335 and LOC 333 with a negative extent of -2.

*Figure 12-4 Direction and extent of traffic events, example*

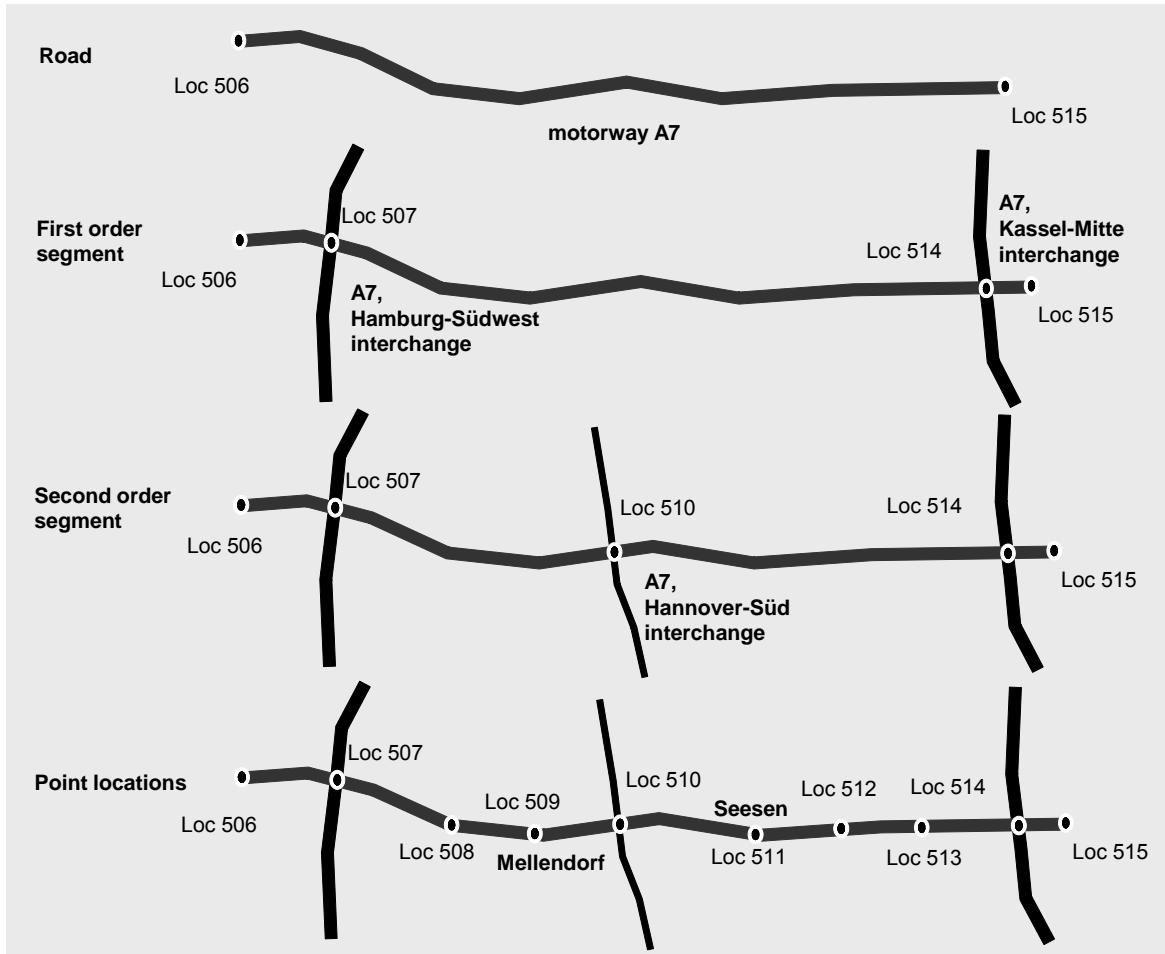


#### 12.2.4 Hierarchical Structure of TMC Locations

TMC segments are linear locations grouped into first and second order segments. They group road sections between two point locations, for example, between freeway intersections. The order group of a segment is defined in the location code table. Second order TMC segments are referenced to first order TMC segments. The first order segments form a TMC road.

The hierarchy is illustrated in Figure 12-5.

Figure 12-5 TMC location hierarchy



## 12.2.5 Linking TMC Information to NDS Features

Routing applications interpret messages about traffic events, and consequently adapt the traffic throughput or traffic flow rate for the affected roads. Based on the transmitted traffic information, alternative routes avoiding roads with reduced traffic flow are calculated and displayed in the navigation system.

Traffic information data needs to fulfill the following requirements:

- References to all levels of data that are used in the Routing and Basic Map Display building blocks
- Access to the regional parts of location codes to make it possible to extract a region out of a larger database, for example, all location codes along or nearby a calculated route or in the neighborhood of a given position
- Possibility to display the message content in maps or as a textual message in the navigation system. For this purpose, the systems shall be enabled to decode the traffic messages to text or synthetic speech of different languages. This is done via lists containing the name for each TMC location, and a textual description for each defined event (the reason for a traffic disruption).

TMC locations often are freeway junctions and exits. In general, they consist of several intersections and links in the topological road network of a digital map.

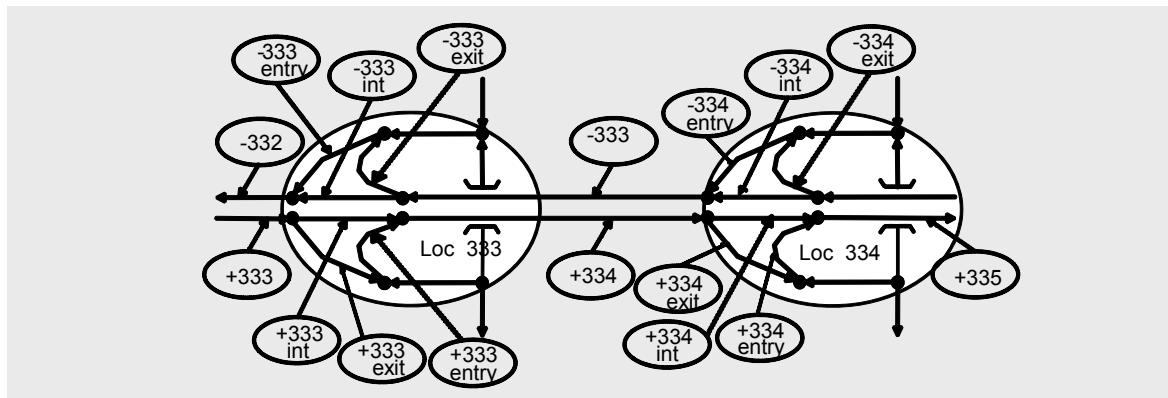
---

<b>Note</b>	Assignments of area locations and segment locations to routing and basic map display features will not be stored.
-------------	---

---

For the route calculation algorithms and for map display purposes, a traffic message relating to a specific TMC location must be assigned to links rather than to intersections. Therefore, the TMC locations need to reference the corresponding links. Figure 12-6 shows a schema of references between TMC locations and links in an NDS database (ext, int, exit, and entry indicate the TMC link type, see Table 12-4).

Figure 12-6 Schema for referencing TMC locations to links in an NDS database



## 12.2.6 TMC Data Structures

NDS uses the following data structures to link traffic information and NDS features:

- Lookup tables for event and location information
- Flexible attributes assigned to features of the Routing and Basic Map Display building blocks

## Lookup Tables for Event Information

The administrative information as well as the text or speech output information of incoming TMC messages is stored in lookup tables. The application uses these lookup tables as follows:

- The application can update or delete stored TMC messages based on the administrative information in the lookup table
- The application can generate text messages sorted in a list by means of text strings associated with the event and supplementary information codes to inform a driver about traffic incidents

In contrast to spoken traffic messages from radio stations, these lists are permanently available. The user can sort or filter them according to his needs, for example, the current car position.

For some events, the lookup tables may provide placeholders, which are replaced by optional values in the TMC message. These so-called *quantifiers* can impact the grammar of an event text, because the values can have singular or plural form, or no quantifier at all is sent.

The following types of quantifiers exist:

- Physical values, for example, length, temperature
- Cardinal numbers, for example, the number of lorries involved in an accident or the number of lanes closed

The following figure depicts the relational database schema for event information in NDS.

Figure 12-7 Relational database model for TMC event information

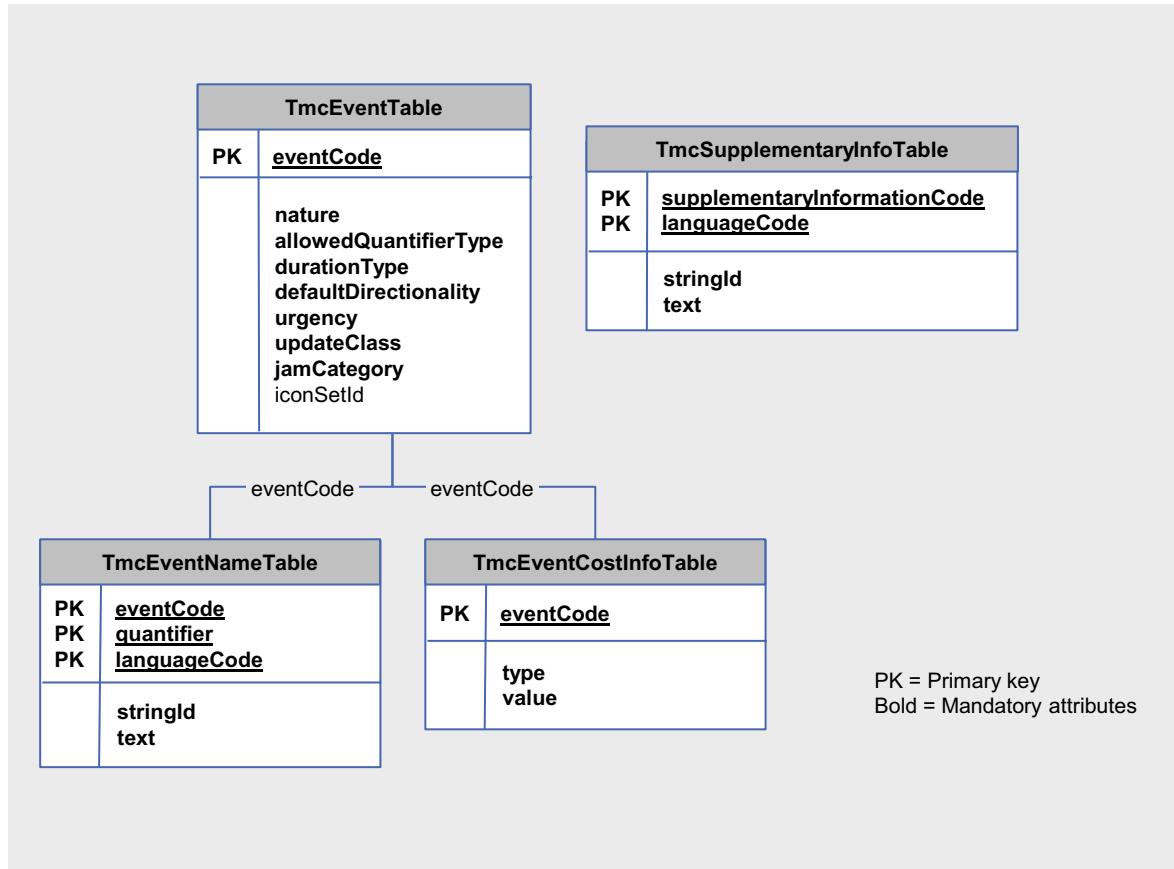


Table 12-2 gives an overview of the lookup tables used for event information. DataScript location: `nds.ti.tmc > event`

Table 12-2 Lookup tables for event information, overview

Table Name	Description
TmcEventTable	<p>Contains language-independent information, which is not used for text and/or speech output</p> <p>The <b>TmcEventTable</b> stores for each event code the implicit information for TMC message administration and the icon set to be used for an event code.</p> <p><b>jamCategory</b> contains a machine-readable version of those textual statements which are relevant to determine the costs caused by a traffic jam (for example, „queuing traffic“, „heavy traffic“).</p>

Table Name	Description
TmcEventNameTable	<p>Contains the text strings for each event code in all available languages</p> <p>For events, which can be used with quantifiers for numbers, several text strings per event may be stored. These text strings can be used for quantifiers in singular or plural form, or if no quantifier is available. The <code>stringId</code> points to a speech data record in the Speech building block.</p>
TmcSupplementaryInfoTable	<p>Contains the text string for each supplementary information code in all available languages</p> <p>The <code>stringId</code> points to a speech data record in the Speech building block.</p>
TmcEventCostInfoTable	<p>Event texts may contain hard-coded physical values, which may be relevant for determining the routing costs caused by an event. In order to avoid parsing event texts at runtime in the application, this table stores the value contained in the event text and the type of the value (for example, speed) for such events.</p> <p><b>Note</b> The values stored in this table are not actual routing costs but rather represent data which can be used to derive routing costs.</p>

## Lookup Tables for Location Information

The lookup tables for location information contain the data that is derived from the TMC location tables, enhanced with some additional information. They are used to reference TMC location codes to NDS tiles and features of the Basic Map Display and Routing building blocks.

The lookup tables for location information are the entry points for decoding TMC messages. They allow to find all information that is relevant for the location code contained in the TMC message.

Figure 12-8 depicts the relational database schema for location information in NDS.

Figure 12-8 Relational database model for TMC location information

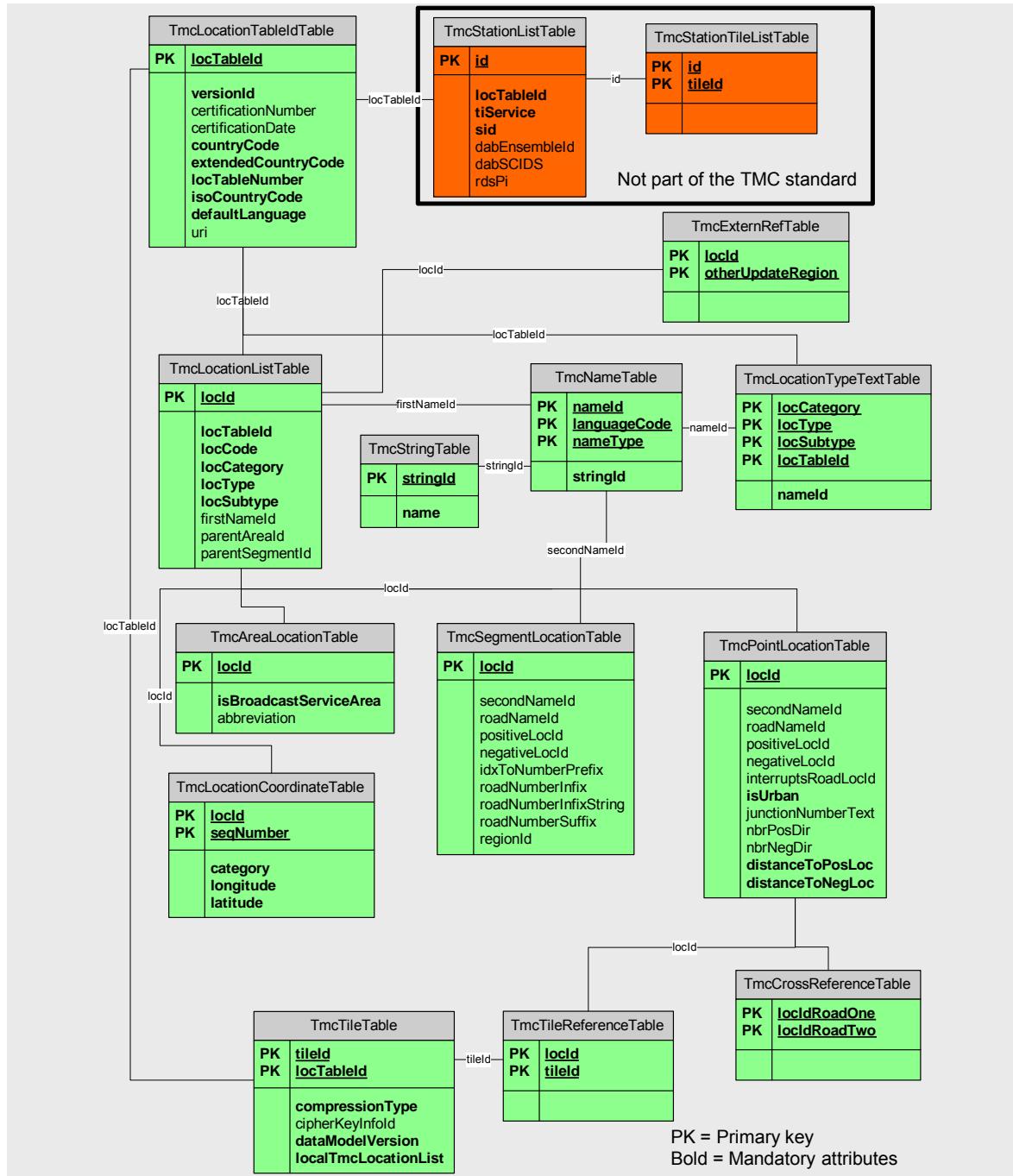


Table 12-3 gives an overview of the lookup tables used for location information. DataScript location: nds.ti.tmc > location

*Table 12-3 Lookup tables for location information, overview*

Table Name	Description
<b>TmcLocationListTable</b>	<p>Entry point for retrieving the relevant NDS data for a location code received in a TMC message</p> <p>The <b>locId</b> is used to retrieve further location-related information from other tables.</p>
<b>TmcLocationTableIdTable</b>	<p>Contains all TMC location tables, which are uniquely defined by the TMC standard by country code, extended country code, and location table number</p>
<b>TmcAreaLocationTable</b>	<p>Contains area location-specific information</p>
<b>TmcSegmentLocationTable</b>	<p>Contains linear location-specific information</p>
<b>TmcPointLocationTable</b>	<p>Contains point location-specific information</p>
<b>TmcNameTable</b>	<p>Contains all location-related strings</p> <p>The translations of a text string have the same <b>stringId</b>, but a different <b>languageCode</b>.</p> <p>It is possible to divide the name string into several parts, such as prefix, body, suffix, and abbreviated versions. Thus, applications with size restrictions can display shorter names.</p>
	<p><b>Note</b> The names stored in the <b>TmcNameTable</b> are assigned by the publisher of the location code table and can, thus, be different from the names stored in the Name building block. The discrepancies that may occur with the names used for Basic Map Display, can, however, be tolerated.</p>
	<p>For more information about names in NDS, refer to Chapter 10 <i>Name Building Block</i> on page 163.</p>
<b>TmcLocationTypeTextTable</b>	<p>Contains the text representations of location types defined by <b>locCategory</b>, <b>locType</b>, and <b>locSubtype</b></p> <p>Since the TMC standard leaves a certain level of freedom to local authorities for assignment of road categories, it is necessary to store the text representations per country. The country is identified via <b>TmcLocationTableIdTable</b> that is accessed by means of <b>locTableId</b>.</p>

Table Name	Description
TmcLocationCoordinateTable	<p>Stores coordinates with locations Depending on the location category, the coordinates have the following meanings:</p> <ul style="list-style-type: none"> <li>- For area locations, the coordinate is a point of the polygon representing the outline of the TMC area location</li> <li>- For linear locations, the coordinate is one of the points of the outline of the TMC linear location. The outline may be formed, for example, by point locations, which reference the linear location (attribute <code>parentSegmentId</code>).</li> <li>- For point locations, the coordinate may be defined for several parts of a complex point location (center of the point location, coordinates of entries and exits).</li> </ul> <p><b>Note</b></p> <p>Applications can use the coordinates of entries and exits to precisely position colored traffic flows on the calculated route on the map. The coordinates from the TMC location tables need to be converted into NDS coordinates.</p>
TmcCrossReferenceTable	<p>Stores the point location codes for roads, which run at the same link(s) and are coded with separate location codes. The application can use this information to find out if two separately reported messages are (partly) covering the same link(s).</p> <p><b>Note</b></p> <p>Motorway A1 in Greece shares the link (road segments) with motorway A2 between Klidi and Axios.</p> <p>If the information is not needed, the <code>TmcCrossReferenceTable</code> may be left empty.</p>
TmcTileReferenceTable	<p>This table allows to look up tile(s) with links associated to certain TMC point locations (identified by <code>locId</code>). The information about the links is stored in <code>TmcTileTable</code>. The information about the association between tile(s) and TMC locations are normalized into two tables in order to avoid redundant data.</p>

Table Name	Description
<b>TmcTileTable</b>	<p>Stores the link information for TMC point locations in form of flexible attributes per tile in a BLOB</p> <p>The link information is relevant for routing and basic map display. Since a tile can cover more than one country, the <b>locTableId</b> and the <b>tileId</b> form the primary key. This covers the cases in which identical TMC location codes of different location tables are located within one and the same tile.</p>
<b>TmcExternRefTable</b>	<p>Stores the IDs of the TMC location and of the respective update region for each location in the TMC location table that is located in another update region</p> <p>The <b>TmcTileReferenceTable</b> and <b>TmcTileTable</b> have no entries for those locations, whereas the other tables store all location data, regardless of the update region.</p>
<b>TmcStationTileListTable</b>	<p>Stores the preferred stations per tile</p> <p>This table is optional, that is, it is not part of the TMC standard. The application can use this information to tune to a preferred station depending on the current car position. To avoid redundant data, the information specific to a station, and the allocation of stations to tiles is normalized into two tables.</p>
<b>TmcStationListTable</b>	<p>Contains the mapping of stations to tiles</p> <p>This table is optional, that is, it is not part of the TMC standard. Each radio station sending TMC messages can be received in a certain area and sends TMC messages for a specified location table. There exist several data bearers (RDS, DAB, etc.) to be used by the stations, depending on which the attributes of this table can be filled.</p>

### Local Attribute Lists

References between TMC locations and features in the Routing and the Basic Map Display building blocks are stored as flexible attributes in tiles of the TMC building block. The attributes are used to:

- Determine whether a route is affected by a TMC message
- Highlight the affected sections of the road network on the display

The TMC attributes assign location codes and TMC-relevant properties, such as direction and flags, to base links, route links, road geometry lines, and map lines for each location code. Thus, location codes may be assigned to more than one feature.

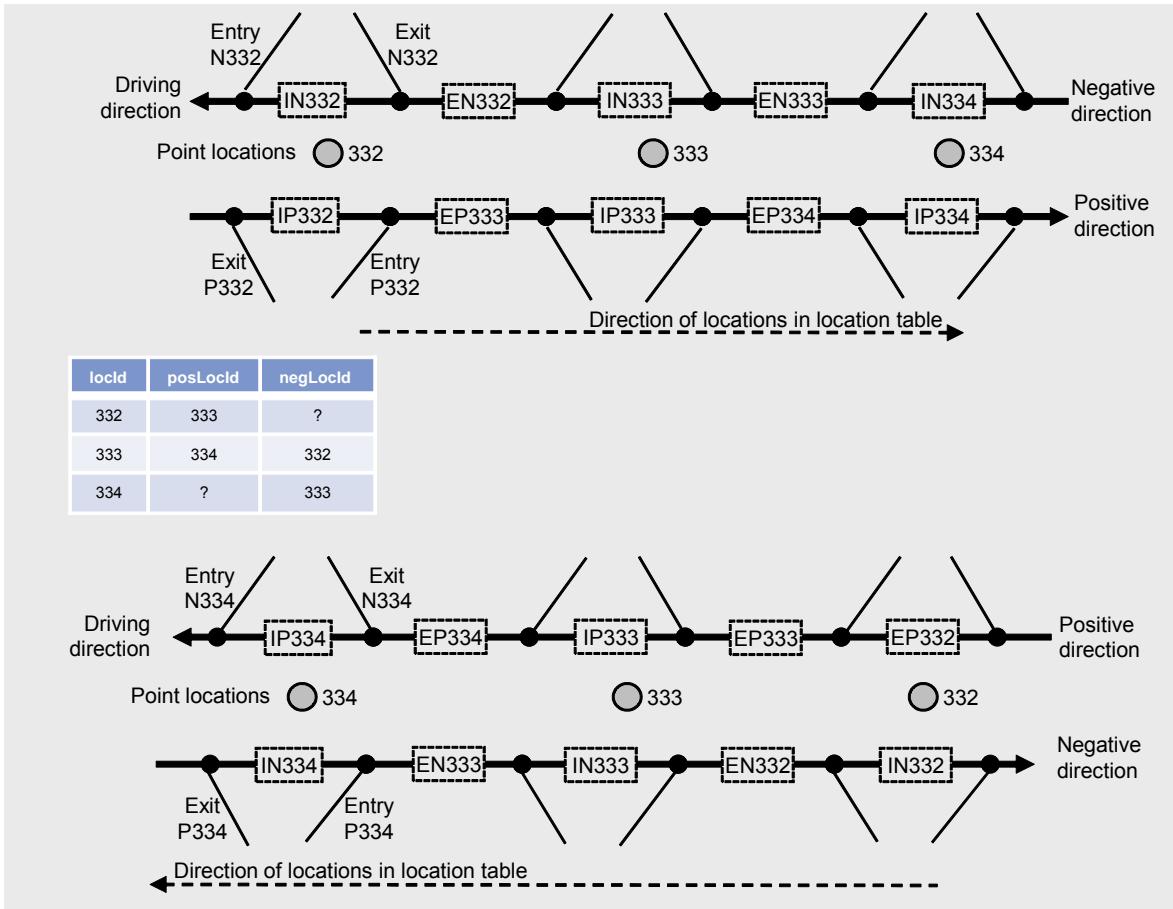
DataScript location: `nds.common.flexibleattributes > LocalTmcLocation`

Table 12-4 Elements of the flexible attribute LocalTmcLocation

Element	Description
tmcDataType	Defines the referencing information that is used for a product The following information may be used for references: – TMC location code – TMC location ID (see Table 12-2) Each product shall only use one referencing type.
tmcLinkType	The link type may be defined as follows: – INTERNAL for links within a TMC location – EXTERNAL for links between TMC locations – ENTRY or EXIT for ramps directly connected to a road carrying location codes
tmcDirection	Identifies the direction information for a link (POSITIVE_DIRECTION, NEGATIVE_DIRECTION) The direction depends on the chaining direction of the TMC locations in the TMC location table. Positive direction is set for links on which the driving direction follows the positive chaining direction (positiveLocId), negative direction is set for links in the opposite direction (negativeLocId).

Figure 12-9 shows examples how direction information for links is identified.

Figure 12-9 Identifying direction information for links



For an example how to calculate the distance between locations, refer to *NDS – Compiler Interoperability Specification, 9.3 Calculation of Distances between Locations* on page 178.

Figure 12-10 shows an example of TMC locations spread over different tiles. The numbers in circles represent base/route link IDs with TMC location codes. The links are stored in the tile in which they start. Each row in Table 12-5 represents an entry in the local TMC attribute list.

Figure 12-10 Example of locations in different tiles

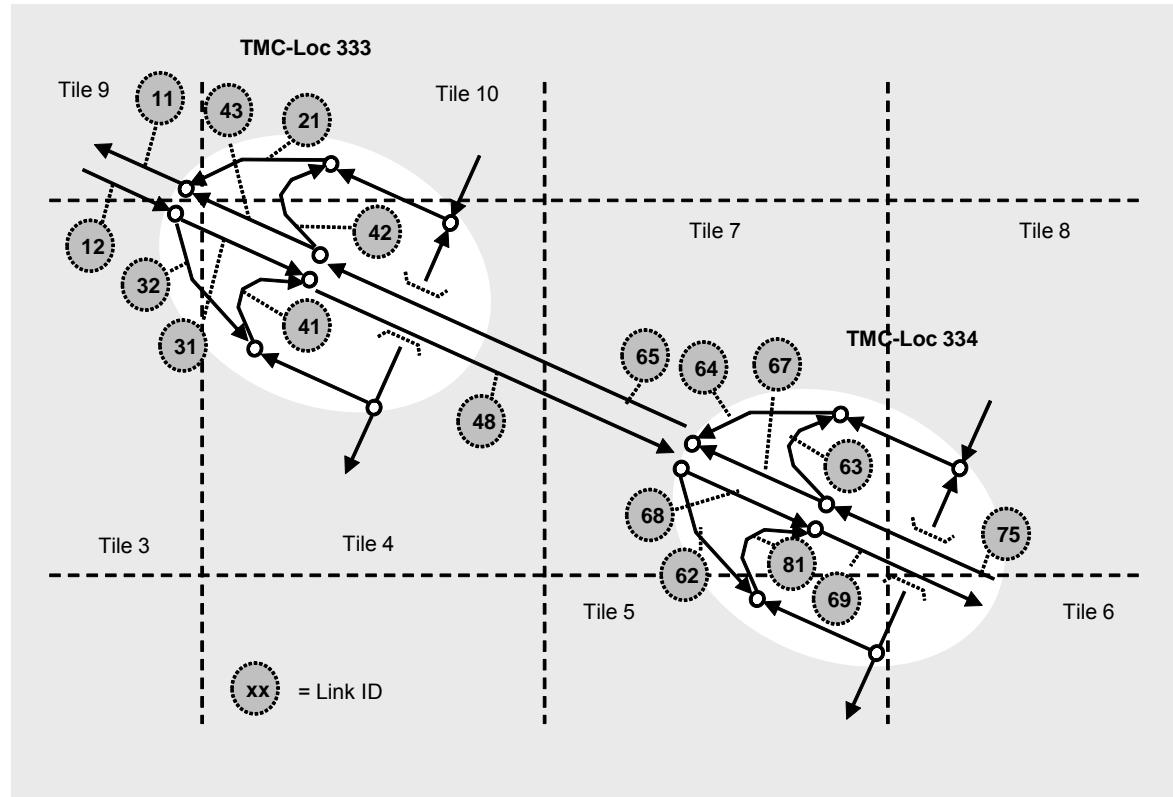


Table 12-5 Local attribute list of TMC locations

TMC Location Code	TMC Direction	TMC Link Type	Tile ID	Link ID
332	Negative	ext	9	11
333	Positive	int	3	31
333	Positive	exit	3	32
333	Positive	entry	4	41
333	Negative	exit	4	42
333	Negative	int	4	43
333	Negative	ext	7	65
333	Positive	ext	9	12
333	Negative	entry	10	21
334	Positive	ext	4	48
334	Positive	entry	5	81
334	Negative	ext	6	75
334	Positive	exit	7	62
334	Negative	exit	7	63

TMC Location Code	TMC Direction	TMC Link Type	Tile ID	Link ID
334	Negative	entry	7	64
334	Negative	int	7	67
334	Positive	int	7	68
335	Positive	ext	7	69

## 12.3 Transport Protocol Experts Group (TPEG)

**Note** The TPEG implementation in NDS bases on the TPEG generation 2 series. This is currently a TISA-internal standard and will be published soon as ISO TS 21219 series.

The Transport Protocol Experts Group (TPEG) defines a byte-oriented, bearer-independent protocol to transmit traffic information data from the provider to the application. TPEG supports a container concept that enables backwards compatible extensions of the TPEG standard.

By means of Application Data Containers (ADC), new TPEG protocol subsets can be created. These subsets are called TPEG applications. An example of a TPEG application is TEC (Traffic Event Compact).

The Location Referencing Container (LRC) allows using several location referencing methods for TPEG messages. Examples for location referencing methods are TPEG-native methods, or standards outside TPEG, such as TMC location codes.

The following section introduces the concept of traffic information via TPEG, and how the TPEG data structures are stored in an NDS database.

### 12.3.1 TPEG Event Information

Similar to TMC, the event information, which is transmitted in the ADC, describes the reason and/or effect of a traffic message. Parts of the event information are transmitted by means of codes. The application uses the codes to look up event information, such as text or speech output, in event lists stored on the application.

TPEG only stores language-dependent information for TPEG events in the database. Language-independent information for message management, such as „update“, „delete“, and „message expiration“, is decoupled from the event information, and is transported in separate fields.

The TPEG standard defines a separate set of so-called *TPEG tables* for each TPEG application. For TEC, TPEG defines, amongst others, the following tables:

- Table for texts indicating the cause of a traffic disruption
- Table for texts describing the level of service (heavy traffic, queuing traffic, stationary traffic, etc.)
- Table for texts giving advice

<b>Note</b>	Not all such TPEG tables contain information for text or speech output. Some tables contain information that only impacts the application logic in the application. These tables do not require to store data in NDS.  An example is table <code>tec003:WarningLevel</code> . This table can be used to prioritize the message processing subject to the warning level.
-------------	---

## Lookup Tables for Event Information

Lookup tables for event information contain text/speech representations of codes transported within a TPEG message.

The event information for TPEG is handled the same way as TMC event information: The text representations can be used to generate text messages, which inform the driver about traffic incidents. Some TPEG event texts contain placeholders, which can be replaced by optional values contained in the TPEG message. These values can represent physical values or cardinal numbers, and thus may need a specific event text entry to provide a grammatically correct (sub)sentence depending on the type of value or quantifier.

In the current version of NDS, only the tables of TPEG-TEC events are modeled. However, any other TPEG application providing TPEG tables representing text/speech output may be added to NDS by adding TPEG application-specific lookup tables.

Figure 12-11 depicts the lookup table for the event information of TEC in NDS.

*Figure 12-11 Lookup table for TEC event information in NDS*

TpegTecEventNameTable	
<b>PK</b>	<u>eventCode</u>
<b>PK</b>	<u>quantifier</u>
<b>PK</b>	<u>languageCode</u>
<b>PK</b>	<u>tableNumber</u>
	<b>stringId</b>
	<b>text</b>
	<b>iconSetId</b>

PK = Primary key  
Bold = Mandatory attributes

Table 12-6 gives an overview of the lookup tables used for event information (currently only one table).

DataScript location: `nds.ti.tpeg > event`

Table 12-6 Lookup table for event information, overview

Table Name	Description
TpegTecEventNameTable	<p>Contains the text string for each TEC event code in all available languages</p> <p>TPEG defines many small tables for TEC, which are stored in NDS in one table to avoid a large number of small tables in the application. The attribute <code>tableNumber</code> indicates the respective TPEG table as defined by the TPEG standard.</p> <p>For events that can be used with quantifiers for numbers, several text strings per event may be stored. These text strings can be used for quantifiers in singular or plural form, or if no quantifier is available. The <code>stringId</code> refers to a phonetic transcription, which is stored in the Speech building block.</p>

### 12.3.2 TPEG Location Information

TPEG supports both, on-the-fly and precoded referencing.

On-the-fly referencing for location information uses existing features of other building blocks to uniquely identify a location on the map, such as road names, road class, or coordinates. Therefore, no dedicated data structures for location information are required in the Traffic Information building block. The referencing method DLR1, which is derived from AGORA-C, is an example of „on-the-fly“ referencing.

For precoded referencing, dedicated data structures for location information must be provided in the Traffic Information building block. NDS provides data structures for TMC, which is also supported by TPEG and may, thus, be used for precoded referencing in NDS. For information, refer to Section 12.2 *Traffic Message Channel (TMC) – Introduction to Concept and Structure* on page 232.

Note	<p>TPEG references TMC locations as specified in ISO/TS 17572-2, and not as specified in EN ISO 14819. When using TMC locations for TPEG, the following differences need to be considered:</p> <ul style="list-style-type: none"> <li>■ The maximum extent for TMC locations in TPEG is 255, and <b>not</b> 31.</li> <li>■ The positive chaining direction within the location table in TPEG is indicated with direction flag value TRUE, and <b>not</b> with value 0.</li> </ul>
------	---

### 12.3.3 TPEG Data Design

As described in Section 12.3.2 *TPEG Location Information* on page 251, no location information needs to be stored for TPEG. Thus, all NDS needs to define are lookup tables for event information.

## 12.4 Vehicle Information and Communication System (VICS)

Information for other traffic information systems, such as VICS may be added to an NDS-compliant product as a proprietary add-on, see Section 3.5 *Extending the Navigation Data Standard* on page 55. To indicate that VICS information is available for a specific route or intersection, the flexible attribute HAS\_VICS is assigned to links and intersections.

## 13 POI Building Block

Points of Interest (POI) specify point locations that are useful or interesting. Navigation systems can provide additional information about POIs, for example, the opening hours of a petrol station. Based on this information, a user can select a POI as a destination for routing.

POI in general is understood as a service for a defined location, giving additional information that helps a user to decide if this destination is of interest.

There are different possibilities to make POI information available:

- Displaying an icon on the map (for example, a petrol station)
- Text, pictures, or sound on request (for example, after selecting an icon on the map)
- Automatically at predefined POI locations along a route (for example, speed trap warnings, information about buildings and other landmarks)

Usually, a user would search for a hotel or a petrol station within a specific area without knowing an address. To support this search, POIs are assigned to categories and carry additional information that helps to refine the search (see Section 13.2.2 *Secondary POI Attributes* on page 260).

Areas of interest are not necessarily points but can also be lines (for example, historic steam railways) or areas (for example, parks) that are displayed on the map. The latter two are modeled by means of references from POI features to line and area features of the Basic Map Display building block.

POIs can be permanent or temporary. Temporary POIs, such as exhibitions, are characterized by a short duration and a higher update frequency. For temporary POIs, a duration is defined and they are handled differently regarding update.

### 13.1 Building Block Structure and Content

The POI building block is an optional building block and is stored in its own database.

#### Types of POI Building Block

There are two possible types of the POI building block:

- Integrated POI building block

The integrated POI building block may contain references to other building blocks of an NDS database. The following references are available:

- References to route links and intersections (Routing building block). These references are stored in the `PoiLinkAccess` and `PoiIntersectionAccess` tables.

---

**Note** By default, references from the POI building block to links in the Routing building block use the feature ID. However, these references may also be modeled by means of a global feature ID. In this case, the flexible attribute `GLOBAL_FEATURE_ID` must be assigned to the related link or intersection. See also Section *Identifying Features* on page 50.

Using global feature IDs enables references to Routing building blocks of different NDS products and thus reuse of the POI database for several NDS product databases.

- 
- References to named objects (Name building block). These references are stored in the `PoiNamedObjectRelationTable` table.

- Non-integrated POI building block

The non-integrated POI building block does not contain direct references to features of other building blocks of an NDS database. The non-integrated POI building block enables an easier use of third-party POIs in NDS databases.

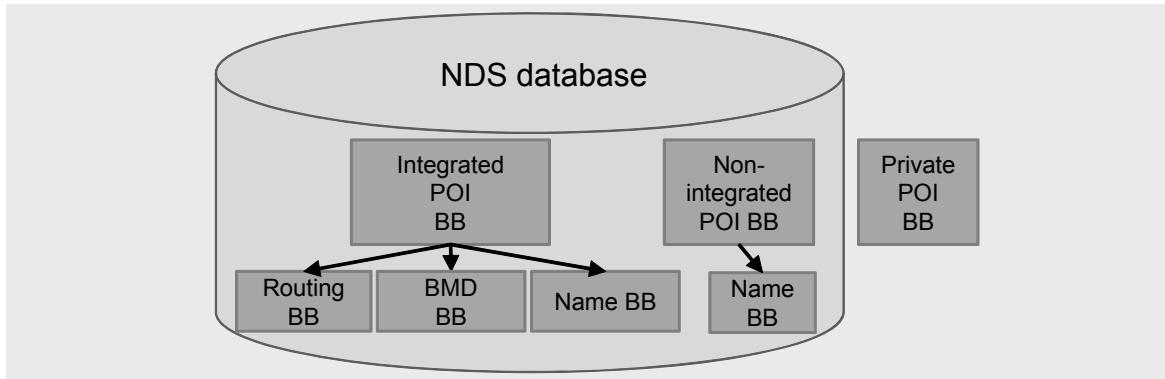
In order to provide names for administrative areas, non-integrated POI building blocks may include a separate Name building block in the database. See also Section *Relation between POI Regions and Named Objects* on page 275.

The non-integrated POI building block uses the same NDS data structures as the integrated POI building block. The POI building block type is derived from the metadata values in the `PoiMetadataTable` table (DataScript location: `nds.poi.metadata`).

In addition to the POIs in the integrated and non-integrated POI building block, an application can collect and save destinations defined by the user as private POI data. Private POIs are user-defined locations that are stored permanently and can be selected for location input at any time. Private POI data must be stored by the application in such a way that they do not interfere with the main POI data.

Figure 13-1 illustrates the database structure.

Figure 13-1 Integrated and non-integrated POI building block



### Data and Storage Method

The data of the POI building block is stored relationally, with the exception of the secondary POI attributes, which are stored in BLOBs.

As POI data is not physically stored in tiles, it does not use explicit leveling, as for example, the data in the Routing building block. Each POI is stored with the geographic location encoded as explained in Section 7.1 *WGS 84 and EGM96* on page 89 and Section 7.2 *Coding of Coordinates* on page 89.

To support specific regional search scenarios, however, the POI building block features virtual tiles, see Section 13.6.3 *Virtual Tiles for POIs* on page 277.

Table 13-1 shows the data contained in the POI building block.

Table 13-1 Data in POI building block

Type of data	Description
POI features	<p>POIs are modeled by means of the POI feature class. POI categories are properties of POI features.</p> <p>For more information, refer to Section 13.2 <i>POI Feature Class</i> on page 259 and 13.2.2 <i>Secondary POI Attributes</i> on page 260.</p>
Metadata	<p>The metadata for the POI building block comprises the following:</p> <ul style="list-style-type: none"> <li>– Definition of supported attributes</li> <li>– Route link reference type</li> <li>– Administrative structure type</li> <li>– Indication whether advanced administrative search is available</li> <li>– Maximum length for name strings for POIs, POI categories, and POI attributes</li> </ul> <p>For more information, refer to <i>POI Metadata</i> on page 255.</p>

### POI Metadata

The following table gives an overview of the metadata for the POI building block.

Table 13-2 Metadata of the POI building block

Metadata	Description
Name string length (maximumPoiNameStringLength, maximumPoiCategoryNameStringLength, maximumPoiAttributeNameStringLength)	Defines the name string length for POIs, POI categories, and POI attributes DataScript location: nds.overall.metadata > PoiMetadata
Supported attributes (supportedAttributes)	Defines supported attributes DataScript location: nds.poi.metadata > PoiMetadataTable
Reference type (routeLinkRefType)	Defines the type of reference to the routing building block DataScript location: nds.poi.metadata > PoiMetadataTable
Type of administrative structure (adminStructureType)	Depending on the building block type, the administrative structure type defines the internal or external administrative structure stored in tables and structures of the nds.name.* package. DataScript location: nds.poi.metadata > PoiMetadataTable
Advanced administrative search (hasAdvancedAdminSearch)	Indicates whether the advanced administrative search (region sets) is available DataScript location: nds.poi.metadata > PoiMetadataTable
Type of building block (isIntegratedPoiBuildingBlock)	Indicates whether it is an integrated or non-integrated POI building block DataScript location: nds.common > BuildingBlockDetailedType
Mapping between categories and secondary attributes (PoiCatToAttrTable)	Defines the secondary attributes available for each POI category DataScript location: nds.poi.metadata

## POI Attributes

POI attributes can be used as filtering criteria for POI search and/or for storing additional information on POIs. There are two groups of POI attributes:

- Primary POI attributes

Primary attributes are relational attributes and are stored as fields in POI-related tables. Primary attributes are used as primary filtering criteria when searching for POIs.

- Secondary POI attributes

Secondary POI attributes supply additional information (values) for primary POI attributes to further refine the search. A restaurant POI, for example, has the secondary attribute Payment Method with values for different credit cards. This POI will be found if an application specifies a credit card name as a search criterion.

Secondary POI attributes containing information that cannot be used for searching, but only provide additional information, are called *supplementary attributes*. A video clip for a museum POI, for example, is not used as a search criterion. It is a supplementary attribute storing a reference to a video clip.

For more information on secondary POI attributes, refer to Section 13.2.2 *Secondary POI Attributes* on page 260.

## Dependencies to Routing Building Block

POIs can be used as destinations for route calculation. For this purpose, POIs must have relations to the road network. NDS provides the following options:

- Option 1: Find the point of the road network that is closest to the location of a POI. The point can be on a link or at an intersection. An algorithm to find such a point can be executed by an application and does not need additional data.
- Option 2: Use simple access points (see Section 13.3.2 *Access Relations between POIs* on page 265).
- Option 3: Use an AGORA-C reference for map-based on-the-fly location referencing.<sup>1)</sup> The AGORA-C reference can be stored as a flexible secondary POI attribute.

It is possible that option 1 causes problems if a POI is located close to a freeway, but can only be accessed via a residential road which is not as close to the POI as the freeway.

In general, using simple access points (option 2), such as link or intersection access points, is the most accurate option, because the access points contain references to elements of the road network. The reference can be stored for an intersection, a link as a whole, or to a point on a link. If a POI can be accessed from different roads, it may have more than one reference.

Simple access points, however, create dependencies to the Routing building block. Using geo access points avoids these dependencies, but is possibly inaccurate.

Option 3 offers a database-independent referencing and enables relations between two databases covering the same region with similar, but not necessarily identical content. These references, however, require more storage space than a database-dependent referencing and can decrease performance.

---

1) AGORA-C is defined in ISO 17572-3

## Dependencies to Speech Building Block

The speech data associated with name strings in the POI building block is stored in the Speech building block. Speech data can be phonetic transcriptions or prerecorded voice fragments. The Speech building block also offers the possibility to match the user's speech input with database items (Automatic Speech Recognition, ASR) which can be used, for example, for location input.

The Speech building block is an optional building block. Its availability is indicated in the `UrBuildingBlockVersionTable`.

Phonetic transcriptions and prerecorded voice records reference the associated name string. This reference direction decouples the Speech building block from the Name, Traffic Information, Routing, and POI building blocks. There is an n-to-m relation between phonetic transcriptions and the name string they refer to.

The references from phonetic transcriptions to POI names, POI category names, and POI attribute names are stored in the Speech building block, in the column `phoneticTranscriptionId` of the following tables: `PoiNameToPhoneticTraTable`, `PoiCatNameToPhoneticTraTable`, and `PoiAttrNameToPhoneticTraTable`. The phonetic transcriptions themselves are stored in the `PoiPhoneticTraTable` of the Speech building block.

The references from prerecorded voice data to POI names, POI category names, and POI attribute names are stored in the Speech building block, in the column `nameStringId` of the following tables: `PoiNameToPrerecordedVTable`, `PoiCatNameToPrerecordedVTable`, and `PoiAttrNameToPrerecordedVTable`. The prerecorded voice data itself is stored in the `PoiPrerecordedVTable` of the Speech building block.

For more information, refer to Chapter 14 *Speech Building Block* on page 281.

## Dependencies to 3D Building Block

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 of the spatial tree BLOB.
- 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 with the bounding box whose center point is located nearest to the POI position.

For more information, refer to Section *NDS – Compiler Interoperability Specification*, 10.10 *References from POIs to Basic Map Display Features* on page 190.

## Dependencies to Basic Map Display Building Block

POIs may reference features of the Basic Map Display building block by means of secondary attributes.

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.

For more information, refer to Section *NDS – Compiler Interoperability Specification*, 10.10 *References from POIs to Basic Map Display Features* on page 190.

## 13.2 POI Feature Class

Points of Interests (POIs) define specific locations that can be used for location input and for storing information about the specific location.

POI data is stored in the **PoiTable**. The table contains the unique POI ID and a corresponding version ID which is needed for updating (see Section 13.7 *Versioning of POIs* on page 279). It also contains information about the region ID, postal code, and phone number attribute (see Section 13.2.1 *Primary POI Attributes* on page 259).

### 13.2.1 Primary POI Attributes

The following primary attributes are available for POIs.

*Table 13-3 Primary attributes of POIs*

Attribute	Description
Geographic position	Stored in the <b>mortonCode</b> field of the <b>PoiServiceLocationTable</b> The geographic position enables a spatial search in POI databases. The search result shows all POIs inside a defined rectangle on the map. The geographic location is encoded by using Morton code. For more information on Morton codes for NDS, refer to Section 7.2 <i>Coding of Coordinates</i> on page 89 and <i>NDS – Compiler Interoperability Specification</i> .
Name	Specifies the name of a POI POIs can have one or more names in one or more languages, see Section <i>Name Attribute for POIs</i> on page 260.
regionId	References <ul style="list-style-type: none"> <li>– The country region: In this case, the region ID can be used for country-based searches</li> <li>– The state region within a country (optional reference): In this case, the region ID can be used for country- or state-wide searches</li> <li>– The logical area within a state or a country (optional reference): In this case, the region ID can be used for POI searches in any administrative area per country or state</li> </ul> The reference to the region ID may also be set to <b>NULL</b> . In this case, a POI is not related to any region and will, thus, not be found in region-based searches. Related topics: <ul style="list-style-type: none"> <li>– POI regions (and logical areas): Section 13.6.2 <i>Relations between POI and Regions</i> on page 273</li> <li>– Using <b>regionId</b> in searches: <i>NDS – Compiler Interoperability Specification</i>, 10.3 <i>Use of PoiTable.regionId in Searches</i> on page 187</li> </ul>

Attribute	Description
postalCode	Defines the postal code of an POI; stored in <b>PoITable</b>
phone	Stores a phone number for the POI using the international E.123 notation ( <a href="http://en.wikipedia.org/wiki/E.123">http://en.wikipedia.org/wiki/E.123</a> ). Stored in <b>PoITable</b> .

### Name Attribute for POIs

POIs can have one or more names in one or more languages, so-called name strings. A POI name string can be a default name, an official or alternate name, and can have synonyms and transliterations.

Thus the POI building block uses the same data structures as the Name building block. However, the name strings for POIs are not stored in the Name building block of an NDS database, but in separate tables in the POI building block.

For more information about the Name building block, refer to Chapter 10 *Name Building Block* on page 163.

The name strings for POIs are stored in the following name tables:

- **PoiToStringTable**
- **PoiNameStringTable**
- **PoiNameStringRelationTable**

### 13.2.2 Secondary POI Attributes

Secondary POI attributes can be assigned to POIs and are stored as type-value pairs in the attribute BLOB of the POI table. The valid types of secondary POI attributes are defined in a BLOB in the **supportedAttributes** field in the POI metadata.

The type definition of a secondary POI attribute consists of the following information:

- Attribute type ID
- Filtering attribute flag: Can be used by the application to determine if the attribute contains only display information or information which can be used as a filtering criterion; the latter enables the application to show the attribute on the user interface for defining search filters.
- Reference to an icon (optional)
- Value format: Defines the format of an attribute value (for example, the Opening Hours attribute has the value type Date-time Range)
- Definition of enumeration values (only available if the value format is ENUMERATION)
- Flags indicating whether enumeration values have names and icons (optional; only available if the value format is ENUMERATION; the payment method attribute, for example, can have the predefined values Cash and Credit Card, each represented with an icon)

The definition of a predefined value of a secondary POI attribute consists of the following information:

- Value ID
- Reference to an icon (optional)

### Grouping of Secondary POI Attributes

Secondary POI attributes may be grouped to supply additional information, for example, to add time dependency information to price information in order to store price variations over time. It is not possible to group attributes that are shared by several POIs.

Secondary attributes are grouped by means of the attribute group ID field.

### Name Strings for Secondary POI Attributes

Each attribute type and each attribute value for secondary POI attributes may have a set of names in multiple languages. The names are stored in the following tables:

- PoiAttrToStringTable
- PoiAttrNameStringTable
- PoiAttrNameStringRelationTable

### Predefined Secondary POI Attributes

NDS does not impose the use of specific secondary POI attributes. Several commonly used secondary POI attributes, however, are predefined to ensure interoperability of NDS databases (DataScript location: `nds.poi.predefined`). Their use is optional for NDS databases.

*Table 13-4 Examples for predefined secondary POI attributes*

Attribute	Comment
Address (POIATTR_ADDRESS)	Address of the POI, usually an official address  A POI may contain several instances of this attribute, for example, if the POI is located on an intersection of two streets and has one address attribute for each street.
Activation (POIATTR_ACTIVATION)	Contains the value of the activation distance in meter
Brand (POIATTR_BRAND_NAME)	Name of a brand, usually used in combination with a reference to an icon set representing the brand
Food type (POIATTR_FOOD_TYPE)	Specifies a food type, for example, Mexican; usually used for restaurant category POIs
Opening hours (POIATTR_OPENING_HRS)	Specifies the opening hours for a POI; can also be used to indicate that a POI service is available non-stop (24-hour indicator)
Multimedia (supplementary) (POIATTR_MULTIMEDIA)	Reference to a data source containing multimedia content, for example, a video on the local file system or an image on the Internet (see Section <i>Multimedia Secondary POI Attributes</i> on page 262)
WebSite (POIATTR_WEBSITE)	Reference to the website containing information related to the POI

## Multimedia Secondary POI Attributes

For secondary POI attributes of multimedia type, references to a multimedia data source using an URI are stored in the secondary POI attribute BLOB. The URI can refer to an external file or other data sources.

The URI has to comply with the scheme defined in the list of official URI schemes (see <http://www.iana.org/assignments/uri-schemes.html> and [http://en.wikipedia.org/wiki/URI\\_scheme](http://en.wikipedia.org/wiki/URI_scheme)). The possible data format is defined by MIME media types (see <http://www.iana.org/assignments/media-types/>).

*Table 13-5 Examples for multimedia secondary POI attributes*

Name of the multimedia attribute	MIME type (example)	Value (example)
POIATTR_MULTIMEDIA	image/jpeg	<a href="http://www.xyz.com/photo1.jpg">http://www.xyz.com/photo1.jpg</a>
POIATTR_MULTIMEDIA	audio/midi	<a href="http://www.xyz.com/about.jsp?anouncment=56">http://www.xyz.com/about.jsp?anouncment=56</a>
POIATTR_MULTIMEDIA	video/mpeg	<a href="file://multimedia/database_video?table=MovieTable&amp;id=5">file://multimedia/database_video?table=MovieTable&amp;id=5</a>
POIATTR_MULTIMEDIA	video/mpeg	<a href="file:///videos/introduction.mpg">file:///videos/introduction.mpg</a>

## Bitmask Approach for Filtering Secondary POI Attributes

To improve performance, it is possible to indicate for each POI, which attributes are actually stored in the list of secondary attributes (`PoiTable.attributeBlob`). In the so-called attribute bitmask (`PoiTable.attrBitMask`), each bit represents one secondary attribute or one enum value of a secondary attribute.

The mapping of bits in the attribute bitmask to secondary attributes or enum values of the secondary attribute is stored in the POI metadata (DataScript location: `nds.poi.metadata > PoiMetadataTable.AttrBitMaskDefinition`).

If the application searches for a specific secondary attribute or an enum value of a secondary attribute, it proceeds as follows:

1. The application checks if the indicator has been defined in the attribute bitmask metadata (`AttrBitMaskDefinition`).
2. If a definition is available, the application creates an integer number, which has a bit that corresponds to the indicator position set to 1.
3. The application can now use the integer number in a query of the `PoiTable` to filter for POIs with the respective secondary attribute or the respective enum value of the secondary attribute.

### 13.3 Relations between POIs

POIs can be related to other POIs. The relation between the two POIs is a parent-child relation in form of a DAG and has one of the following relation types assigned:

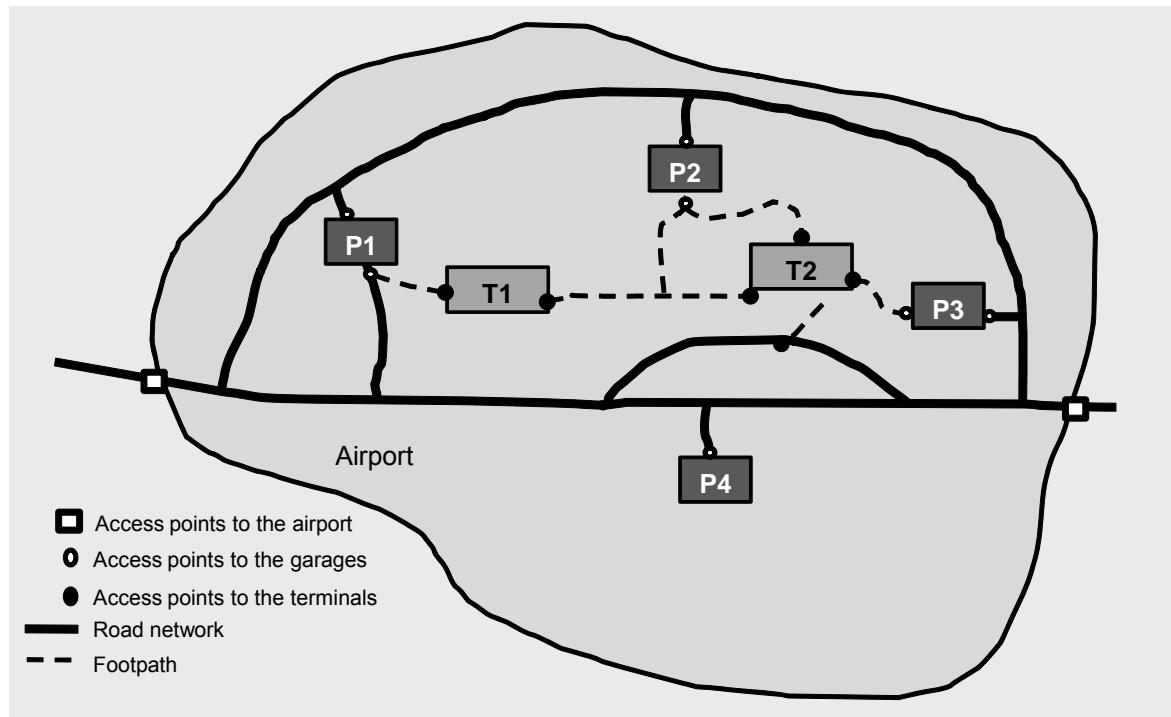
- PARTOF: POI is a physical part of another POI
- ACCESS: POI represents an access way to another POI
- LOGICAL\_ACCESS: POI represents a logical access point to another POI
- GUIDANCE\_POINT: POI represents a guidance point and can be used for „intuitive“ route guidance
- ACTIVATION: POI represents an activation point which enables applications to show information on a POI before passing the POI
- PREFERRED: One or more children of a POI are preferred compared to other children of the same POI. A restaurant POI, for example, can have five parking places as children and two of them have the relation type PREFERRED because they are closer to the restaurant than the other parking places.

In DataScript, the relation types are defined in `nds.poi.main > PoiRelationType`. The individual types will be explained in more detail in the following sections.

### 13.3.1 Part-of Relations between POIs

A POI can be a physical part of another POI. A parking lot can, for example, belong to an airport terminal and the airport terminal to an airport. The relation makes it possible to offer the user a list of terminals or parking facilities to refine the search after selecting an airport. An example is shown in Figure 13-2.

Figure 13-2 Relations of POIs at an airport



A POI may have a part-of relation to more than one POI. Part-of relations between POIs are stored in the **PoiRelationTable** (DataScript location: `nds.poi.main`). For more information, refer to the *NDS – Physical Model Description*.

POIs with a part-of relation can also be accessed independently. This is explained in the following example.

#### Example

A parking lot is related to a monument POI, because it is the most suitable parking lot for this monument. The parking lot is not restricted to visitors of the monument, but is also open for parking in general. Therefore, it is also assigned to the POI category for parking lots.

If the use of a parking lot, however, is restricted to visitors, it is still related to the monument POI, but either contains an attribute indicating private parking or it is additionally related to the private parking POI category.

### 13.3.2 Access Relations between POIs

A POI represents a real-world object with a geographic location. The geographic location is often used for location input. In some cases, however, a POI can only be approached using a gate, a path, a ramp, and so on, which is physically displaced from the geographic location of the POI.

Physical objects that can be used to approach a POI are called *access points*. Therefore, a user has to be guided to an access point first, and then to the POI. Figure 13-2 shows two access points for the airport and further access points for the parking lots and the terminals.

There are two types of access points:

- Simple access points have one attribute, either a geographic location or a reference to the routing network. They are either stored in the `PoiGeoAccess` table (for geographic locations) or in the `PoiLinkAccess` and `PoiIntersectionAccess` tables (for references to routing features).
- Complex access points have additional attributes, for example, type (parking, gate, garage, entry), use (public, private, staff-only, VIP), opening time, entrance fee, and so on. Complex access points are stored as POIs in the `PoiTable` with the relation `ACCESS` to a parent POI and the POI category `POICAT_ACCESS_POINT`.

Both simple and complex access points are optional.

The rules for filling the POI access tables are described in the *NDS – Compiler Interoperability Specification, 10.6 POI Access Tables* on page 188.

### 13.3.3 Logical-access Relations between POIs

Logical access points are optional and are used for searching POIs within a corridor stretching along a route. Logical access points are pre-calculated at compilation time of the POI database.

Logical access points reduce the calculation effort for the application for extra routes to the POIs along the route. Without logical access points the application has to estimate which POIs are along the route based on the air distance. This would either be inaccurate, or route calculation to all potential POIs would be necessary and cause a reduced performance. Each logical access point can be related to:

- One or more POIs that can be reached from that logical access point (see „logical access point 01“ and „logical access point 02“ in Figure 13-3)
- One routing link (typically a freeway exit)

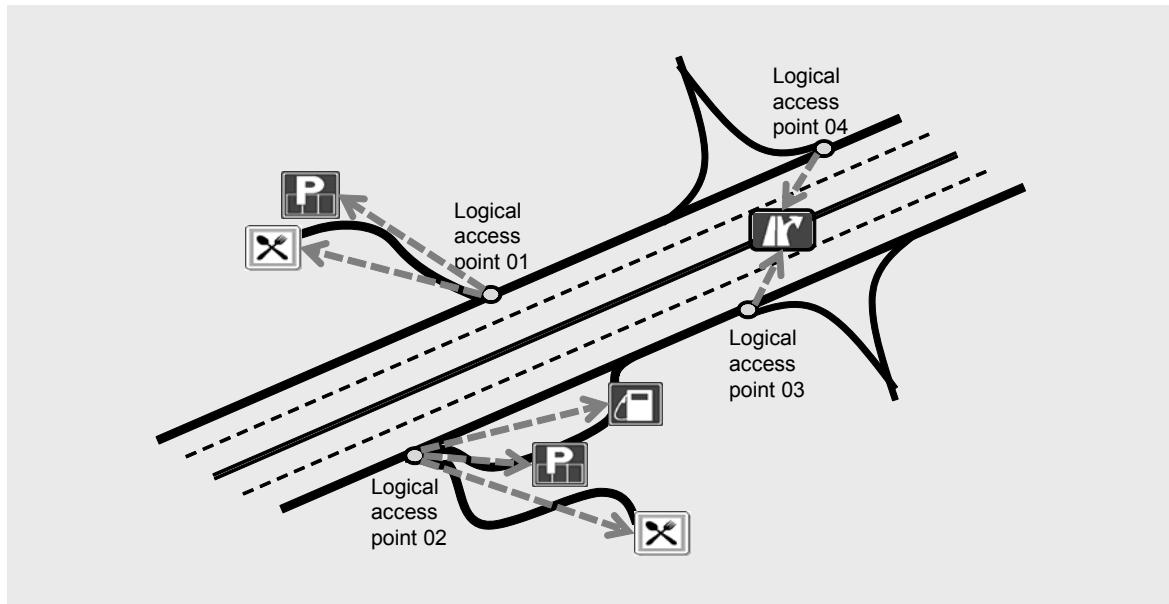
POIs can have one or more logical access points as depicted in Figure 13-3.

Logical access points are stored in the POI table with the `LOGICAL_ACCESS` relation for each parent POI that can be reached from the logical access point.

The POI category is defined as a logical access point. The references from POIs with a logical access relation to routing features are stored as simple access points (see Figure 13-2).

Logical access points shall not be displayed on the map and not be used for location input. They provide additional points on the calculated route helping the application to find POIs along this route.

Figure 13-3 Logical access points and POIs on a freeway



### 13.3.4 Guidance-point Relations between POIs

Guidance points are optional and may be used for giving „intuitive“ route guidance, such as „Turn left after the yellow gas station“. POIs suitable for guidance may be assigned to the category `POICAT_GUIDANCE_POINT`. The position of a guidance POI indicates the point where the application is to output the guidance information.

If the guidance information is to be given at an additional point, for example, at the opposite side of an intersection, the guidance POI may be related as a parent to another POI, using the `GUIDANCE_POINT` relation. The position of the child POI indicates the point where the application can additionally output guidance information.

Guidance-point relations between POIs are stored in the `PoiRelationTable` (DataScript location: `nds.poi.main`). For more information, refer to the *NDS – Physical Model Description*.

### 13.3.5 Activating POIs along a Route

The POI activation function enables navigation systems to provide information about POIs before reaching the actual POI. Activation points and activation attributes are optional.

The POI activation function can be realized by one of the following methods:

- Assigning an activation point to a POI

An activation point is a POI nearby the *activated POI*, that is, a POI with an assigned activation point. An activation point contains attributes like a video clip, or other multimedia, with information about the activated POI. Activation points can, for example, be used to play announcements before reaching famous buildings or other landmarks.

Activation points are stored as POIs in the POI table and have the following properties:

- POI category activation point (**POICAT\_ACTIVATION\_POINT**)
- Relation **ACTIVATION** to the activated POI

- Assigning an activation attribute to a POI

The activation attribute is a secondary attribute of an activated POI and contains a default value for the default activation distance in meter.

- Assigning a non-zero value to the **activationRadius** field for a category stored in the **PoiCategoryTable**

The value represents the activation radius in meter. The application activates the POIs related to this category when a geographic location within the activation radius is reached.

## 13.4 POI Categories

Due to the large number of POIs, NDS provides the POI category property for the POI feature class. There are two types of POI categories:

- Super-categories: Summarize POI categories under a generic term
- Leaf categories: Identify POI categories that do not have any subsidiary categories

---

**Example** The POI super-category **Vehicle** contains all POI categories thematically related to vehicles, such as petrol stations, car dealers, car repair, car wash.

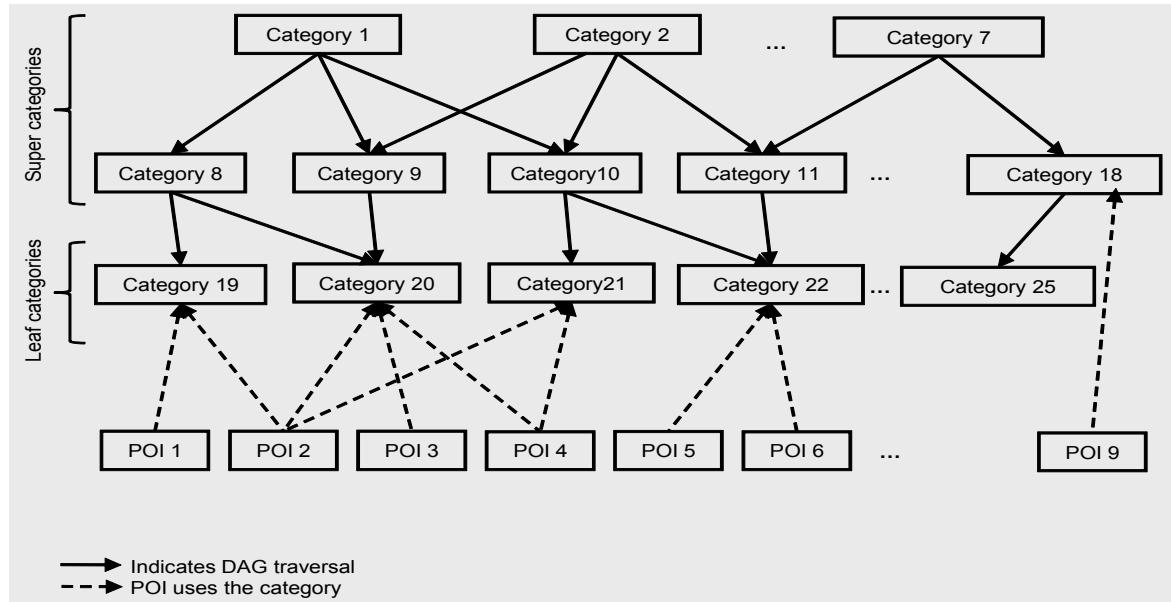
---

For a full list of standard categories, refer to Physical Model under `nds.poi.predefined > StandardCat`.

POIs may be assigned to multiple categories. A church, for example, may be assigned and be found respectively in the POI categories Culture and Sights. A POI category may also be assigned to more than one super-category. The hierarchy is represented by a directed acyclic graph (DAG). This is illustrated in Figure 13-4.

The user starts the POI selection at an entry point (root) and traverses the DAG until the user reaches the desired POI. In the levels in between, the user can refine the search by choosing one of the categories.

Figure 13-4 Directed acyclic graph of categories and POIs



The POI categories represented in the DAG are stored in the `PoiCategoryTable`. The hierarchical relations between the categories are stored in the `PoiCatRelationTable` in form of parent-child relations. Table 13-6 shows the entries in the `PoiCatRelationTable` table.

Table 13-6 `PoiCatRelationTable` table

Field	Description
<code>parentCatId</code>	Reference to the category in the <code>PoiCategoryTable</code> representing the parent category The value must be set to <b>NULL</b> for root categories.
<code>childCatId</code>	Reference to the category in the <code>PoiCategoryTable</code> representing the child category The field value must <b>not</b> be <b>NULL</b> .

The relations between POIs and POI categories are stored in the `PoiServiceLocationTable`. This table combines the categorization of POIs with their position on the map.

### Related Topics:

- 13.6.1 *POI Service Locations* on page 273

### 13.4.1 Attributes of POI Categories

Table 13-7 shows the attributes for POI categories.

*Table 13-7 Attributes for POI categories*

Attribute	Description
Category ID (catId)	Uniquely identifies the POI category
Name	Name of the POI category in one or more languages Each category may have several name strings in several languages, see Section <i>Names for POI Categories</i> on page 269.
Standard category (standardCat)	Specifies predefined categories For more information, refer to Section <i>POI Standard Categories</i> on page 270.
Selection entry (selectionEntry)	Indicates whether the POI category can be used as the entry point for POI category selection in a navigation system
Scale level ID (scaleLevelId)	Reference to the scale range in the <i>PoiScaleLevelTable</i> table This is a recommended scale range for the display of icons. For more information, refer to: <ul style="list-style-type: none"><li>- Section 13.5 <i>Representing POIs as Icons</i> on page 270 and Chapter 20 <i>Icons</i> on page 355</li><li>- <i>NDS – Compiler Interoperability Specification, 10.7 Scale Ranges for POI Categories and POI Icons</i> on page 189</li></ul>
Icon set ID (iconSetId)	Reference to the icon set representing this category An icon can be selected from the icon set based on the scale and icon use type.
Activation radius (activationRadius)	Defines the activation radius in meters for the POIs in this POI category For more information, refer to Section 13.3.5 <i>Activating POIs along a Route</i> on page 266.
isProtected	Indicates whether a POI update or import can change the content of this category. This attribute is used for third-party data.

### Names for POI Categories

POI categories have a name (for example, airport) for which several name strings in one or more languages may be available. A POI name string may be a default name, an official or alternate name, and may have synonyms and transliterations.

Thus, the POI building block uses the same data structures as the Name building block. However, the name strings for POIs are not stored in the Name building block of an NDS database, but in separate tables in the POI building block.

For more information about the Name concept for NDS, refer to Chapter 10 *Name Building Block* on page 163.

The name strings for POI categories are stored in the following tables:

- PoiCatToStringTable
- PoiCatNameStringTable
- PoiCatNameStringRelationTable

### POI Standard Categories

NDS does not impose the use of specific POI categories. However, several commonly used POI categories and a generic POI category (POICAT\_NDSGENERAL) are predefined to ensure interoperability for NDS databases. Their use is optional, but the predefined identifiers must not be used for non-predefined categories.

Also, POIs that semantically belong to one standard category, for example, restaurants, must be assigned to these, even if custom categories are used. In this case, these POIs must be assigned to both, the standard and the custom category. This is established in the *Respecting the standard* rule, see Section *Rules for Extensions* on page 56.

---

**Example** Examples for POI standard categories are:

- Airport
- Restaurant
- Hotel
- Parking
- Activation point
- Access point

---

For more information on POI standard categories, refer to *NDS – Compiler Interoperability Specification, 10.5 POI Standard Categories* on page 188.

## 13.5 Representing POIs as Icons

POIs can be represented by an icon on the map display. POI icons are stored in the `poiIconTable` table, which is an instance of the `IconTable`, see Chapter 20 *Icons* on page 355.

POI features reference an icon set stored in the `poiIconTable` table, and not a particular icon. An icon set consists of a number of icons with the same “logical” meaning, but with different physical properties, for example, different pixel sizes or coloring schemes (day and night).

An icon set for an airport category, for example, can consist of several icons of different pixel sizes for different map sizes, but all depict an airport symbol.

When drawing the icons on the map, the application selects one of the icons from the icon set. The selection bases on multiple criteria, such as the zoom level and/or user preferences. NDS provides the filtering criteria for icon sets to support the application in the selection. The filtering criteria are stored in the `poiIconTable` table.

### 13.5.1 Drawing Icons for POIs

A POI may be related to one or more icons:

- Icons for POI category
- Icons for secondary attribute and value (optional, only if they have associated icons)
- POI-specific icons stored as a reference to an icon set in `PoiServiceLocationTable.iconSetId`.

The application's user interface and the user preferences decide which icons to draw. As a general guideline, NDS recommends that one icon is selected as the main representative for a "POI as a service". The main representative is usually the icon related to the corresponding POI category. If available, the main representative can also be:

- The icon associated to a brand or chain attribute
- The icon associated with a secondary POI attribute

Additional icons, such as the 24-hour indicator, can be displayed on the map to give additional information to the user.

### 13.5.2 Overlapping POI Icons in Map Display

To define a drawing priority for overlapping icons in map display, the following parameters of the `IIconTable` can be used:

- `iconDisplayArrangement`

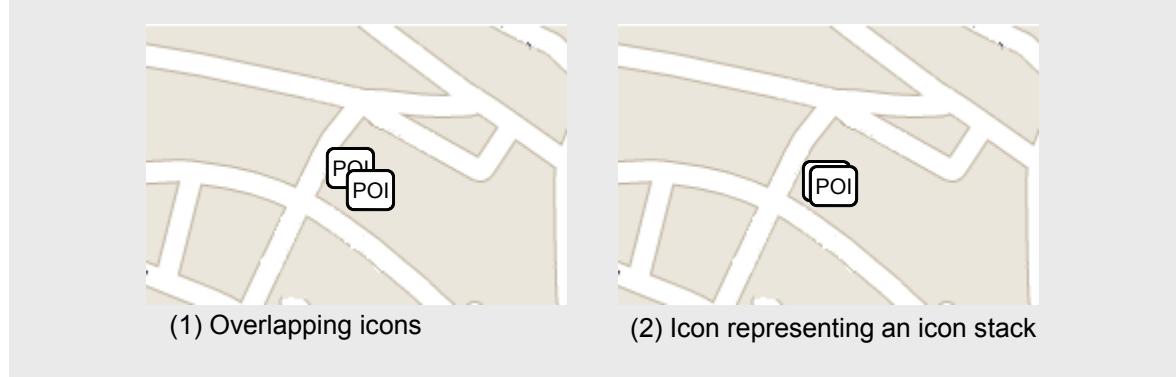
This parameter is used to change the arrangement of POI icons on the map display. NDS provides the following options for icon display arrangement:

- `STACKABLE`: A specific icon denoting an icon stack represents the overlapping icons.
- `DISPLACE`: The icon with the highest drawing priority replaces the overlapping icons.
- `OVERLAP`: The icons overlap according to their drawing priority.
- `NO_ARRANGEMENT`: No specific arrangement for overlapping icons is defined.

- `iconDrawingPriority`

This parameter is used to define a drawing priority for overlapping POI icons in map display. High drawing priority is indicated by high numbers. The parameter value `DEFAULT_DRAW_PRIO` is used if no drawing priority is set.

Figure 13-5 Example for overlapping icons and an icon representing an icon stack



## 13.6 Accessing POIs for Location Input and Map Display

The POI building block provides different data structures for accelerating map display, and regional POI searches, as well as enhancing the destination input via POIs.

- Accelerated map display for POIs and POI categories by means of the POI service location table, see Section 13.6.1 *POI Service Locations* on page 273
- Spatial search for POI using a bounding box-based query: For example, the application can find a named object in the Name building block and its bounding box for any given geographical area. The application converts the bounding box either into a query that bases on Morton codes, or to a set of virtual POI tiles and finds all related POIs. The result may contain more POIs than expected, but as this is a minor side effect, it is usually acceptable.

For information on virtual POI tiles, see Section 13.6.3 *Virtual Tiles for POIs* on page 277.

For information on generating IDs according to Morton code, refer to *NDS – Compiler Interoperability Specification*, 3.1.1 *Generating IDs According to Morton Order* on page 55.

- Regional search using the relation between POIs and geographic areas. See Section 13.6.2 *Relations between POI and Regions* on page 273.
- Selecting POIs from a category tree, filtering POIs by their category, see 13.4 *POI Categories* on page 267.
- Destination input by means of NVCs, see section 13.6.4 *Next-valid Character Trees for POIs* on page 278
- Full-text search, see Chapter 19 *Full-text Search Building Block* on page 345

### 13.6.1 POI Service Locations

The relations between POIs, POI categories, POI icons, and geographic positions are stored in the `PoiServiceLocationTable` (DataScript location: `nds.poi.main`). This information is sufficient to quickly display POIs as icons in a map.

If virtual tiles are used for the POI building block (see Section 13.6.3 on page 277), POI service locations can be stored on different levels. This enables applications to quickly load all service locations belonging to a specific tile on a specific level. This facilitates selective display of POIs at a small map scale, for example, display of airports on overview maps.

### 13.6.2 Relations between POI and Regions

The `regionId` attribute can be used to search POIs country-wide or state-wide. POIs can, optionally, be searched within any given geographic area by using the relations between POIs and geographic areas.

#### Explicit Relation between POIs and Geographic Areas

Geographic areas can belong to one of the following area types:

- Administrative areas. Administrative areas are part of the administrative hierarchy (see Section 10.2 *Named Object Features* on page 167). Areas from any arbitrary pair of these areas either have a contained-in relation or do not intersect. Furthermore, the complete map is covered by these areas.
- Named areas. Named areas are not part of the administrative hierarchy. They may intersect with each other or with administrative areas.

Figure 13-6 depicts possible relations between administrative and named areas.

Figure 13-6 Relations between administrative and named areas

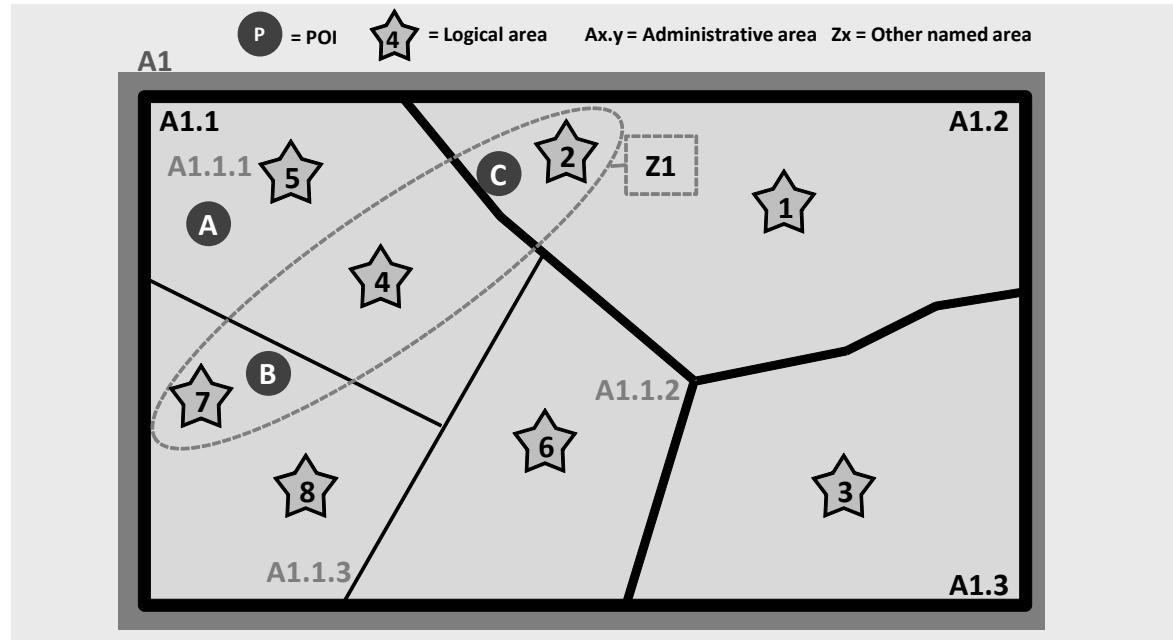
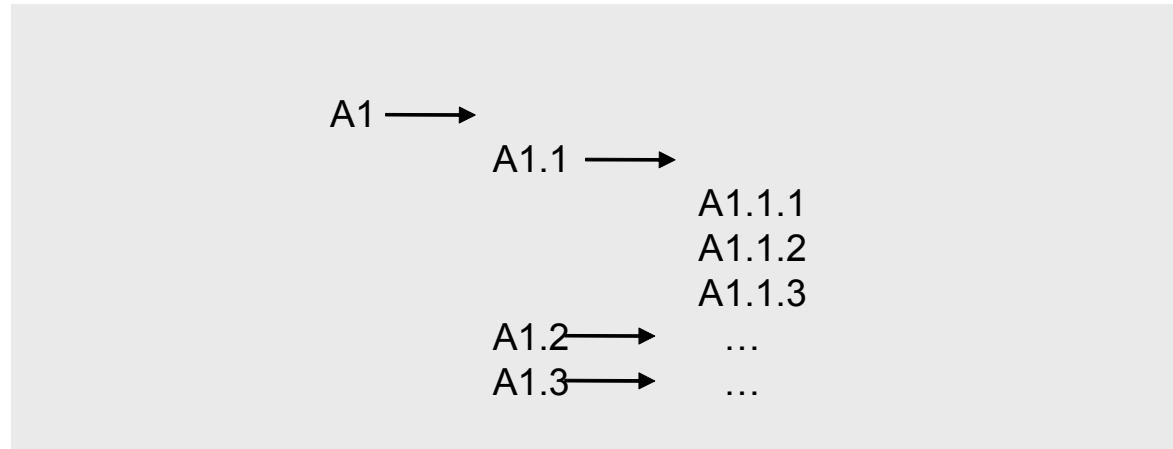


Figure 13-7 illustrates the administrative areas formed by the following administrative hierarchy:

Figure 13-7 Administrative hierarchy for the example in Figure 11-5



The area Z1 is a named area and therefore does not belong to the administrative hierarchy. In Figure 13-6, it intersects the administrative areas A1.2, A1.1.1 and A1.1.3.

It is assumed that a POI is always located in an area at the lowest administrative level that covers the whole map. This is usually the level that consists of order 8 areas. In Figure 13-6, POI "A" is located in administrative area A1.1.1, POI C is located in administrative area A1.2. POI B is located in administrative area A1.1.3 and in named area Z1.

POIs, therefore, are in many-to-many relationships to geographical areas. For an optimized search performance it is necessary to avoid this relation type. A POI shall only be related to one area. This is achieved by introducing a set of logical areas which are mutually disjunctive. These logical areas are called *POI regions*.

### POI Region

A logical area is defined as the result of a unique combination of administrative areas on the lowest level of the administrative hierarchy and named areas for the corresponding map. A set of logical areas is generated as follows:

- Splitting of all administrative areas on the lowest level of the administrative hierarchy and named areas
- Identification of all disjunctive areas resulting from the splitting

The set of logical areas forms a partitioning, that means, the following must be true:

- Two logical areas do not overlap, except at their borders
- The disjoint union of all logical areas equals the extent of the original map.

---

<b>Note</b>	Postal areas are not included in a unique combination of areas. The number of regions resulting from this would be too large.
-------------	---

---

The logical areas in Figure 13-6 are labeled with stars:

- Logical area 3 matches administrative area A1.3
- Logical area 6 matches administrative area A1.1.2
- Logical area 5 is only a part of administrative area A1.1.1
- Logical area 4 is a part of the area A1.1.1 and is also part of the named area Z1

A POI region can either be a country, a state within a country, or a logical area within a state or a country. A POI is related to only one region.

### Relation between POI Regions and Named Objects

The administrative areas are defined in the Name building block. For generating a set of POI regions, the named objects of the administrative area feature class are used. For integrated POI building blocks, the existing Name building block shall be reused. For non-integrated POI building blocks, a separate Name building block (with its own administrative structure) shall be stored in the product database provided for the POIs.

The storage method is determined in `PoiMetadataTable.adminStructureType`. The POI building block may reference country and state regions in the Name building block only, or it may also reference other administrative and non-administrative regions. This is determined in `PoiMetadataTable.hasAdvancedAdminSearch`:

- If set to FALSE, country and state regions may be referenced.
- If set to TRUE, any administrative or non-administrative area may be referenced.

## Virtual POIs

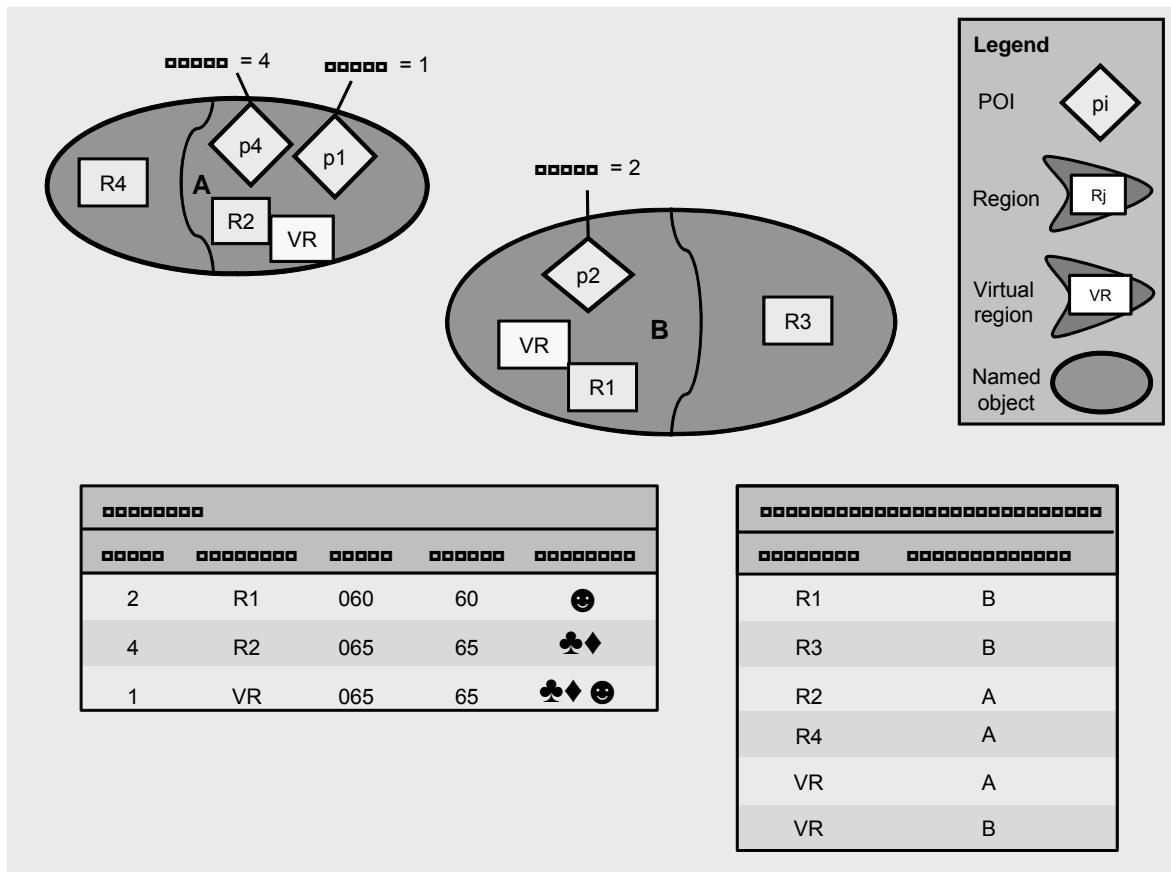
It may be necessary to establish a virtual relation between a POI and a named object, meaning that a POI physically belongs to one named object but logically also belongs to another named object. This applies, for example, to airport POIs: P1 is physically located in city A, and must be found when searching for airports in city A. City B is located nearby city A. Therefore, P1 must also be found when searching for airports in city B. In this context, POI P1 is usually referred to as a *Virtual POI* or *Vanity POI*.

A POI is related to exactly one region, whereas a region can be related to one or more named objects. Proceed as follows to store a POI which shall be found in two or more named objects:

1. Define a "virtual" region VR and assign a unique ID to this region.
2. Store a relation between the virtual region and each of the relevant named objects in the `PoiNamedObjectRelationTable` table.
3. Assign the POI to the virtual region by storing the ID of the virtual region in the `poiTable.regionId`.

As a result, the POI is represented with a single record in the `poiTable` table and can thus be found in each of the relevant named objects.

Figure 13-8 Vanity POI



### 13.6.3 Virtual Tiles for POIs

Tiles are only available for the building block types in which data is stored BLOB-oriented, see also Section *Tiles and Building Blocks* on page 92.

The POI building block stores its data relationally, meaning the features are stored individually as single rows in the database. This facilitates updates and specific search scenarios. For accessing all POIs within a specific geographic area, however, the tile concept would be more favorable.

For this reason, NDS provides the concept of *virtual tiles*, which may be used for the POI building block. Virtual tiles leverage the property of SQLite to physically store rows sorted according to their primary key. In order to apply the virtual tile concept to POIs, the key in the POI service location table (DataScript location `nds.poi.main > PoiServiceLocationTable.serviceLocationId`) must be assigned according to Morton order. If this is the case, the virtual tile table (DataScript location: `nds.poi.main > PoiVirtualTileTable`) can associate each packed tile ID with a range of service location IDs (`minId` and `maxId`) which represent all POIs located in that tile. The virtual tile table can be used in SQL queries to load all POI service locations belonging to a given geographic tile in an efficient way.

The `PoiVirtualTileTable` is mandatory. It shall contain only one level. For this level, the Boolean `hasPoiVirtualTiles` must be set in the level metadata (DataScript location: `nds.overall.metadata > LevelMetadataTable`).

### 13.6.4 Next-valid Character Trees for POIs

POI NVC trees use the NVC data structures of the Name building block; however, the POI NVC trees themselves are stored in the POI building block (`poiNvcTable`). The nodes of POI NVC trees refer to POIs. This is indicated by the `isPoiNvcTree` flag, which is set to TRUE for this type of NVC tree. The referenced POIs may belong to different POI categories. For detailed information on the structure and properties of NVC trees, refer to Section 10.6 *Next-valid-character Trees* on page 186.

Table 13-8 shows how the NVC tree data structures are used for POIs.

Table 13-8 Data structures for POI NVC trees

Data	DataScript location	Description
<code>categoryId</code>	<code>nds.name.nvctree</code>	The <code>categoryId</code> is normally enclosed in <code>nvcPoiReference</code> , but used directly for the <code>ShortLeafNode</code> . It gives information on the categories of the POIs referenced by an NVC node.
<code>PoiCategoryDescription</code>	<code>nds.name.nvctree</code>	Defines a tuple of a POI category ID with the number of remaining names in this category.  The number of remaining names can be used, for example, to show the user the number of POIs in a category when the user searches the POI NVC.
<code>nvcPoiReference</code>	<code>nds.name.nvctree</code>	Reference from the NVC node to the POI. The references use the ID, one or more categories of the POI, and general properties of a NVC reference (for example, number of contained names).

Data	DataScript location	Description
poiId	nds.name.nvctree	The poiId is normally enclosed in nvcPoiReference, but used directly for the ShortLeafNode.

## 13.7 Versioning of POIs

Due to a higher change rate, POIs need to be updated frequently either individually or as a group. To support the POI update, NDS defines a mandatory version per POI. Additionally, it is possible to optionally define a version per POI group.

Table 13-9 Versioning information for POIs

Versioning information	Description
POI version	<p>Version ID of a POI, stored in the <code>PoiTable</code></p> <p>The POI data provider provides a new version number whenever one of the POI attributes in the POI or POI names table changes.</p>
POI virtual tile	<p>POIs may be grouped based on virtual tiles. The POI tile version information is stored in the <code>PoiVirtualTileTable</code>. It contains a group identifier (packed tile ID), the version ID, and the information whether the group is partially updated.</p> <p>For more information, refer to <i>NDS – Update Specification</i>, 4.1 Versioned Items on page 33.</p>

For more information about updating NDS databases in general, refer to the *NDS – Update Specification* document.



## 14 Speech Building Block

The Speech building block is used to store speech data for names that can be retrieved from the Name, Traffic Information, Routing, or POI building blocks. It contains two different types of speech data:

- Phonetic transcriptions, see Section *Phonetic Transcriptions* on page 281
- Prerecorded voice fragments, see Section *Prerecorded Voice Data* on page 282

The Speech building block is optional and may be omitted.

### Phonetic Transcriptions

Phonetic transcription describes the representation of speech sounds using symbols in phonetic alphabets, such as the International Phonetic Alphabet (IPA). Such alphabets are necessary, as most languages show a discrepancy between their official orthography and the actual pronunciation.

As navigation systems mainly deal with proper names which often have an irregular pronunciation, phonetic transcription is an integral part of navigation databases.

Navigation systems use phonetic transcriptions for the following purposes:

- Converting text to speech, for example, for route guidance messages or location input (Text-to-Speech (TTS))
- Matching the user's speech input with database items, for example, for location input (Automatic Speech Recognition (ASR))

Integrated phonetic transcriptions in NDS need to be correct, consistent and coherent. Therefore, NDS requires a binding phonetic transcription specification for such data. NDS already provides the following phonetic transcription specifications:

- Language tags to support multilingual applications
- Language-specific versioning mechanism

In addition, NDS database suppliers must provide a document containing admissible phonetic symbols with the following information:

- One table of (usually language-independent) suprasegmental elements for stress marks, syllables, and word boundaries.
- For each language a table containing admissible phonetic symbols. For each of these symbols, the following information has to be provided:
  - Associated hexadecimal Unicode scalar values
  - Description of the phonetic features (for example, voiceless bilabial plosive)
  - At least one example (orthography + phonetics) demonstrating the use of each symbol in a wider context

The tables can be used by developers to adjust their implementations. They are not part of the NDS data, but shall be delivered in form of a document.

Due to the variety of speech recognition engines available on the market, the Speech building block supports only basic phonetic transcription storage. Engine-specific optimization structures, such as search trees, may be added as a proprietary add-on.

NDS supports the following phonetic notation systems for phonetic transcription:

- International Phonetic Alphabet (IPA)
- (Extended) Speech Assessment Methods Phonetic Alphabet ((X)-SAMPA)

All SAMPA-based symbol sets comprise machine-readable characters (7-bit ASCII in particular), while IPA uses a specific mix of partially modified Greek and Latin characters. Both, IPA and SAMPA define a set of available symbols. The phonetic transcription, that is, the interpretation and the use of each symbol in specific contexts, is defined separately in phonetic specifications. In NDS, these phonetic specifications are stored in separate databases as described in Section 14.1 *Building Block Structure and Content* on page 282.

### Prerecorded Voice Data

Prerecorded voice data can be used for speech output by navigation systems, especially for route guidance. There are two types of prerecorded voice fragments:

- Common data: General driving instructions used repeatedly, such as „turn left“ or „keep right“
- Individual data: Names of intersections, squares, and roads as well as signpost names

NDS supports both common and individual voice data in the following formats:

- ADPCM (Adaptive Differential Pulse Code Modulation, see also [http://en.wikipedia.org/wiki/Pulse-code\\_modulation](http://en.wikipedia.org/wiki/Pulse-code_modulation))
- MP3

## 14.1 Building Block Structure and Content

Name strings for which a phonetic transcription or prerecorded voice data is available, are referenced by the corresponding phonetic transcription or voice data. This reference direction decouples the Speech building block from the Name, Traffic Information, Routing, and POI building blocks. As the Speech building block is optional, its availability has to be indicated in the `UrBuildingBlockVersionTable`.

Several database files can be available in one NDS product database in order to support phonetic transcriptions for several speech engines, as well as different formats for prerecorded voice data. For this, the following types for the Speech building block are defined:

- In `speech > common .ds`, the `PhoneticTranscriptionFormat`, defining the format of the phonetic transcriptions used:
  - X-SAMPA
  - LH+
  - StarRec-SAMPA

- TA-IPA
  - NT-SAMPA
  - SVOX-PA
- In `speech > common.ds`, the `PrerecordedVoiceFormat`, defining the format of the prerecorded voice data:
- ADPCM
  - MP3

Table 14-1 shows the data contained in the Speech building block.

*Table 14-1 Data in Speech building block*

Type of data	Description
Phonetic transcription	<p>The exact definition of a phonetic transcription depends on the type of transcription, meaning whether it is a transcription for a name string of the Name building block, a POI name, a TMC event, or another name. For more information about the physical model, refer to the <i>NDS – Physical Model Description</i> (DataScript location: <code>nds.speech &gt; phonetictranscription</code>).</p> <p>The definition may include:</p> <ul style="list-style-type: none"> <li>- ID string representation</li> <li>- Language code</li> <li>- Speech type, such as           <ul style="list-style-type: none"> <li>NOT_AVAILABLE (phoneme has no type)</li> <li>FIRST_APPROPRIATE (first appropriate standard pronunciation)</li> <li>STANDARD (alternative standard pronunciation)</li> </ul>           Speech types are defined in <code>nds.speech &gt; common</code> </li> <li>- Order of phonetic transcriptions</li> <li>- Grammatical case of the phonetic transcription; enables a TTS application to identify the correct transcription for declined names. For more information, refer to <i>Grammatical Cases</i> on page 286.</li> <li>- ID of the associated entry in the <code>PhoneticTranscriptionAuxTable</code> containing the combinations of preposition and definitive article for specific use cases. For more information, refer to <i>Articles and Prepositions</i> on page 286.</li> <li>- For transcription of name strings: same information as for name strings, such as name use type, name string relation, etc</li> </ul> <p>A phonetic transcription refers to one or more names, see Section <i>References from Phonetic Transcriptions and Prerecorded Voice Data</i> on page 284.</p>

Type of data	Description
Prerecorded voice data	<p>The exact definition of a prerecorded voice record depends on the type, meaning whether it is a voice record for a name string of the Name building block, a POI name, or a TMC event. For more information about the physical model, refer to the <i>NDS – Physical Model Description</i> (DataScript location: <code>nds.speech &gt; prerecordedvoice</code>).</p> <p>The definition may include:</p> <ul style="list-style-type: none"> <li>- ID of the name string for which prerecorded voice data is available</li> <li>- Index of the name string</li> <li>- Speech type, such as           <ul style="list-style-type: none"> <li>NOT_AVAILABLE (phoneme has no type)</li> <li>FIRST_APPROPRIATE (first appropriate standard pronunciation)</li> <li>STANDARD (alternative standard pronunciation)</li> </ul>           Speech types are defined in <code>nds.speech &gt; common</code> </li> <li>- BLOB with recorded voice data</li> </ul> <p>A record with prerecorded voice data refers to one or more names, see Section <i>References from Phonetic Transcriptions and Prerecorded Voice Data</i> on page 284.</p>
Metadata	<p>The following metadata is available for the Speech building block:</p> <ul style="list-style-type: none"> <li>- Building block type (DataScript location: <code>nds.common &gt; BuildingBlockType</code>)</li> <li>- Phonetic transcription format (DataScript location: <code>nds.overall.metadata &gt; PhoneticTrancscriptionMetadata</code> <code>PhoneticTranscriptionFormat</code>)</li> </ul>

## References from Phonetic Transcriptions and Prerecorded Voice Data

The Speech building block stores references from all phonetic transcriptions and prerecorded voice data to the corresponding name strings, as well as the speech data itself. There is an n-to-m relation between phonetic transcription and name strings.

---

<b>Note</b>	This not only applies to the Name building block, but also to TMC and POI names.
-------------	--

---

## Tables of the Speech Building Block

References and speech data are stored in the following tables of the Speech building block:

Tables defined in `nds.speech.phonetictranscription`:

- `NamedObjectPhonemeTable` (stores both references to name data and phonetic transcriptions for named objects)
- `PoiNameToPhoneticTraTable` (stores references from phonetic transcriptions to POI names)
- `PoiCatNameToPhoneticTraTable` (stores references from phonetic transcriptions to POI category names)

- **PoiAttrNameToPhoneticTraTable** (stores references from phonetic transcriptions to POI attribute names)
- **PoiPhoneticTraTable** (stores phonetic transcriptions for POI data)
- **TiPhoneticTraTable** (stores both references to TMC data and phonetic transcriptions for TMC features)
- **PhoneticTranscriptionAuxTable** (stores combinations of prepositions and definite articles), see *Articles and Prepositions* on page 286.

Tables defined in `nds.speech > prerecordedvoice`:

- **NamedObjectToPrerecordedVTable** (stores references from prerecorded voice data to named objects)
- **NamedObjectPrerecordedVTable** (stores prerecorded voice data for named objects)
- **PoiNameToPrerecordedVTable** (stores references from prerecorded voice data to POI names)
- **PoiCatNameToPrerecordedVTable** (stores references from prerecorded voice data to POI category names)
- **PoiAttrNameToPrerecordedVTable** (stores references from prerecorded voice data to POI attribute names)
- **PoiPrerecordedVTable** (stores prerecorded voice data for POI names)
- **TmcEventPrerecordedVTable** (stores references from prerecorded voice data to TMC event names)
- **TmcLocationPrerecordedVTable** (stores references from prerecorded voice data to TMC location names)
- **TpegEventPrerecordedVTable** (stores references from prerecorded voice data to TPEG-TEC event names)
- **TiPrerecordedVTable** (stores prerecorded voice data for TMC features)

## 14.2 Grammatical Structures for Phonetic Transcriptions

NDS provides additional structures for storing grammatical data, such as prepositions, definite articles for names, and grammatical cases. This data enables TTS applications to generate grammatically correct voice guidance instructions, also for languages in which nouns and/or articles are declined after specific prepositions. This applies, for example, to German and Slavonic languages.

### Articles and Prepositions

Combinations of prepositions and definite articles are stored in the `PhoneticTranscriptionAuxTable` (DataScript location `nds.speech > phonetictranscription`). Each combination is identified by an ID (`phoneticTranscriptionAuxId`). This ID may be used to associate a phonetic transcription with a specific combination of preposition and definite article. There is an n-to-m relation between phonetic transcriptions and preposition/article combinations.

NDS distinguishes two use cases for preposition/article combinations:

- “Turn into” use case (`intoUseCasePrefixString`): Prepositions and articles for guidance instructions like “Turn right **into** Washington Street” (English) or “Rechts abbiegen **in den** Sachsenweg” (German)
- “Follow” use case (`followUseCasePrefixString`): Prepositions and articles for guidance instructions like “Follow **the** Washington Street” or “Folgen Sie **dem** Sachsenweg”

### Grammatical Cases

Some languages, for example the Slavonic languages Russian or Croatian, require specific grammatical cases after certain prepositions, meaning that the following nouns including street names must be declined. For this reason, NDS provides the `grammaticalCase` attribute to store the case of a specific phonetic transcription. This enables TTS applications to select the grammatically correct noun form for guidance instructions.

The values for the `grammaticalCase` attributes are defined in `nds.speech > common`.

---

**Example** The grammatically correct translation for “Turn left/right into “Petrinjska ulica” in Croatian is: “Skrenite lijevo/desno **u** Petrinjsku ulicu”, meaning that both a specific preposition and a declined street name must be used.

To enable a TTS application to generate a correct instruction, the declined street name (“Petrinjsku ulicu”) is stored as a separate phonetic transcription; the attribute `grammaticalCase` of this transcription is set to `ACCUSATIVE_CASE`.

The preposition (“u”) is stored in the `PhoneticTranscriptionAuxTable` and assigned to the phonetic transcription of the declined street name via the `phoneticTranscriptionAuxId` as `intoUseCasePrefixString`.

---

## 14.3 Optimization Structures for Phonetic Transcriptions

Due to the variety of speech engines it is not possible to standardize speech recognizer search trees. Therefore, a general container format for speech recognizer search trees shall be specified as a proprietary add-on.

Several search tree containers for different TTS and speech recognizer engines may be included.

For speech recognition based on available hierarchical location input structures (for example, speech recognition for all cities of Germany, all roads of Hildesheim), the references from the respective NVC trees to named objects can be used to retrieve phonetic transcriptions. However, as building up speech recognition search trees based on a large amount of phonetic transcriptions takes a long time, proprietary search trees may be required for large NVC trees or other use cases, for example, free text speech recognition.



## 15 Digital Terrain Model Building Block

Advanced Map Display is defined as advanced content for map display. Currently, the following content is modeled in advanced map display:

- Digital Terrain Model, meaning the digital representation of ground surface topography or terrain
- Orthoimages, meaning satellite and aerial images of the surface (see Chapter 16 *Orthoimages Building Block* on page 305)
- 3D Objects, meaning three-dimensional models of real world objects (see Chapter 17 *3D Objects Building Block* on page 309)

NDS provides separate building blocks for this content. This chapter introduces the Digital Terrain Model building block. The Digital Terrain Model building block is an optional building block.

### Digital Terrain Model

A digital terrain model (DTM; also known as digital elevation model, DEM) is a digital representation of ground surface topography or terrain which can be modeled by a grid of squares (a so-called *raster*), or as a triangular irregular network (TIN).

DTM is used to render 3D views of the world. There are two different kinds of 3D views:

- 3D perspective, also called bird's eye view
- Driver's perspective, also called worm's eye view

The 3D views can be used in combination with features of the basic navigation building blocks, such as basic map display and routing features, as well as with other advanced map display features, such as 3D objects and orthoimages. As the elevation data which is currently available for DTM is too coarse and not stored in a way that is independent of road geometry, DTM data is not used to enhance the geometry for advanced routing or advanced driver assistance systems.

### Representations of Digital Terrain Model in NDS

There are two possible representations of DTM in NDS:

- Height Maps: A grid of squares with height information is stored in the NDS database. The navigation application can render this height information to display a 3D view of the terrain.
- Batched Dynamic Adaptive Meshes (BDAM): The terrain is modeled by means of triangulated irregular networks (TIN) organized in BDAM patches.

The two representations are defined in `nds.common > BuildingBlockDetailedType`. Both representations use the same input data, but differ in the way the data is processed.

The image format for the two representations is defined in the level metadata (DataScript location: `nds.overall.metadata > LevelMetadata > dtmImageFormat`). Table 15-1 shows the data, which can be stored for DTM images.

*Table 15-1 DTM image data*

Metadata	Description
<code>dtmImageFormat</code>	Defines the image format The following values are possible: <ul style="list-style-type: none"> <li>- PNG</li> <li>- JPEG</li> <li>- JPEG_2000</li> <li>- SIMPLE_ARRAY: Defines that an array of values is stored instead of an image</li> </ul>
<code>heightResolution</code>	Defines the height resolution for images

## 15.1 Height Maps

A height map is a grid of squares which is used to store height values for surface elevation data. The height values are derived from a height map picture of the respective terrain in gray scales which is available in the input data. The gray scale values are converted to height values.

The application can render the height information stored in the grid to display a 3D view of the terrain.

In NDS, the height map is represented by one grid per tile. The grid is built by a defined number of height points, starting at the south-west corner of the tile and distributed in equidistant order on the tile. The number of grid points per tile is determined at compile time.

For more information, refer to *NDS – Compiler Interoperability Specification, 12.1 Height Map Tile Dimensions* on page 195.

### 15.1.1 Structure and Content of Height Maps

The height map information is stored per tile in the `DtmHeightMapTileTable` (DataScript location: `nds.dtm.heightmap`). Table 15-2 shows the data contained in the `DtmHeightMapTileTable`.

*Table 15-2 Height map data*

Data	Description
<code>id</code>	Packed tile ID of the height map tile with height information
<code>versionId</code>	Identifies the version of the height map tile for updates For more information, refer to <i>NDS – Update Specification, 5.2 Subversion and Version Control for NDS</i> on page 40.

Data	Description
cipherKeyInfoId	Reference to the cipher key info, which can be used to decrypt the BLOB field For more information, refer to Section 8.3 <i>Cipher Key Info Structure</i> on page 104.
heightOffset	Encodes the reference height offset (from -32768m up to 32767m)
ndsData	Contains the binary height map data in form of an image or simple array The height values of this image contain the heights relative to the height offset stored for this tile. The absolute height value is calculated by adding the height offset to the height value derived from the image.

### 15.1.2 Partitioning of Height Information into Tiles and Levels

The digital terrain model is stored in correspondence with the NDS tiling scheme. For each tile level, a height resolution needs to be defined. The sampling rate defines how many height values are stored, that is, every x meter one height value in latitude and longitude.

The NDS data supplier assigns the height resolution values to levels. The height resolution defines the unit of the height encoding and thus defines the smallest possible difference between two height values. Table 15-3 gives an example.

Table 15-3 Examples for height resolutions in NDS tiling scheme

NDS level	Sampling rate [m]	Example height resolution [m]	Tile width [km]
0	81920	500	19660,8
1	40960	500	9830,4
2	20480	200	4915,2
3	10240	200	2457,6
4	5120	200	1228,8
5	2560	100	614,4
6	1280	100	307,2
7	640	10	153,6
8	320	10	76,8
9	160	10	38,4
10	80	1	19,2
11	40	1	9,6
12	20	1	4,8
13	10	0,5	2,4
14	5	0,5	1,2
15	2,5	0,5	0,6

The following guidelines apply for the definition of the height resolution:

- The height data should not be stored in a higher resolution than provided in the input data. If the input data provides a data resolution of 30 m, for example, level 12 would be the lowest DTM level.  
Due to the small size of DTM tiles and a low number of higher level tiles, it is easily possible to load tiles of a higher level for display, assuming the same display window in a lower level.
- It is not necessary to store a resolution for each level. If a DTM tile of a higher level is, for example, identical to the DTM tile of the next lower level, the lower level is not needed. This is, for example, the case if a tile contains water or other plain structures of the same height.

### 15.1.3 Content of a Single DTM Tile

Each DTM tile contains a height map that covers the complete area of the tile.

The physical encoding of the data is equal to the encoding of raster images. The number of pixels is determined by the available input data resolution and the resolution defined for the level.

With respect to the WGS 84 coordinate system, NDS tiles are quadratic. But taking the real extent of the covered ground into account, tiles are nearly rectangular trapezoids with the width decreasing towards the poles (height stays fixed). The ratio of the raster images will, therefore, change from the equator to the poles from quadratic to rectangular forms. Hence, the number of pixels equals the next higher power of 2 depending on the actual ratio, for example, 128 x 256 pixels for ratio 1:2, and 32 x 256 pixels for ratio 1:8.

The height values are stored per vertex.

---

<b>Note</b>	The uncompressed size of a height map for a certain tile level can be derived from the tile width (in meter) and the height resolution: <ul style="list-style-type: none"><li>■ Number of pixels = (tile width / height resolution)<sup>2</sup></li><li>■ Size of image in bytes = Number of pixels x number of bytes per height value</li></ul> For fast handling of height map data by the application, the number of pixels is rounded to a power of 2. This avoids waste of memory when using height maps with shaders.
<b>Example</b>	On level 13, the number of pixels for a tile with a width of 2400 m and a sampling rate of 10 m, the number of pixels is 240 x 240. This is rounded to 256 x 256 pixels = 65536 pixels, which leads to a tile size of 128 KB.

---

In order to enhance compression, the lowest height value offset per tile is defined as 0 and all other values are relative to this height. For example, consider a unit of 0.5 meter per height value for level 13 and a level 13 tile for which all heights lie between 200m and 250m. Then the offset is encoded by value 400 and the heights lie between the values 0 and 100. Negative values may be used for the offset; this is necessary to describe, for example, the area around the Dead Sea.

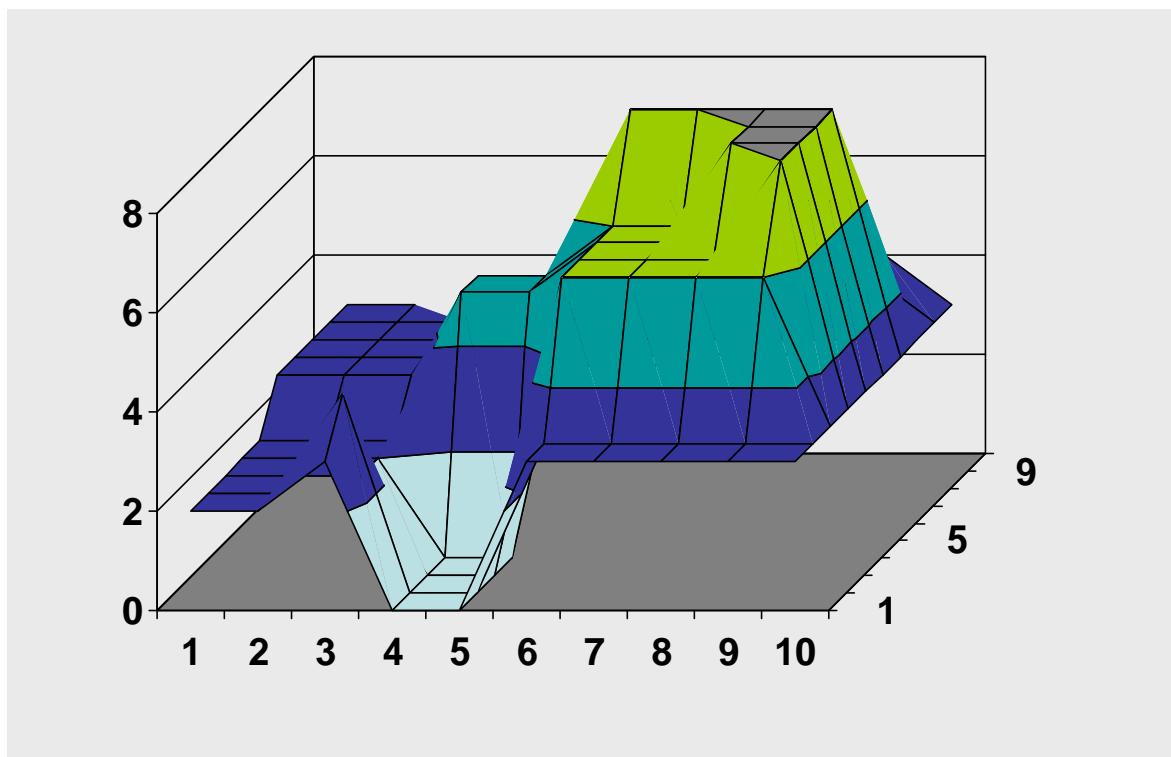
For the sampling of height values, an interpolation is needed to determine the higher resolution of coordinate units based on the lower resolution of height values.

**Example** The application can perform this interpolation, if the ratio of coordinate unit to pixel is small, for example, 1:10 (see Table 15-3).

To fit the input data to NDS tiles it must be interpolated. An offset to an anchor point is not stored due to the different resolutions of the input data and the NDS tiling scheme.

Figure 15-1 shows a height map in form of heights stored in a 3D coordinate system, visualized in an Excel graphic.

*Figure 15-1 Height map as 3D diagram in Excel, example*



### 15.1.4 Compression for Height Maps

Height maps store absolute height values in a regular grid. For terrains with small height variance, such as plateaus, height data, therefore, is highly redundant, and an efficient compression is recommended.

The following compression types are possible:

- Image format compression: Compression of height maps can be conducted by using picture formats that support compression, such as PNG, JPEG, and JPEG2000. All of these formats, however, have drawbacks:
  - The combination of JPEG with a high compression rate (for example 40%) creates artifacts in the digital terrain model, such as stairs. JPEG2000 supports a more sophisticated compression by using a wavelet method which can be used for fast compression.
  - The PNG format supports different delta compression methods and the same deflate compression algorithm as zlib. The decompression of PNG images, however, is slower than the decompression of JPEG images.
- Delta compression (`nds.dtm.heightmap > SimpleArrayHeightMapData`): Delta compression indicates not to store plain raster image formats, but to apply special encoding. Delta values can be applied for each line. The height value of the first point in the next line is relative to the height value of the first point of the previous line.

The costs for transformation from a delta compressed format to an uncompressed format are high, especially when just sampling parts of the height map. In order to have a more efficient transmission and a small memory footprint for cached data, one offset value per tile can be stored and the heights then be encoded relative to this offset.

In order to make partial sampling of height maps possible, NDS recommends to combine the image compression approach with storing an offset for each tile encoding the minimum absolute height of the tile. Delta compression can be applied additionally for PNG, as the influence of delta compression on the performance of partial sampling of PNGs is not yet clarified.

## 15.2 Batched Dynamic Adaptive Meshes (BDAM)

Batched Dynamic Adaptive Meshes (BDAM) are used for out-of-core rendering of fully scalable data for large digital terrain databases.

The geometry data is stored in so-called *BDAM patches* organized in a pair of binary trees. A BDAM patch is a polygon primitive modeled by means of a compound of right-angled, isosceles triangles. Within the BDAM patch, the triangles are organized in TINs (Triangulated Irregular Network), and the polygons are organized in triangle strips. BDAM patches can be combined in different sizes and, thus, combine different levels of detail (LOD) in a single map display view.

The texture data for the terrain is stored in a tiled quadtree. If a more detailed texture is needed, the algorithm proceeds to the next lower level to obtain four textures with a more detailed resolution for the texture. As geometry data is stored in triangles, one step in the texture quadtree means two steps in the binary tree for geometry data. Texture data is stored sequentially.

This approach makes it possible to perform a hierarchical view frustum culling and a view-dependent texture and geometry refinement at each frame by means of a stateless traversal algorithm (see Section *Level of Detail* on page 297). To avoid holes in the terrain model, error levels are assigned to BDAM patches and stored with the geometry data. For geometry refinement, the following rule applies: The error level of the two short sides of the BDAM patch must be equal to the error level of the long side of the next detailed level.

For general information on BDAM, refer to <http://www.crs4.it/vic/cgi-bin/bib-page.cgi?id='Cignoni:2003:BBD'>.

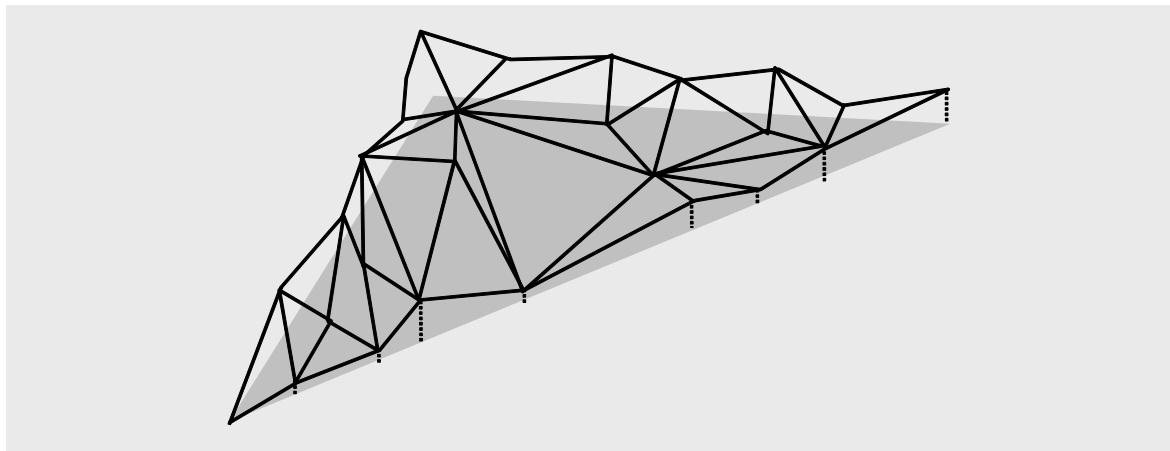
### TINs in BDAM Patches

Each BDAM patch contains a TIN mesh that covers the entire surface of the BDAM patch. A TIN is a vertex-based representation of the physical land surface. It is made up of lines and irregularly distributed nodes with three-dimensional coordinates (x, y, z), that are arranged in a network of non-overlapping triangles.

For more information on TINs, refer to [http://en.wikipedia.org/wiki/Triangulated\\_irregular\\_network](http://en.wikipedia.org/wiki/Triangulated_irregular_network).

Figure 15-2 illustrates the TIN mesh in a BDAM patch.

Figure 15-2 BDAM patch surface covered by a TIN mesh



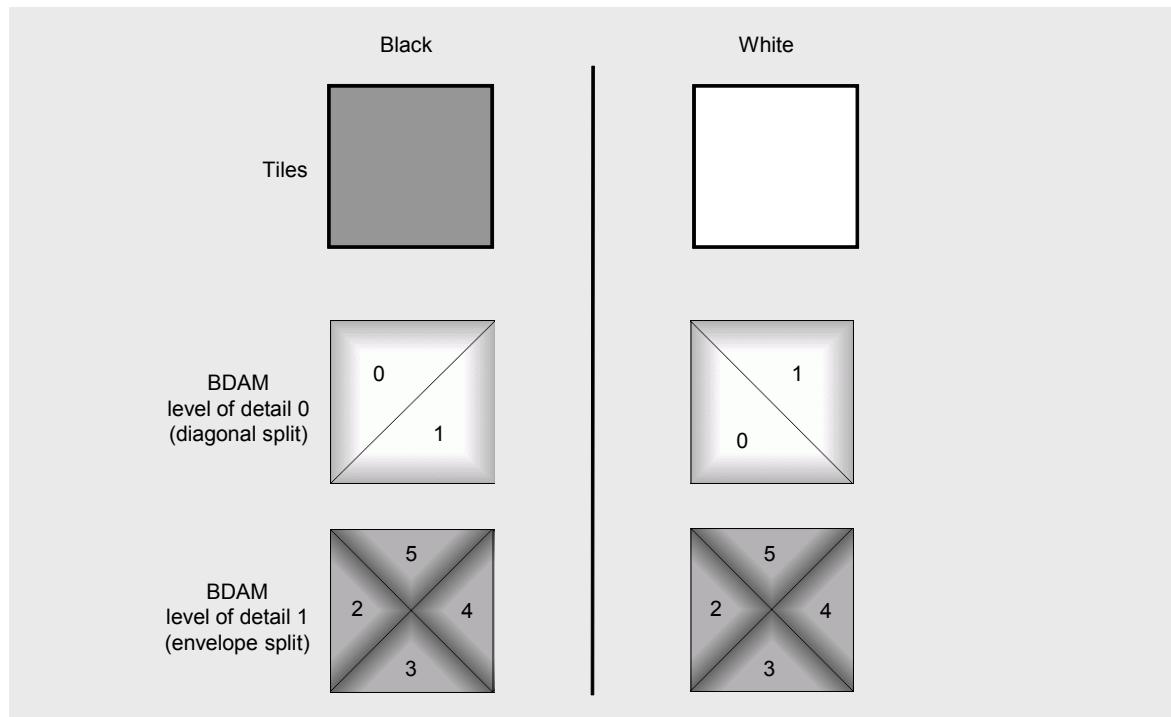
### 15.2.1 Assigning BDAM Patches to NDS Tiles

BDAM patches are assigned to NDS tiles and each BDAM patch can, therefore, be addressed via its corresponding NDS tile. An NDS tile is split into two BDAM patches. The split line is a slash or a backslash in alternating order following this rule: In all NDS levels, the south-west tile is regarded as a black tile. The horizontal and vertical neighbor tiles of a black tile are regarded as white tiles. Thus, a pattern similar to a chess board is created. In this pattern, the black tiles use the slash, the white tiles use the backslash for the diagonal split.

In the next BDAM level of detail (see *Level of Detail* on page 297), the NDS tile is split into four parts by two diagonal cutting edges. This is called envelope split or X split.

Figure 15-3 illustrates the splitting rules.

*Figure 15-3 Splitting rules for BDAM patches*



The BDAM patches of an NDS tile are addressed via index numbers. The indexing is different for black and white NDS tiles. Table 15-4 illustrates the indexing of BDAM patches for both BDAM levels of detail. The reference point for all coordinates stored in BDAM patches is the center point of the NDS tile assigned to the respective BDAM patch.

*Table 15-4 Indexing for black and white tiles*

NDS Tile	BDAM Patch Index
Black tile	0: North-west 1: South-east
White tile	0: South-east 1: North-west

NDS Tile	BDAM Patch Index
Black and white tile	2: West 3: South 4: East 5: North

### 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. This can be applied, for example, to objects or areas far away from the camera position. A real-world error would then result in a small screen-space error which would not be noticed on the display. The level of detail concept is used in computer graphics to speed up real-time rendering without a visible loss of detail on the screen.

For more information on level of detail, refer to [http://en.wikipedia.org/wiki/Level\\_of\\_detail](http://en.wikipedia.org/wiki/Level_of_detail).

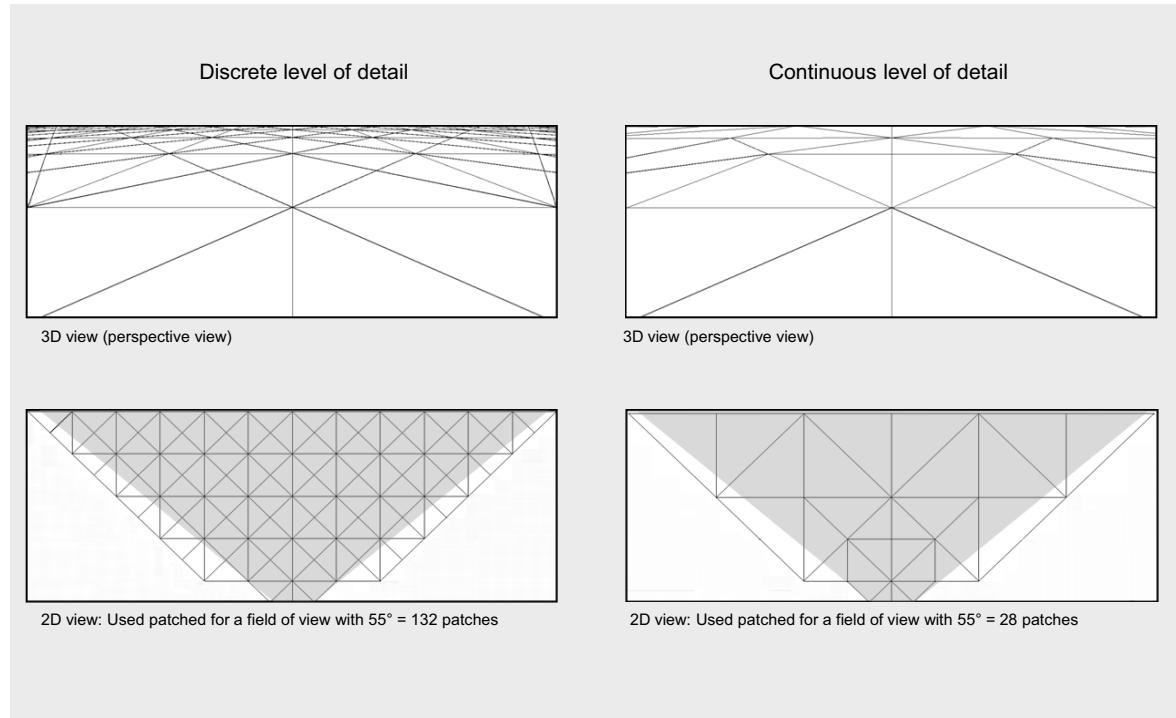
For large objects, such as the surface of the earth, being partly close to the camera position and partly far-off the camera position, choosing one LOD for the entire object is not sufficient as this would lead to one of the following effects:

- A high workload for the graphics engine by using a high LOD for the entire object
- Insufficient LOD for the parts close to the camera

To display large objects correctly and to support high-performance rendering, it is necessary to have multiple LODs at the same time. It is important that the different LODs match on their respective boundaries and do not have holes or other graphic artifacts. This method is called *continuous LOD*.

Figure 15-4 shows the advantage of continuous LODs instead of discrete LODs, where the entire object is displayed with the same level of detail.

Figure 15-4 Continuous and discrete level of detail (BDAM)



### Assigning BDAM Levels to NDS Levels

The complete area of the globe is covered by BDAM patches of equal size, building a repetitive pattern without holes or overlappings.

For each NDS level, two BDAM levels of detail exist (in the following referred to as *BDAM levels*). The BDAM patch of the next detailed BDAM level is half the size of the BDAM patch of the higher BDAM level. In the NDS tiling scheme, the tiles on the next detailed level are quartered. Two subsequent changes of BDAM levels, therefore, correspond to one change of NDS levels .

To each level N, the BDAM levels N.0 and N.1 are assigned. BDAM level N.0 is the less detailed level.

Figure 15-5 shows the NDS tiling scheme for levels 0 to 3 and the corresponding BDAM levels.

Figure 15-5 NDS levels and BDAM levels of detail

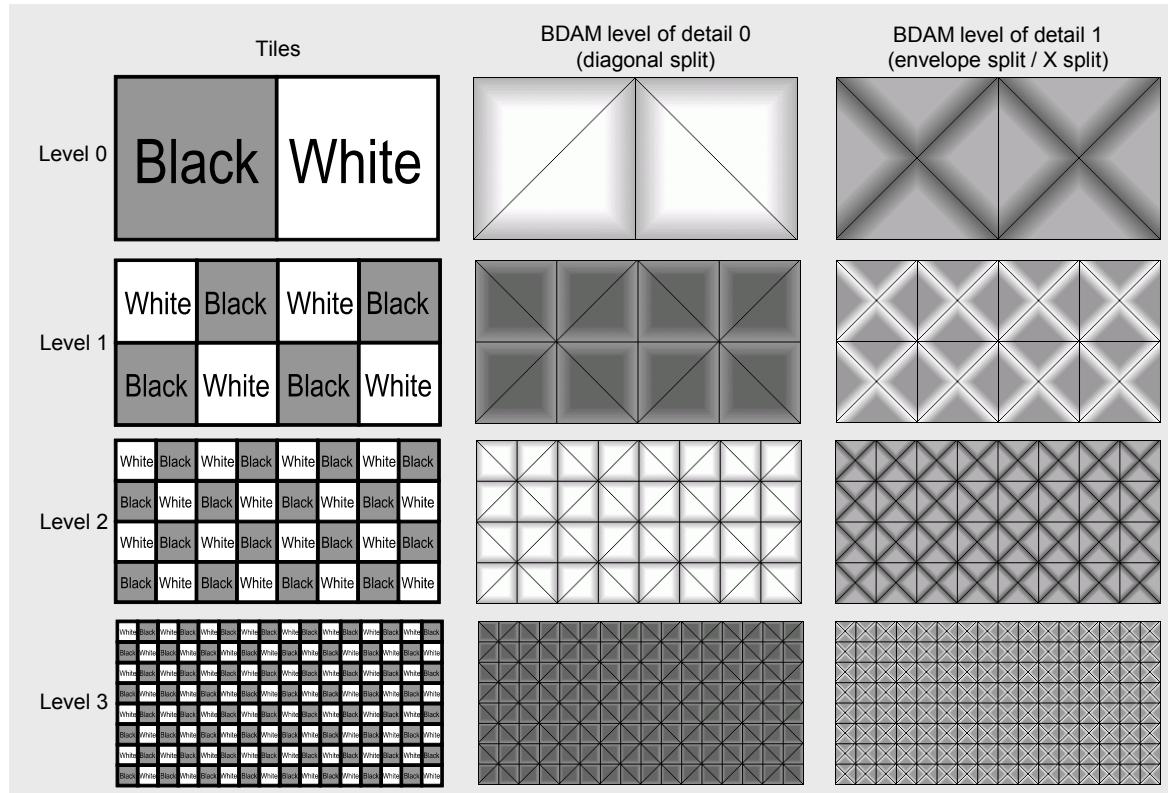


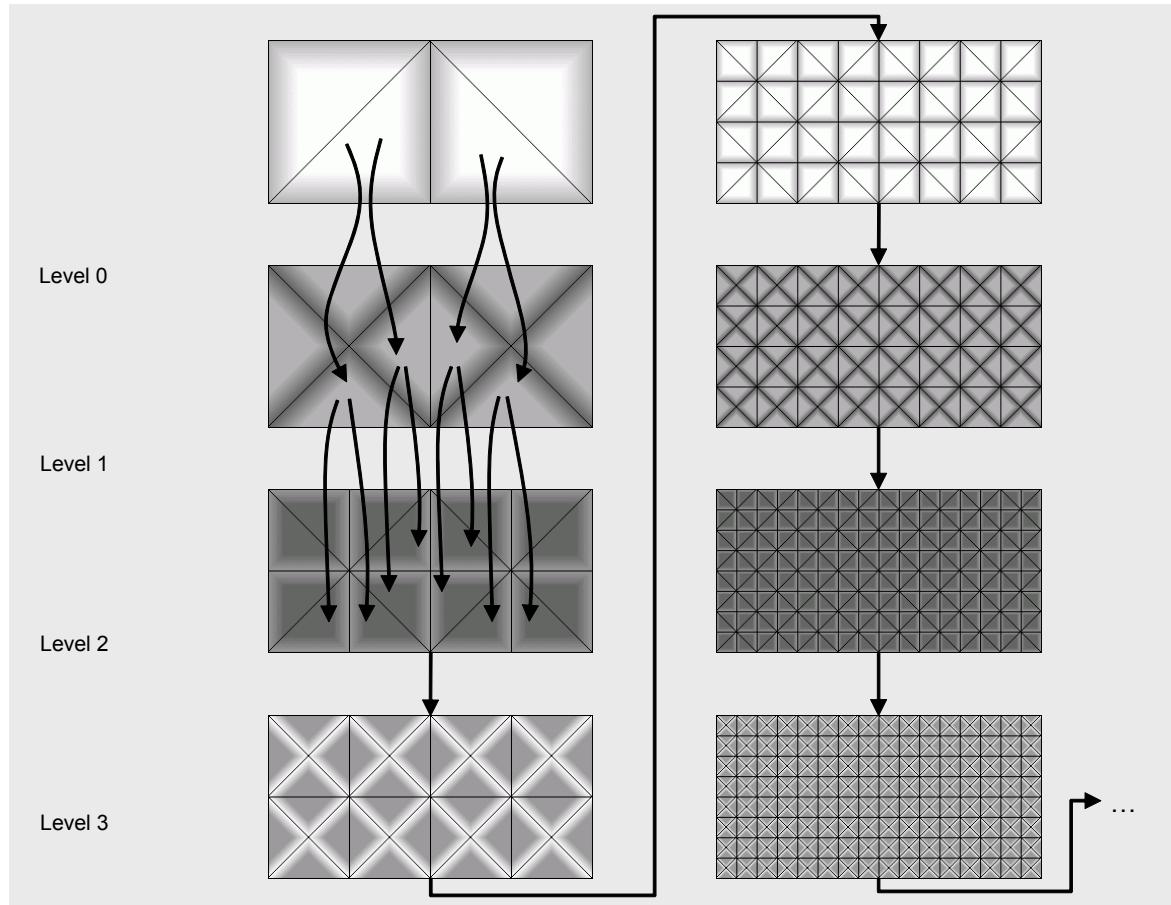
Table 15-5 shows the allocation of NDS levels and tiles to BDAM levels and BDAM patches.

Table 15-5 NDS levels and BDAM levels of detail

NDS level	Number of tiles	BDAM level	Number of BDAM patches
0	2	0.0	4
		0.1	8
1	8	1.0	16
		1.1	32
2	32	2.0	64
		2.1	128
3	128	3.0	256
		3.1	512
4	512	4.0	1024
		4.1	2048

Figure 15-6 illustrates the recursive splitting of patches in different BDAM levels per NDS level.

Figure 15-6 Recursive splitting of patches in different BDAM levels



### Rules for Mixing BDAM Levels

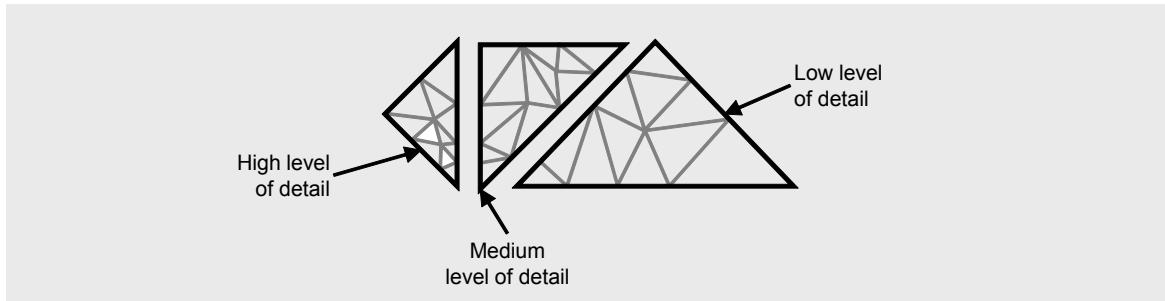
As outlined in Section *Level of Detail* on page 297, it is required to mix different levels of detail to display large objects correctly, and at the same time support high performance rendering. To avoid holes or other graphic artifacts, the different BDAM levels must match at their conjoined borders, and the common side length of two BDAM patches must be equal.

A BDAM patch is a right-angled, isosceles triangle. In right-angled, isosceles triangles the ratio of the long side (hypotenuse) over either of its short sides (catheti) equals the square root of two. Hence, the following rules for mixing BDAM patches apply:

- The long side of a BDAM patch in BDAM level N matches
  - The long side of another BDAM patch in BDAM level N
  - One of the short sides of a BDAM patch in BDAM level N-1
- The short side of a BDAM patch in BDAM level N matches
  - One of the short sides of another BDAM patch in BDAM level N
  - The long side of a BDAM patch in BDAM level N+1

This is illustrated in Figure 15-7.

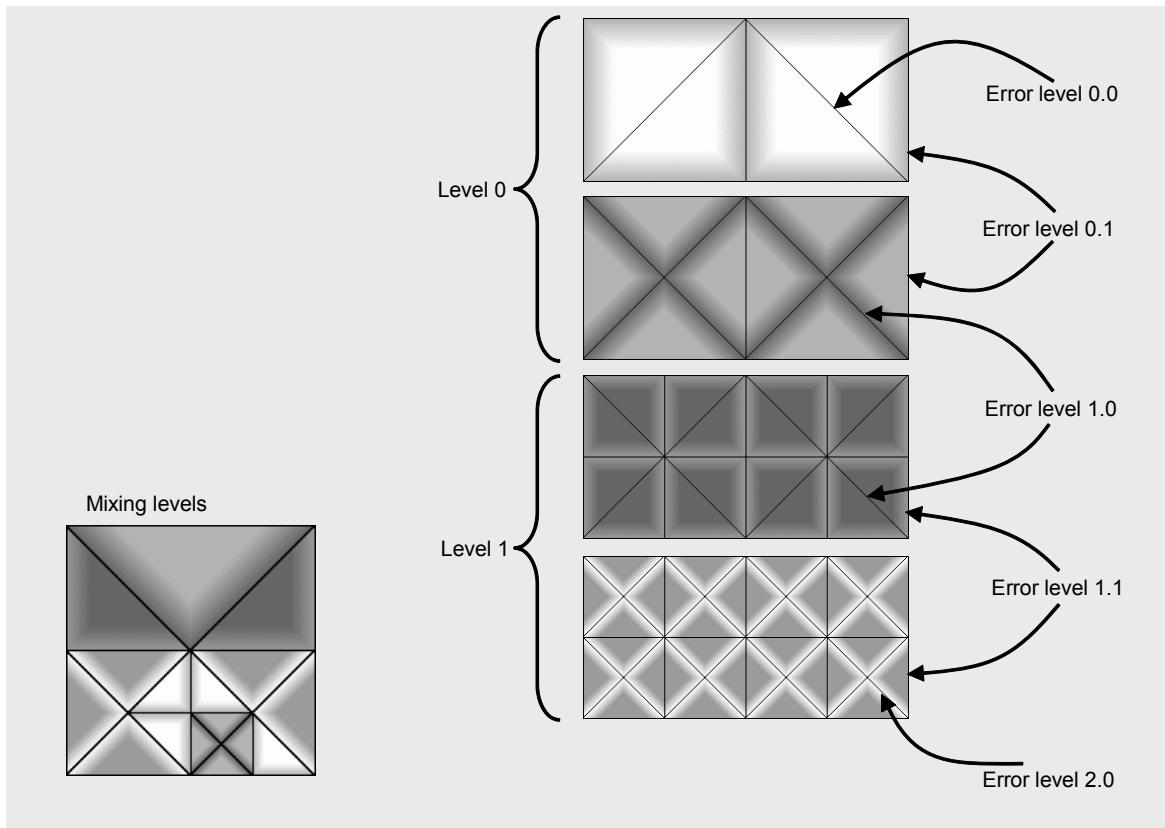
*Figure 15-7 Mixing BDAM levels of detail*



The rule for mixing BDAM levels is expressed in terms of an error level. Error levels are set for the compilation of BDAM patches. One error level is set for the long side of the BDAM patch and one error level is set for the two short sides. Matching sides have the same error level.

Figure 15-8 depicts the error levels assigned to the long and short sides of the BDAM patches on different NDS levels. It also shows a tile with mixed BDAM levels.

*Figure 15-8 Mixing BDAM level of details, error levels*



## 15.2.2 Building Block Structure and Content

The Digital Terrain Model building block uses the NDS tiling scheme and may have data on all NDS levels from level 0 to level 15. Which level is used depends on data density: For areas with high data density, such as large cities or mountain areas, level 15 is needed. For areas with low data density, such as oceans, deserts, or non-digitized areas, level 4 or 5 are sufficient. This reduces the required database memory for less important areas, and at the same time provides a high level of detail for areas where it is necessary.

### BDAM Metadata

The DTM level metadata define the metadata for each BDAM level (see Table 15-6).

DataScript location: `nds.overall.metadata > DtmLevelMetadata`

*Table 15-6 Metadata for BDAM*

Metadata	Description
<code>coordWidth</code>	Lat and long values of a tile
<code>altitudeDeltaShift</code>	Defines the position of the decimal point.
<code>spaceErrors</code>	Space error array For each DTM level, two sublevels are stored.

### Modelling BDAM Patches in NDS

For each NDS tile, a corresponding `DtmSurfaceTile` (DataScript location: `nds.dtm.surface`) exists. The `DtmSurfaceTile` refers to the six BDAM patches distributed over two BDAM levels (see Figure 15-3). The BDAM patches are represented by `DtmSurfaceTin`.

### Rendering BDAM Patches

For rendering BDAM patches, a single array is used to index the arrays for vertices and normals in order to display the height of a BDAM patch without texture information. The index array represents only triangle strips, and not triangle fans or single triangles. The application can decide to take an orthoimage as a texture or build a texture based on data from the Basic Map Display building block.

- 
- |             |  |
|-------------|--|
| <b>Note</b> | The complete area of a BDAM patch has to be covered without overlapping.<br><br>To join two separate triangle strips into one triangle strip, a degenerate triangle must be generated to bridge the gap between the two triangle strips. |
|-------------|--|
- 

A vertex may be stored multiple times in the vertex array and may have a different normal assigned to each of its instances. This facilitates the display of sharp edges or mountain peaks. The peak vertex of the Cheops pyramid, for example, is stored four times in the vertex array with the same coordinates and references to four different normals that are directed to the four cardinal directions.

## BDAM Pattern Tile

In the same way as in Basic Map Display, tile patterns can be defined in BDAM for map elements that extend over the whole area of several tiles. The provision of such patterns can significantly save storage space because identical tiles need to be stored only once. This is useful for larger areas with identical map content, such as large lakes, oceans, forests, or deserts.

Patterns are assigned to tiles via reference tables (DataScript location: `nds.dtm.bdam > DtmBdamPatternTileTable`).



## 16 Orthoimages Building Block

Advanced Map Display is defined as advanced content for map display. Currently, the following content is modeled in advanced map display:

- Orthoimages, meaning satellite and aerial images of the surface
- Digital Terrain Model, meaning the digital representation of ground surface topography or terrain (see Chapter 15 *Digital Terrain Model Building Block* on page 289)
- 3D Objects, meaning three-dimensional models of real world objects (see Chapter 17 *3D Objects Building Block* on page 309)

NDS provides separate building blocks for this content. This chapter introduces the Orthoimage building block. The Orthoimage building block is an optional building block.

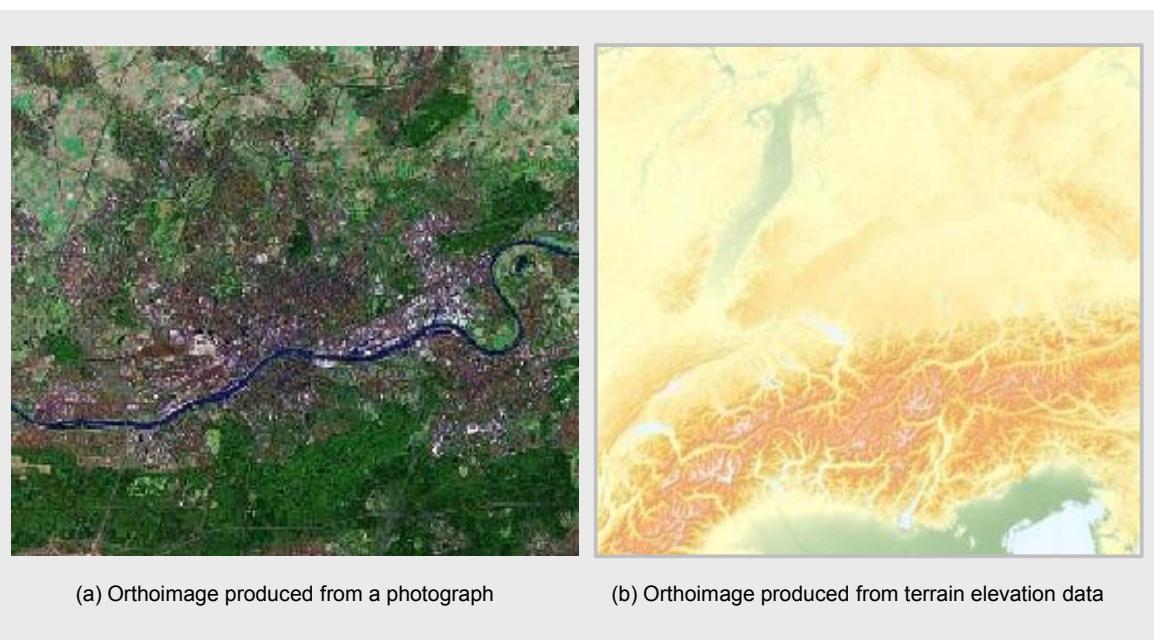
### Introduction to Orthoimages

Orthoimages are raster images usually produced from aerial or satellite photographs. In order to minimize image distortions caused by topographical variations and camera tilt, the photographs are geo-referenced and ortho-rectified. The ortho-rectification also reduces cracks and mismatches when combining adjacent or overlapping images from different sources.

Orthoimages can also be created from other sources; terrain elevation data, for example, can be used to produce images which depict elevations with colors.

Figure 16-1 shows examples of orthoimages.

*Figure 16-1 Orthoimages, examples*

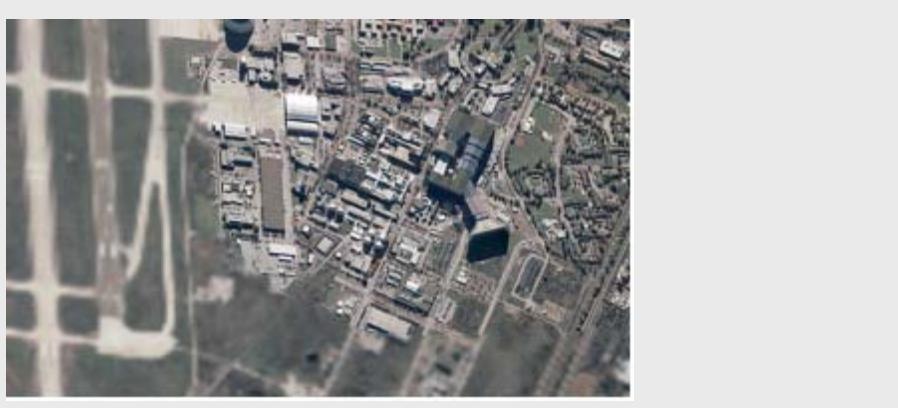


## 16.1 Tiling Scheme for Orthoimages

Orthoimages use the NDS tiling scheme specified in Chapter 7 *Partitioning of Geographic Data* on page 89. They are geo-referenced to WGS 84 and are resampled to NDS tiles of an appropriate level, depending on coverage and pixel resolution. The resampling process also resolves distortions that can arise due to the fact that the west-to-east size of a tile decreases with increasing latitude. That means that as one pixel of an orthoimage corresponds to a distance in degrees in the WGS 84 geographic coordinate system and not in meters, a tile corresponds consequently to a trapezoid, rather than a rectangle on the earth's surface. This effect increases from equator towards the poles.

Each orthoimage tile must cover the entire tile area. If the source data only covers parts of a given NDS tile, the missing parts must be interpolated from lower resolution data during the compilation process. NDS tiles for which no source data is available may be omitted. In this case, the application decides whether to use orthoimage tiles from other levels or to use a default background image.

Figure 16-2 Background bitmap with varying ground resolution



## 16.2 Building Block Structure and Content

Orthoimages are part of Advanced Map Display data and are stored in a separate building block. This facilitates database updates. Each image is stored as a BLOB in the orthoimage tile table (DataScript location: `nds.orthoimages.main > OrthoImageTileTable`).

Multi-resolution display of orthoimages is supported. This enables the application to display the images of different levels (and of different resolutions) together, that is, in a perspective projection of a 3D scene with high resolution images in the front and low resolution images in the background. It is also possible to store several sets of orthoimages for the same geographic area. For the Alps, for example, summer and winter satellite images can be stored.

### Storage Format

Orthoimages are stored in one of the following file formats:

- JPEG
- JPEG\_2000
- PNG
- DirectDraw Surfaces (DDS)

DataScript location: `nds.orthoimages.main > imageFormat`

## Orthoimage Metadata

Orthoimages use the following global metadata:

- Orthoimage type: Image or elevation colored (DataScript location: `nds.common > OrthoImageType`)
- Weather condition depicted on the image, for example, snow, sunshine, rain (DataScript location: `nds.common > weather`)
- OrthoImageMetadata: Weather condition depicted on the image, for example, snow, sunshine, rain. Also defines the pixel resolution in north-south direction (vertical resolution), which applies to all orthoimages in an instance of the building block (see *NDS – Compiler Interoperability Specification, 13.2 Resolution* on page 199; DataScript location: `nds.overall.metadata > OrthoImageMetadata`)
- OrthoImageMetadataTable: Level number and orthoimage source type, for example, satellite or aerial (DataScript location: `nds.orthoimages.main > OrthoImageMetadataTable`)
- Version information as defined in the version table (DataScript location: `nds.overall.productdbversion > VersionTable`)
- Cipher information as defined in the cipher key information table (DataScript location: `nds.overall.drm > CipherKeyInfoTable`)



## 17 3D Objects Building Block

Advanced Map Display is defined as advanced content for map display. Currently, the following content is modeled for advanced map display:

- 3D Objects, meaning three-dimensional models of real world objects
- Digital Terrain Model, meaning the digital representation of ground surface topography or terrain (see Chapter 15 *Digital Terrain Model Building Block* on page 289)
- Orthoimages, meaning satellite and aerial images of the surface (see Chapter 16 *Orthoimages Building Block* on page 305)

NDS provides separate building blocks for this content. This chapter introduces the 3D Objects building block, which is an optional building block.

For achieving a good performance in displaying 3D objects in a navigation system, a balance between the following two contradicting requirements has to be achieved:

- High frame rates, for example, 30 frames per second
- Detailed frame display

NDS supports these requirements by storing 3D object data in nodes of a spatial tree structure, thus enabling the navigation system to display detailed graphics at an acceptable frame rate. 3D objects are, therefore, not stored in correspondence with the NDS tiling scheme.

### 17.1 Building Block Structure and Content

The 3D Objects building block consists of one spatial index structure per update region. The nodes of the spatial index structure contain the 3D object data, and have unidirectional references to geometry, material, and texture information, as well as template buildings.

Figure 17-1 depicts the structure of the 3D Objects building block.

Figure 17-1 Structure of the 3D Objects building block

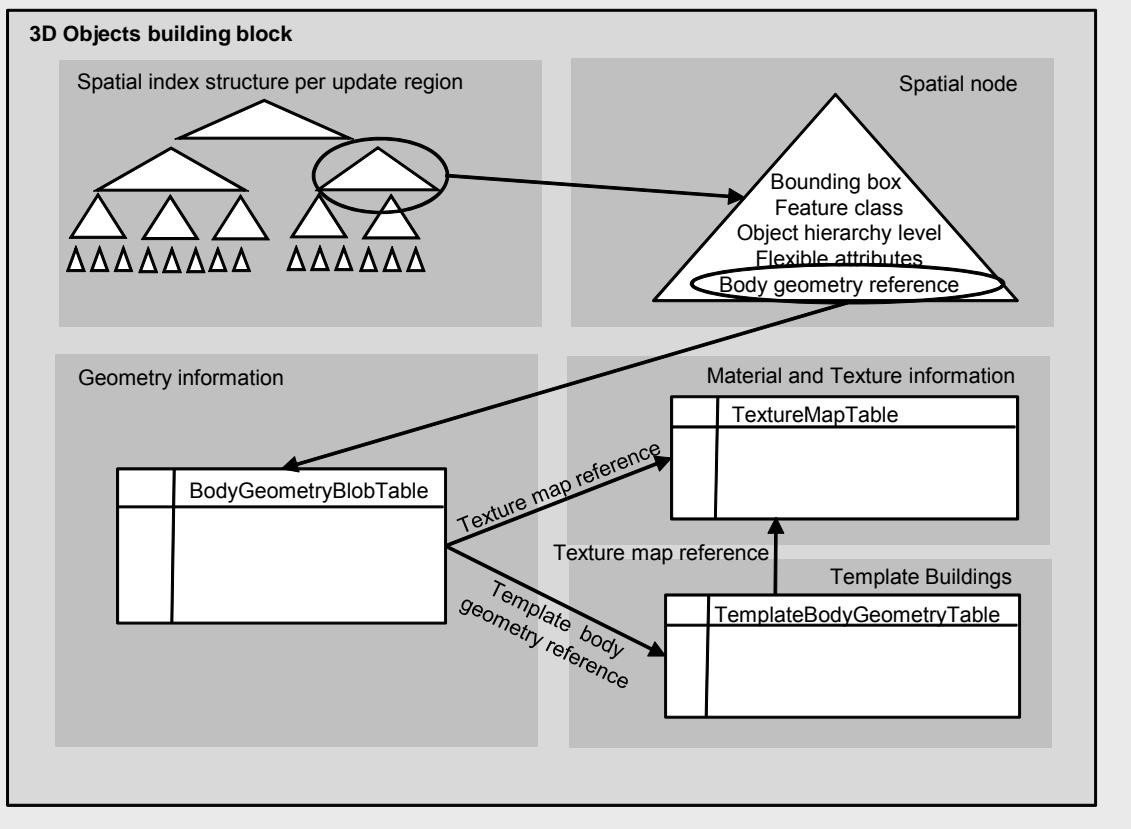


Table 17-1 gives an overview of the data stored in the 3D Objects building block.

Table 17-1 Data in 3D Objects building block

Type of data	Description
Spatial index structure	Data structure following the concept of R-trees; used for spatial access to 3D object features For more information, refer to Section <i>Spatial Index Structure</i> on page 311.
Spatial node	Node in the spatial index structure; used to store 3D object data For more information, refer to Section <i>17.2 3D Object Feature</i> on page 312.
Geometry information	Body geometry information for 3D objects, stored in BLOBS; the nodes of the spatial tree structure reference the geometry information For more information, refer to Section <i>17.3 Geometry Information for 3D Objects</i> on page 322.

Type of data	Description
Material and texture information	Describes the surface of 3D objects by means of references to texture information, material content and color values, which determine how the surface interacts with light  For more information, refer to Section 17.4 <i>Material and Textures</i> on page 331.
Template buildings	Template body geometry information, which can be used to reuse geometry information for recurring building types  For more information, refer to Section 17.3.4 <i>Template Body Geometry</i> on page 327.

Table 17-2 Metadata of the 3D Objects building block

Metadata	Description
Availability of 3D objects attribute types (threeDAttributeTypeAvailability)	Defines the 3D spatial node attribute types that are available for an update region
BMD feature classes (bmdFeatureClasses)	Lists the available 3D feature classes for an update region

## Spatial Index Structure

The spatial index structure is a tree data structure following the concept of R-trees, which allows indexing multi-dimensional information, for example, the coordinates of geographical data. In NDS, the spatial index structure is used for spatial access to 3D object features, which are stored in nodes of the spatial tree. NDS extends the R-tree by two properties:

- In R-trees, only leaf nodes contain or reference geometry data. In the spatial index structure used for NDS, also inner nodes contain or reference geometry data.  
These additional references are required, because NDS provides different object abstraction levels in a spatial tree (see Section 17.2.3 *Object Hierarchy Level* on page 319). The relation between the abstraction levels of one object is given implicitly by the tree hierarchy. Thus, additional access times to eliminate all abstractions but one of the same object can be avoided. Also, the subtree traversal can stop, if the geometry of an inner node is not relevant (see Section 17.2.2 *Bounding Box* on page 315).
- R-trees store no information about the different levels of detail. The NDS spatial index structure, however, contains information about different levels of detail (see 17.3.2 *Level of Detail for 3D Objects* on page 325).

This information can be used as an additional search criterion, and also enable the subtree traversal to stop at an early stage, for example, at a coarse level of detail.

There is one spatial index structure per update region, splitting the geographic area of the update region into hierarchically nested, and possibly overlapping bounding boxes assigned to each node of the spatial index structure.

Spatial trees can be partitioned, that is, they are split into a root subtree and one or more child subtrees. Partitioning of spatial trees enables the application to load only a small area which improves performance and reduces memory consumption. For more information, refer to *NDS – Compiler Interoperability Specification*, 14.1 *Partitioning of Spatial Trees* on page 203.

## 17.2 3D Object Feature

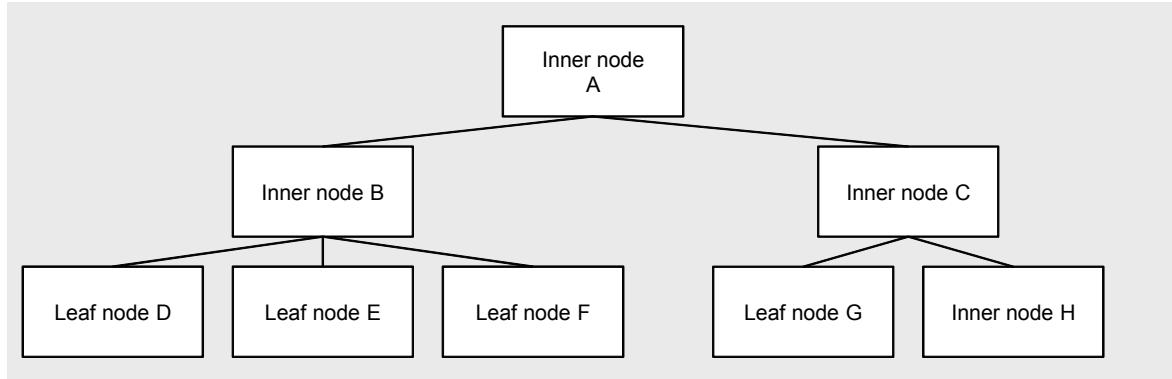
A 3D object is a representation of a real world object, for example, a building as a three-dimensional geometry model. 3D objects can be displayed together with features of the basic navigation building blocks, such as basic map display features, as well as with other advanced map display features, such as the digital terrain model.

The information required to identify and locate 3D Object features is stored in the spatial nodes of the spatial index structure.

The following types of spatial nodes are defined for NDS:

- Inner nodes (*SpatialTreeInnerNode*) are nodes with at least two and at most 65 child nodes, and are specified by a 6-bit value. Inner nodes may represent a 3D object. In this case, the inner node has a feature class and object hierarchy level, and has references to body geometry.
- Leaf nodes (*SpatialTreeLeafNode*) have no children and always represent a 3D object. Thus, a feature class, an object hierarchy level, and references to body geometry are mandatory for leaf nodes.
- Reference nodes (*SpatialTreeRefNode*) are identical to inner nodes, but have no leaf nodes of their own. The leaf nodes of reference nodes are located in a different BLOB, and, thus, the reference nodes need additional information about their assigned leaf nodes.  
Reference nodes are required for partitioning the spatial tree. For more information, refer to *NDS – Compiler Interoperability Specification*, 14.1 *Partitioning of Spatial Trees* on page 203.

Figure 17-2 depicts a spatial tree with inner nodes and leaf nodes.

*Figure 17-2 Spatial tree with inner nodes and leaf nodes*

Each subtree represents a 3D object. The object hierarchy level information stored in the spatial nodes of the subtree gives information about where in the subtree the 3D object is located. The topmost inner node of the spatial structure, for example, represents all 3D objects of an update region.

### 17.2.1 Properties of 3D Object Features

Table 17-3 shows the properties of the nodes in the spatial index structure that define the 3D object features, and their availability for the different node types (inner, reference, and leaf nodes).

DataScript location: `nds.objects3d.spatialnode > SpatialTreeNode`

*Table 17-3 Properties of 3D objects stored in spatial nodes*

Properties	Description	Inner	Reference	Leaf
<code>type</code>	Defines the node type (inner, leaf, or reference node)	X	X	X
<code>numOfChildrenStartWith2</code>	Defines the number of children of the node; the minimum number of children is 2  Example: <code>numOfChildrenStartWith2 = 0</code> defines a node with two children <code>numOfChildrenStartWith2 = 1</code> defines a node with three children	X	X	

Properties	Description	Inner	Reference	Leaf
childSubTree	<p>Stores the ID of the child subtree referenced by the reference node; the subtree structure itself only contains the ID and an array of spatial nodes.</p> <p>DataScript location of the subtree table: nds.objects3d.main &gt; SpatialSubTreeTable</p>	—	X	—
boundingBox	Defines the spatial extension of an area with 3D objects per spatial node	X	X	X
objectHierarchyLevel	Fixed attribute for spatial nodes; the object hierarchy level and the bounding box are used for referencing 3D objects	X	X	X
feature3DClass	<p>Defines the feature class of a 3D object represented by a spatial node</p> <p>Exactly one feature class is assigned to each leaf node. Inner nodes only have an own feature class, if they represent identifiable 3D objects (see Section 17.2.5 <i>Identifying 3D Objects</i> on page 320).</p>	optional	optional	X
aggregatedFeature3DClasses	<p>Stores the feature class information of the referenced child nodes of a reference node</p> <p>The feature class information of the child nodes of a reference node has to be stored explicitly as a property of the reference node, because the child nodes are located in a different BLOB.</p>	—	X	—

Properties	Description	Inner	Reference	Leaf
bodyGeometryReference	<p>References the BLOB with geometry information for the leaf node's 3D objects, and the index range of the elements within that BLOB</p> <p>The references are stored explicitly for leaf nodes. For inner nodes, the references are retrieved from the respective child nodes at runtime. For more information, see Section <i>17.5 References to Geometry Information</i> on page 335).</p>	optional	optional	X

## 17.2.2 Bounding Box

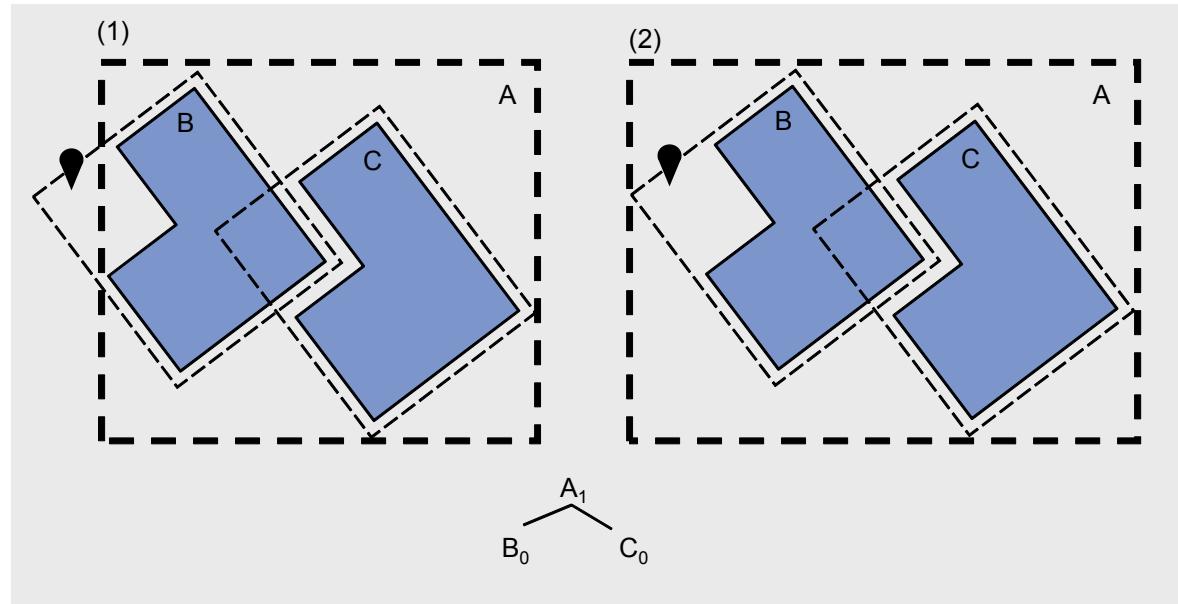
The bounding box defines the spatial extension of 3D objects represented by the subtree starting at that node. The bounding box gives the application information about the 3D objects of a specific area without actually retrieving the 3D object information itself. If, for example, the bounding box is not visible in a specific viewshed, the 3D objects contained in the bounding box will not be visible either, and, thus, the application does not need to retrieve the geometry information, nor does it have to further traverse the spatial tree.

DataScript location: `nds.objects3d.spatialnode > BoundingBox`

The bounding box covers the 3D object represented by a spatial node completely, including the bounding boxes of the 3D objects represented by the subtree starting at that spatial node. For spatial nodes without leave nodes, the bounding box covers the geometric representation in all levels of detail (see *17.3.2 Level of Detail for 3D Objects* on page 325). Bounding box A in Figure 17-3 (1) covers only the objects itself, but not the bounding boxes of the child nodes. In this example, the POI referencing object B would not be found: As it is not contained in bounding box A, the child nodes are not investigated further. Therefore, this option must not be used. In Figure 17-3 (2), the POI can be found (see also *NDS – Compiler Interoperability Specification, 14.2 Bounding Boxes* on page 205).

Nodes are split when they become too large.

Figure 17-3 Bounding Box

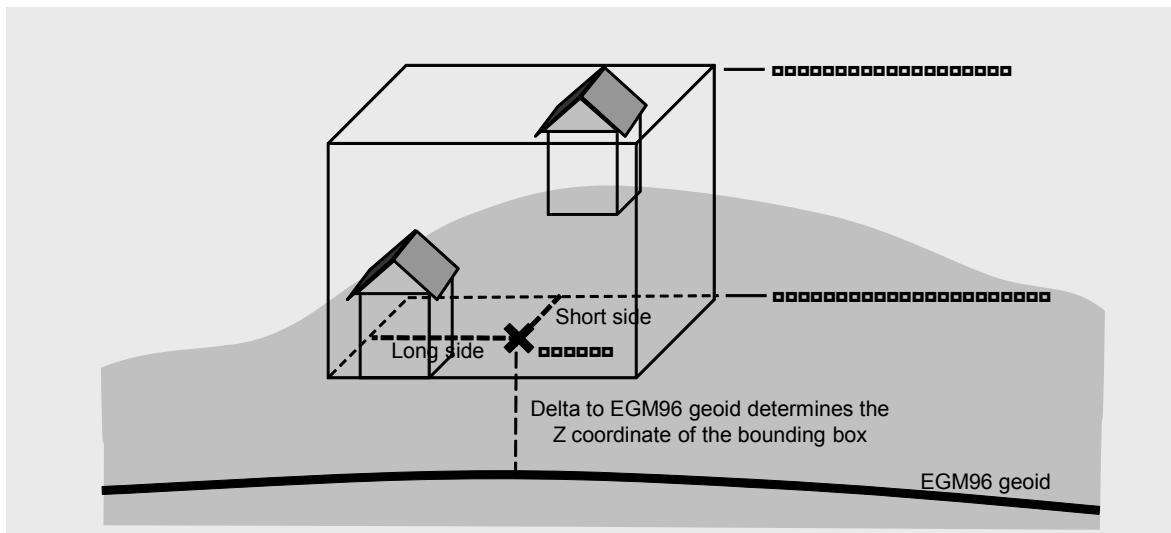


The bounding box contains the following information (see Figure 17-4):

- **origin**  
Defines the center point of a bounding box. **origin** is a 3D point. Its longitude and latitude match the center of the base area of the bounding box. The z coordinate equals the height of the base area above EGM96 geoid.
- **longSide**  
Spanning vector of the long side of the bounding box (2D vector)
- **shortSide**  
Spanning vector of the short side of the bounding box (2D vector)
- **topHeightOverGround, bottomHeightOverGround**  
Defines the vertical extent of the bounding box (integer)

**Note** The value for **bottomHeightOverGround** can be different from the height value of **origin**, if for example, only a part of the building is referenced.

Figure 17-4 Bounding box for 3D objects



The bottom of the bounding box is stored in centimeters relative to the EGM 96 geoid (DataScript: `nds.objects3d.spatialnode > origin.height`).

To fit the real-world requirements for 3D objects best, the two extents (short side and long side) are stored as independent `Vector2D`. This enables the compiler to rotate the bounding boxes, and also use parallelograms to adjust the bounding box to the position of the 3D objects. This is, for example, useful for cities. This is depicted in Figure 17-5 and Figure 17-6.

Figure 17-5 Rotated bounding boxes for 3D objects

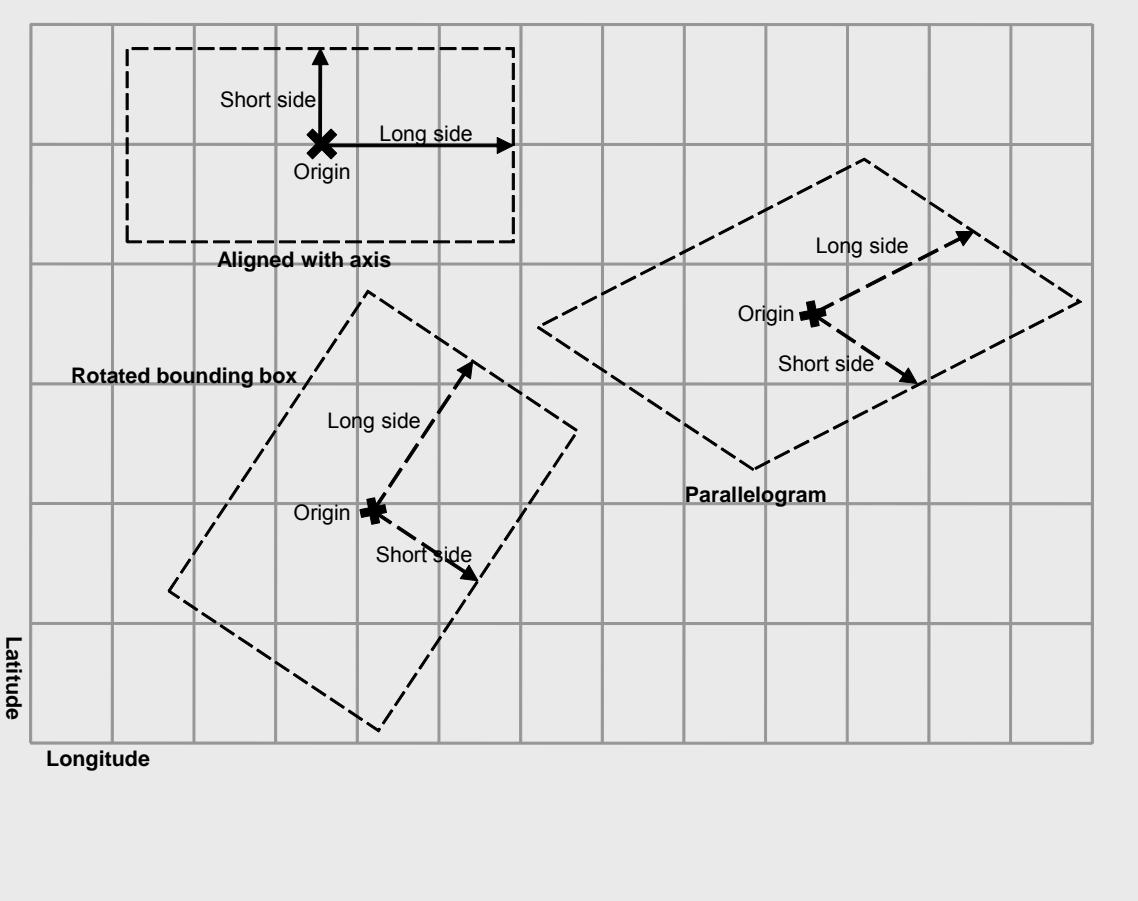


Figure 17-6 Use cases for rotated bounding boxes



### 17.2.3 Object Hierarchy Level

The object hierarchy level is stored as a fixed attribute at the spatial node. Together with the bounding box, the object hierarchy level is used for referencing a 3D object.

DataScript location: `nds.objects3d.spatialnode > objectHierarchyLevel`

Within a spatial subtree representing a 3D object, the object hierarchy level defines the actual position of the object in the tree hierarchy. It can, thus, be used to filter the spatial tree more efficiently:

- Tree search is facilitated, as the search does not need to traverse the tree to the leaf nodes.
- Parent and child nodes with identical bounding boxes can be identified unambiguously.

This is depicted in Figure 17-7: Object A and object B share the same center point, but can be identified unambiguously by means of their object hierarchy level. For references to object A, the search can stop at object A, without considering B and C.

*Figure 17-7 Use of object hierarchy level*

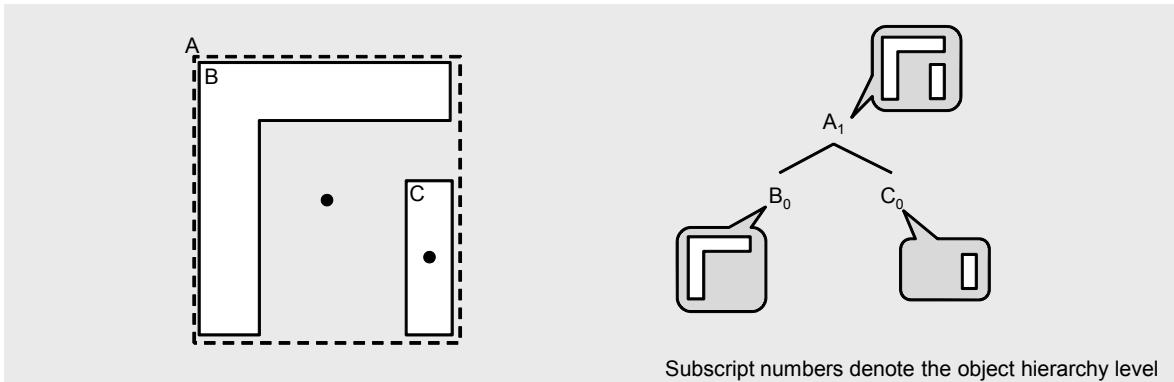


Table 17-4 shows examples for object hierarchy levels.

*Table 17-4 Object hierarchy level – Examples*

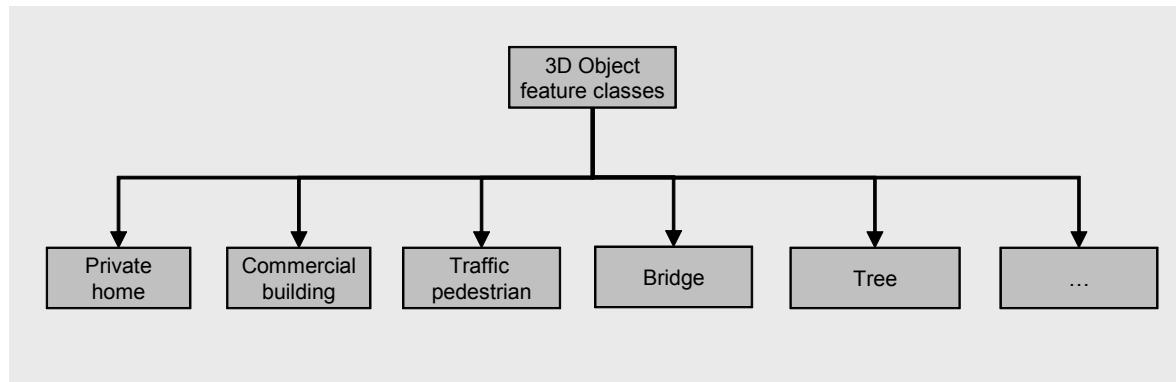
Value	Description
7	High level grouping, for example, a factory site with open space and several groups of buildings
1	Block level, for example, a group of factory buildings or office buildings
0	House
-1	Part of a house, for example, the top floor of a house
-8	No level; assigned to inner nodes, for which no useful grouping is available

### 17.2.4 Feature Classes for 3D Objects

The feature class is a classification, which is used to map 3D objects to real world objects. The application can use this information for providing corresponding styles. The 3D Objects building block makes use of the feature classes of the Basic Map Display building block: The type `feature3DClass` refers to `bmdFeatureClasses`, which lists the available feature classes of the Basic Map Display building block for a specific update region.

DataScript location: `nds.objects3d.spatialnode > feature3DClass`

*Figure 17-8 Examples for 3D object feature classes*



### Aggregated Feature Classes for Spatial Nodes

To support filtering functions, such as „highlight all tunnel roofs“, the feature class information of leaf nodes must be available in inner nodes and reference nodes. For inner nodes, the feature class is not stored explicitly, because: It can be retrieved at runtime by retrieving the feature classes of child leaf nodes, reference nodes, and inner nodes of the respective spatial tree BLOB.

The feature class aggregation cannot be used for reference nodes, however, as the respective leaf nodes are stored in a different BLOB. The aggregated feature class information for reference nodes is, therefore, stored in the `aggregatedFeature3DClasses` property.

### 17.2.5 Identifying 3D Objects

The spatial tree index can change during an update due to spatial nodes that are moved to a different BLOB. Therefore, the index cannot be used as a unique feature ID to identify a 3D object.

To uniquely identify a 3D object, the following information is required:

- Longitude
- Latitude
- Bottom height over ground
- Object hierarchy level

The longitude, latitude, and height over ground information identifies a point within a bounding box of a spatial node. This information can be used to find the respective node at runtime by traversing down the spatial tree.

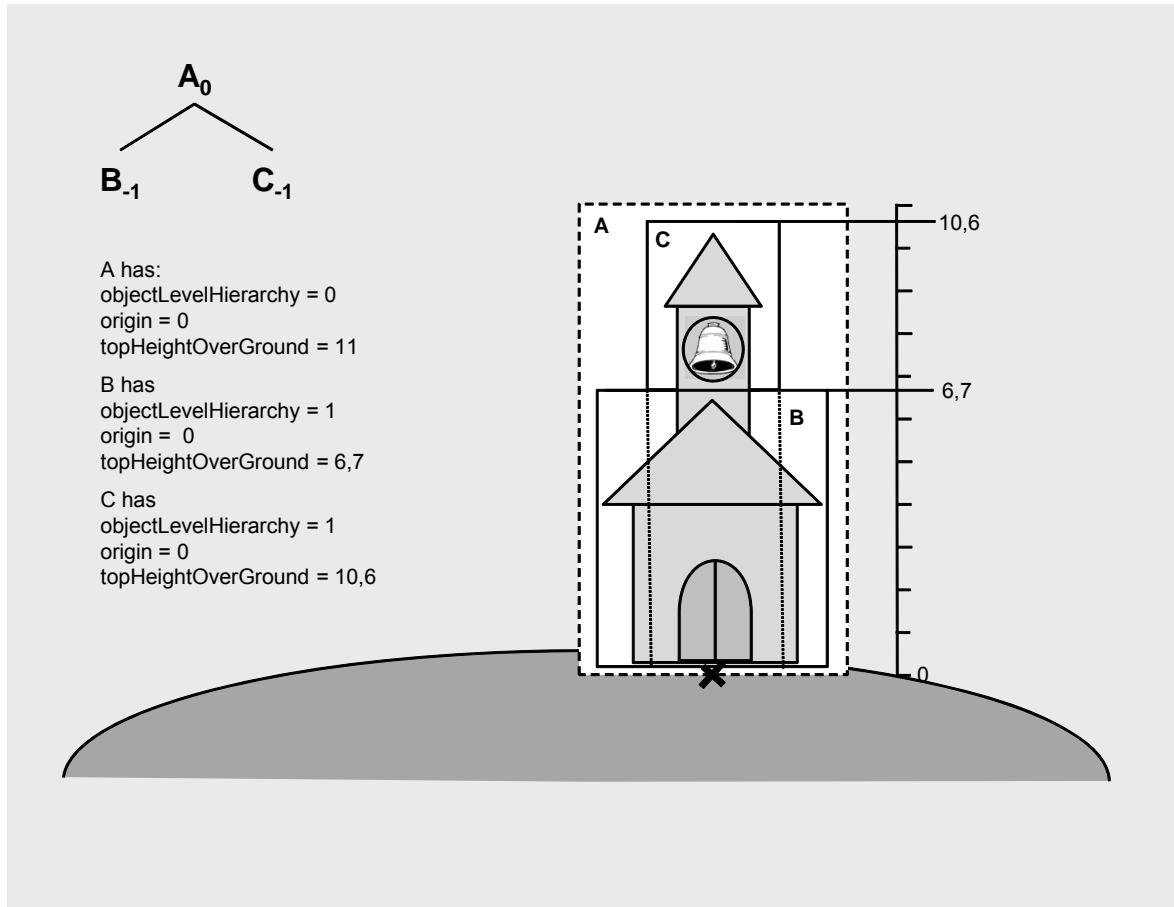
As bounding boxes may overlap, identifying the point is not sufficient to uniquely identify a 3D object. For 3D objects with parent-child relations, the object hierarchy level supplies the information required to uniquely identify the respective 3D object in the tree hierarchy.

For more information on overlapping bounding boxes, refer to *NDS – Compiler Interoperability Specification*, 14.2.1 *Bounding Boxes Sharing a Center Point* on page 205.

### Example for Identifying 3D Objects

A church consists of the main building and a bell tower. The longitude/latitude/bottom height information locate the 3D object within the bounding box of the 3D object „church“. The object hierarchy level is the same for the main building and the bell tower. The `topHeightOverGround` value, however, makes it possible to uniquely identify the bell tower.

Figure 17-9 Example for Identifying 3D Objects



## 17.2.6 References

3D Objects can have references to

- Named objects (see Section *References to Name Building Block* on page 336)
- POIs (more information to follow in the next release)

3D Objects can be referenced by POIs, see Section 17.8 *References to 3D Objects* on page 336

## 17.3 Geometry Information for 3D Objects

The 3D object geometry is defined by means of vertices and normals in a 3D coordinate system, using WGS 84 for latitudes and longitudes, and EGM96 for altitudes. NDS stores the geometry content for 3D objects in BLOBs, which are referenced by one or more spatial nodes. Each BLOB contains the geometry information for the 3D objects of one specific level of detail.

Table 17-5 shows the data stored in the body geometry BLOB (DataScript location: `nds.objects3d > bodygeometry > BodyGeometryBlob`).

*Table 17-5 Data stored in the body geometry BLOB*

Data	Description
<code>bodyGeometryBlobId</code>	Primary key; used for referencing the body geometry BLOB by the <code>SpatialNode</code>
<code>numXYBits</code> <code>numZBits</code>	Determines: <ul style="list-style-type: none"> <li>– Number of bits used for latitude and longitude</li> <li>– Number of bits used for altitudes (in centimeters)</li> </ul>
<code>coordShiftXY</code> <code>coordShiftZ</code>	Determines the coordinate shift value for latitude/longitude and altitude
<code>origin</code>	Defines the reference point for the coordinates of the geometry content
<code>bodyGeometries</code>	Array containing geometry information for 3D objects

Each BLOB contains a list of `BodyGeometry` structures, each of which represents one LOD of a 3D object. The representation can describe:

- A part of a building; this is used for huge buildings, which need to be split in parts, for example, a wing of the Louvre, or a single floor of a skyscraper
- A single building
- A group of buildings, for example, a set of houses in a city district

Table 17-6 shows the geometry information for 3D objects (DataScript location: `nds.objects3D.bodygeometry > BodyGeometryBlob > BodyGeometry`).

Table 17-6 Body geometry data

Data	Description
elevationDelta	Used to determine the absolute elevation of a body geometry: BodyGeometryBlob.origin.Z + elevationDelta = absolute elevation  For more information, refer to Section 17.3.1 <i>Elevation Information for 3D Objects</i> on page 324.
bodyGeometryType	Type of body geometry  NDS may support different variants of body geometry for displaying 3D objects. Currently, only the following variant is defined: Triangle strip without texture information (BG_INDEXED_TRIANGLE_STRIP).  For more information, refer to Section 17.3.3 <i>Body Geometry Type</i> on page 326.
bodyGeometryContent	Specifies the data type for the body geometry content  Currently only the bodyGeometryIndexedTriangleStrip type is defined.

## Vector in 3D Space

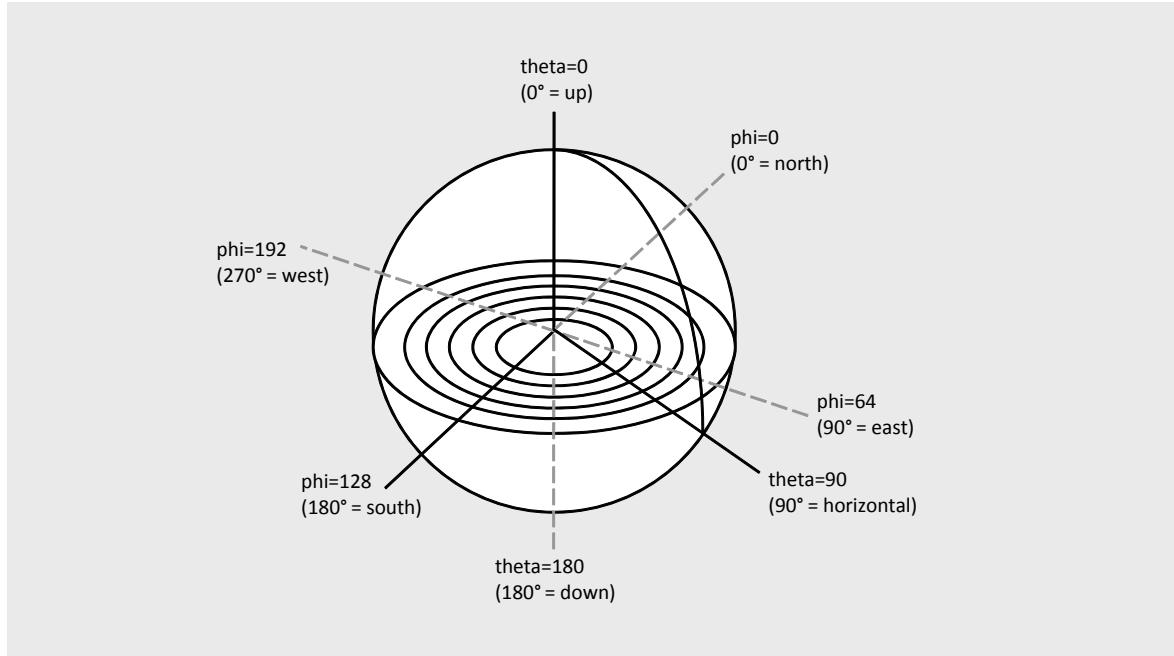
Vectors have a variable bit length for X/Y coordinates and Z coordinates (height). X/Y coordinates with less than 32 bits need an additional origin. Body geometry uses smaller values in order to save space; this means, they refer to a local coordinate system. For example, for the numXYBits value 15, the value range is [-32768..+32767] and specifies a spanwidth of ~590m on the equator for longitudes or ~590m for latitudes.

## Normal in 3D Space

Normals are defined by two angles:

- Phi defines the head on a compass, starting with 0 for north direction in clockwise orientation.
- Theta describes a range from 0° to 180°; 0° describes the upwards direction, 90° describes the horizontal direction, 180° describes the downward direction.

Figure 17-10 Normal sphere



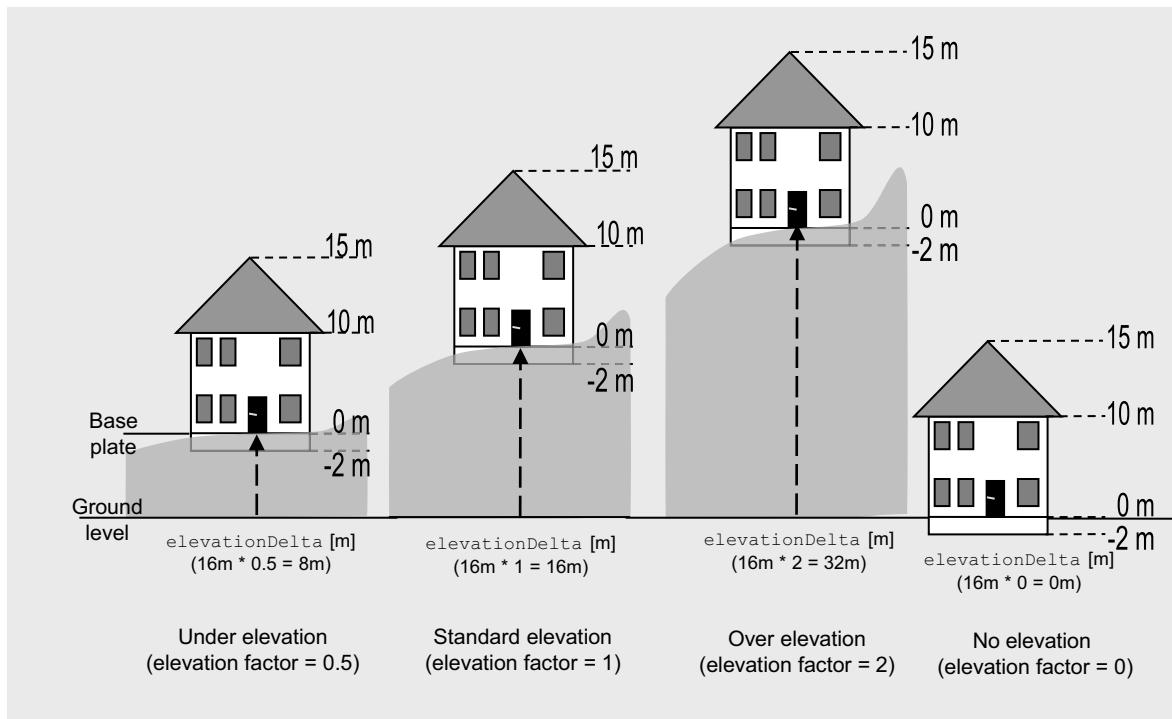
### 17.3.1 Elevation Information for 3D Objects

Elevation is defined as the distance in centimeters between the EGM 96 geoid and the 3D object's origin as defined in the `BodyGeometryBlob.origin.Z + elevationDelta` value.

Navigation systems may define a scaling factor for elevation to emphasize the impression of 3D in map display, or to highlight important buildings on the map. The elevation value is multiplied by the scaling factor. The building height remains the same.

Figure 17-11 shows a 3D object with 16m elevation and different scaling factors for elevation.

Figure 17-11 Elevation and scaling factor – Example



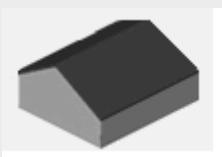
### 17.3.2 Level of Detail for 3D Objects

The level of detail (LOD) is used in computer graphics to decrease complexity of 3D objects. NDS defines four LODs (see Table 17-7).

For more information, refer to Section 17.5 *References to Geometry Information* on page 335.

Table 17-7 Level of detail for 3D objects

Level of Detail	Example
LOD 0 – Regional model / overview In LOD 0, a block of buildings is merged and depicted as one 3D object.	

Level of Detail	Example
<p>LOD 1 – Simplified city model (block model)</p> <p>In LOD 1, a simplified form is depicted. A cuboid, for example, is depicted to represent a building.</p>	
<p>LOD 2 – City model with textures and roof shapes</p> <p>In LOD 2, the simplified form is enhanced with texture information and shapes, such as a roof.</p>	
<p>LOD 3 – Detailed city model (architecture model)</p> <p>In LOD 3, as many details as are available for the 3D object are depicted. For buildings, for example, doors and windows can be displayed in LOD 3.</p>	

### 17.3.3 Body Geometry Type

The **BodyGeometryType** supports different variants for displaying 3D objects. Some buildings are complex and need to be modeled individually, most buildings, however, can be modeled by a small set of predefined standard buildings or template buildings (see [17.3.4 Template Body Geometry on page 327](#)).

The **BodyGeometryIndexedTriangleStrip** type is a basic geometry information type for a 3D object representation.

The triangle strip always contains two arrays of equal size, one array containing the vertices, and one array containing the normals. The elements of the two arrays, which belong together, form a tuple. This tuple is indexed by the index array of the render group (see [Section 17.3.5 Render Groups on page 328](#)).

The triangle strip may optionally contain the following arrays:

- Array for separate colors for each vertex
- Array containing texture coordinates
- Array containing additional texture coordinates; this array is used for faces which are rendered with two overlaying textures

All arrays contain the same number of elements, and the item indices correspond.

In addition to the above mentioned arrays, the `BodyGeometryIndexedTriangleStrip` contains an array for render groups. This array uses the data contained in the other arrays (see Section 17.3.5 *Render Groups* on page 328).

#### 17.3.4 Template Body Geometry

NDS supports the reuse of geometry information for recurring building types by means of template body geometries. Recurring building types share the same shape, or the same shape and texture. All Seven Eleven<sup>1)</sup> buildings in Japan, for example, are of the same shape and texture. Petrol stations may share the same shape, but have different textures.

DataScript location: `nds.objects3d > main`

Templates are defined globally in the `TemplateBodyGeometryTable` by means of a unique ID (`TemplateBodyGeometryId`) and a template body geometry (`TemplateBodyGeometry`). The template body geometry is defined similarly to body geometry, but uses a specific 3D vector (`TemplateVector3d`). The 3D vector is introduced in order to reuse templates worldwide without adjusting the aspect ratio (normalized in metric space). For a matrix with a neutral rotation composite, x represents longitude, and y represents latitude.

Body geometries can refer to template body geometries. For several objects with the same template, array types can be used.

The template body geometry type has the following attributes:

- Position
- Rotation and scale (optional)
- Template geometry reference: Reference to the ID of the actual template body geometry in the `TemplateBodyGeometryTable`

---

<b>Example</b>	An example for one position is a rotation and scaling invariant object, such as a tree. An example for an array of positions is a forest, that is, several trees. An example for rotation is a petrol station shown from another angle, for example, the entrance from northeast instead of the entrance from the south. An example for rotation and scale is a warehouse, which is scaled down by 10% and the entrance is shown from the southeast instead of the south.
----------------	---

---

Variants may be used for different levels of detail. In lower levels of detail, for example, the faces of a building may be displayed with a single color instead of textures. For such LOD-specific building variants, no texture coordinates are needed.

NDS provides the following body geometry types for template buildings:

---

1) Seven Eleven is a chain of convenience stores in Japan.

- BG TEMPLATE GEO TYPE POSITION
- BG TEMPLATE GEO TYPE POSITION ROTATION
- BG TEMPLATE GEO TYPE POSITION ROTATION SCALE
- BG TEMPLATE GEO TYPE POSITION ARRAY
- BG TEMPLATE GEO TYPE POSITION ROTATION ARRAY
- BG TEMPLATE GEO TYPE POSITION ROTATION SCALE ARRAY

### 17.3.5 Render Groups

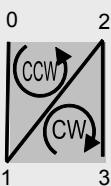
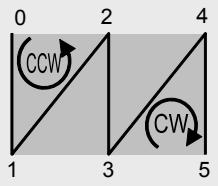
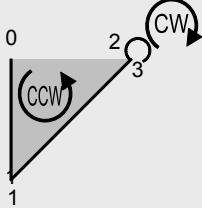
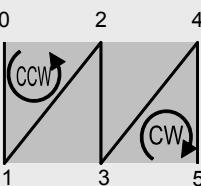
The polygons (triangle strips) created by the combination of vertices and normals are grouped into so-called render groups of arbitrary size.

DataScript location: `nds.objects3d.bodygeometry > RenderGroup`

A render group indexes at least four tuples (vertex/normal pairs). For each triangle that is added to the render group, additional vertices are needed. The number of additional vertices is defined in the `consecutiveQuads` value of the render group.

Figure 17-12 shows two examples for render groups with different `consecutiveQuads` values.

*Figure 17-12 Render groups*

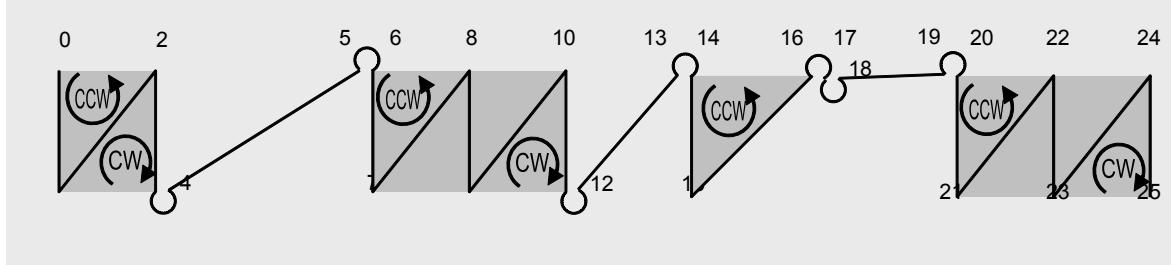
RenderGroup 1	RenderGroup 2	RenderGroup 3	RenderGroup 4
<code>████████████████ = 0</code> # of vertices = 4 # of triangles = 2	<code>████████████████ = 1</code> # of vertices = 6 # of triangles = 4	<code>████████████████ = 0</code> # of vertices = 4 # of triangles = 2 1 degenerated triangle, because the two last vertices share the same index number	<code>████████████████ = 1</code> # of vertices = 6 # of triangles = 4
			

A render group usually consists of an even number of triangles, starting with two triangles. The even number of triangles ensures that a counterclockwise winding can be retained across two render groups, and, thus, also ensures that the edge direction fits. For faces with an odd number of triangles, the last vertex is duplicated. This is depicted in render group 3 in Figure 17-12.

**Note** The winding alternates: Triangles with even index numbers (0, 2, 4) have a counterclockwise winding (CCW). Triangles with odd index numbers (1, 3, 5) have a clockwise winding (CW).

The graphic card video memory can connect the triangle strips of render groups with the same material or color by adding two degenerated triangles. This is done by doubling the last tuple of the first triangle strip, and the first tuple of the second triangle strip. This is depicted in Figure 17-13.

Figure 17-13 Connecting render groups by adding degenerated triangles



For each render group, the following information can be defined either in the `bodyGeometryIndexedTriangleStrip` or in the `TemplateBodyGeometry`:

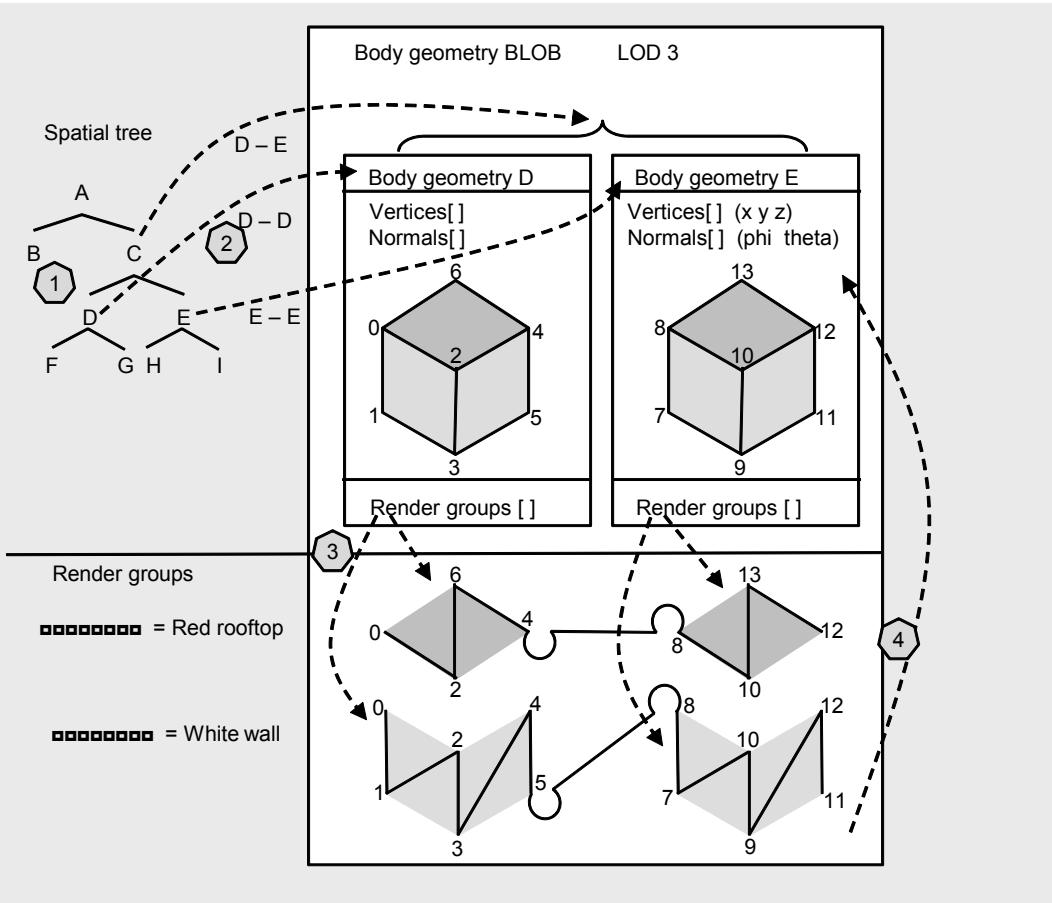
- Color  
The color can be defined either per vertex, or globally.
- Texture yes or no  
If `hasTextureCoords` is true, one or two textures can be specified for the render group.

Render groups can be used, for example, to draw all red roofs in one step. Thus, it is not required to change colors while drawing.

### 17.3.6 Using the 3D Objects Data Structures

Figure 17-14 gives an overview of the architecture of the 3D Objects building block.

Figure 17-14 Architecture of 3D Objects building block



The numbers 1 to 4 in the figure refer to the following steps:

1. Traverse the spatial tree to find the spatial node, which has to be rendered.
2. Look up reference to body geometry.
3. Look up references to render groups per material (the render groups carry material properties).
4. The index refers to vertices and normals, texture coordinates, and extended texture coordinates within the BLOB.

## 17.4 Material and Textures

Materials describe the surface of 3D objects. In computer graphics, a natural material is represented either by means of constant color values, or by textures, and how the surface interacts with light.

The following lighting properties are known for material definition:

- Ambient lighting: Describes the object color with a constant value and simulates indirect lighting from the environment
- Diffuse lighting: Describes the effect of a dim surface, which emits light evenly in all directions
- Specular lighting and shininess: Defines the color, and its intensity, for specular reflected light. The shininess defines how strong the material reflects specular light. The larger the value is in the range from 0..128, the smaller the light reflection spot is.
- Transparent: Identifies transparent material; this information is required to determine the drawing order, as transparent material always needs to be processed after opaque material.

Material content is defined in NDS by a texture type and the following lighting properties:

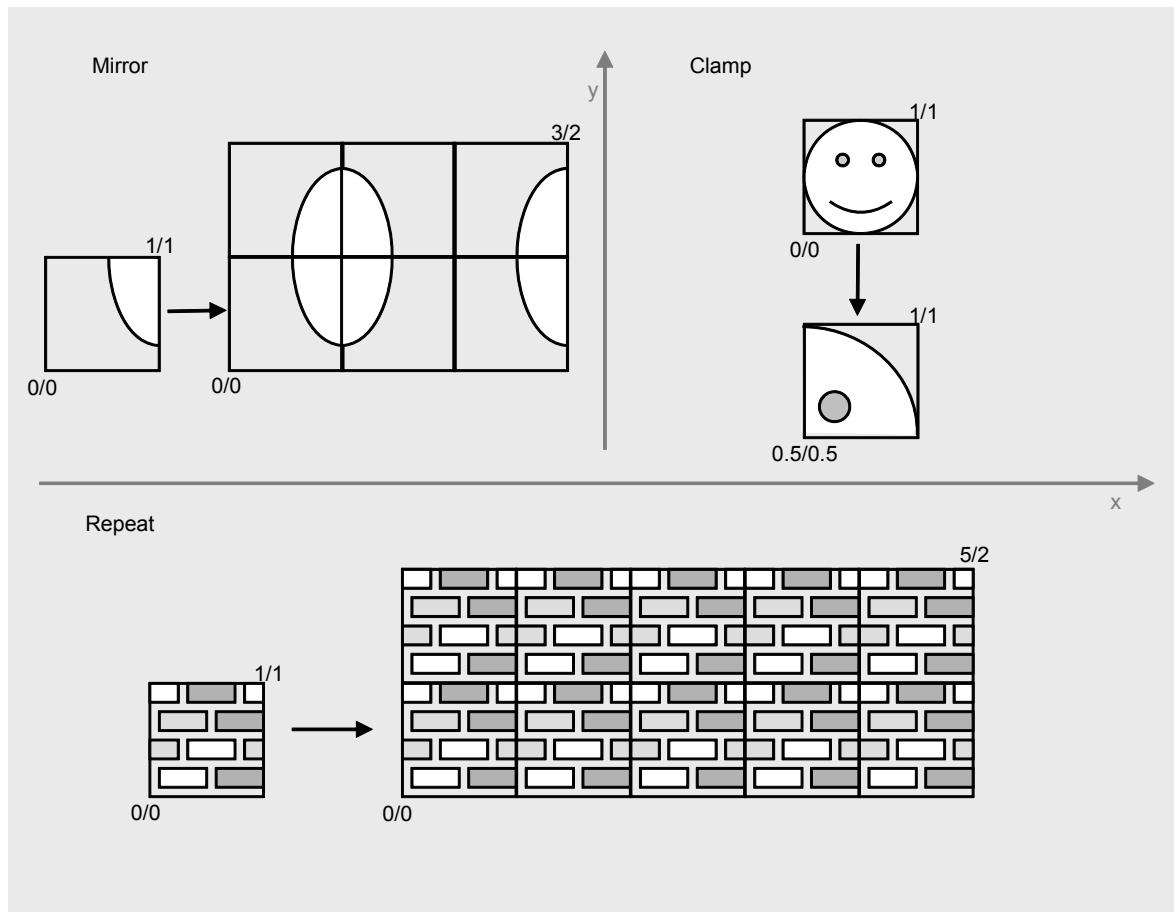
- `hasAmbientAndDiffuse`
- `hasSpecularAndShininess`
- `isTransparent`

A material may refer to a texture by means of texture references (`TextureReference`). The reference refers to one texture (`SubImage`) within a texture map.

The texture type defines how textures are used. The following texture types are provided in NDS (see Figure 17-15):

- Clamp: Texture coordinates reduced to the range
- Repeat: Integer part of the texture coordinates is ignored.
- Mirror: Vertical mirror, if integer part of x is odd; horizontal mirror, if integer part of y is odd

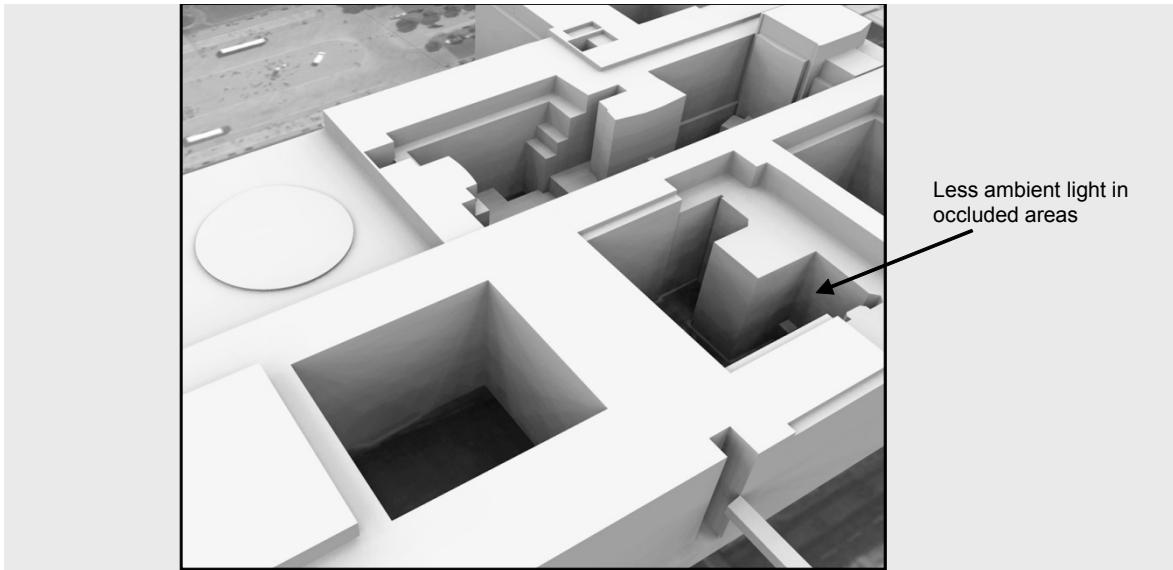
Figure 17-15 Texture types



## Color Gradients

The type `hasColorPerVertex` is used to model a color gradient, or for occlusion. Color gradients can, for example, be used to display verdigrised copper roofs. Occlusion is used, for example, for shadowed courtyards, where the occluding buildings influence the brightness of the concealed walls (see Figure 17-16).

Figure 17-16 Occlusion



## Textures

Textures describe the surface structure of 3D objects, such as brick walls, rough plaster, and so on. The texture information is stored as bitmap images in the texture map table. The images can be stored in different formats. Besides JPEG and PNG, more hardware-oriented formats are supported. The texture map is applied to the surface of a 3D object by means of UV mapping (see [http://wikipedia.org/wiki/UV\\_Mapping](http://wikipedia.org/wiki/UV_Mapping)).

DataScript location: `nds.common.texture > TextureMapTable`

The following two texture types exist:

- Pattern textures that can be strung together, such as brick walls
- Specific textures, such as brand names on entrance doors

In NDS, textures are stored globally as sub images in the texture map table.

Table 17-8 Properties of the texture map table

Property	Description
<code>textureMapId</code>	Unique ID for a texture map
<code>textureImageFormat</code>	Format of the texture image. The following formats are allowed: <ul style="list-style-type: none"> <li>- PNG</li> <li>- JPG</li> <li>- JP2000</li> <li>- DDS</li> <li>- PVRTC</li> <li>- ETC</li> <li>- KTX</li> </ul>
<code>image (BLOB)</code>	Bitmap file with image content

Property	Description
subImageArray	Describes the location of a single texture in the texture map

### Applying textures to 3D Objects

For applying texture maps to 3D objects, NDS maps the local texture coordinate system to the vertices of the body geometry of the 3D object. For this, texture coordinates are used. They are defined as `float16`. The reference point for the local coordinate system ( $x=0, y=0$ ) is the bottom-left corner of the respective texture.

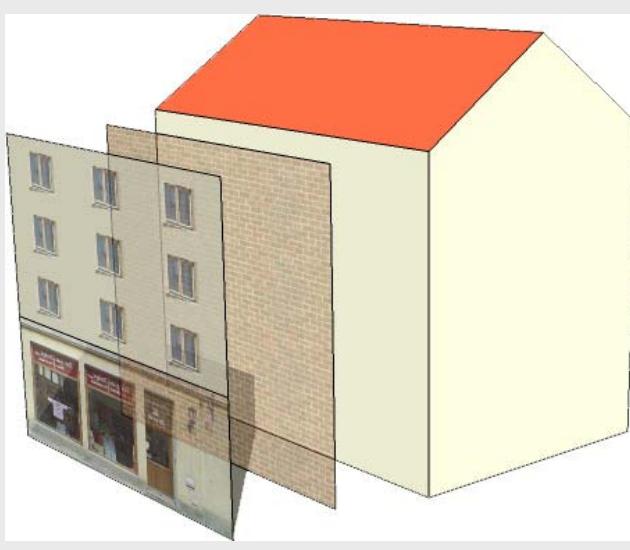
If a texture map contains no sub image, repetitions are possible. If a texture map contains several sub images, then pattern mode may be used (no repetitions), meaning that a part of the texture can be mapped to a surface.

It is possible to apply several texture layers to a 3D object. To display a building with a brick wall, windows and doors, for example, a first texture layer representing a brick wall is rendered, then a second layer representing windows and doors (see Figure 17-17).

The two layers are modeled by means of `hasTextureCoords` (contains the first layer) and `hasTextureCoordsAdditional` (contains the second layer).

If some parts of the overlying texture have to be made transparent, an alpha value is used.

Figure 17-17 Texture layers



## 17.5 References to Geometry Information

The nodes of the spatial index structure reference the **BodyGeometryBlob** by means of the unique BLOB ID. As each node contains at least one LOD representation of 3D objects, the reference also contains information about the LOD of the 3D object, for which geometry information is needed.

The **bodyGeometryReference** is a bit mask, which indicates the availability of levels of detail. The mask must be filled sequentially without gaps as follows:

- If the most significant bit (MSB) is set(1), LOD 0 is available.
- The next bit indicates that LOD 1 is available.
- The next bit indicates that LOD 2 is available.
- If the least significant bit (LSB) is set(1), LOD 3 is available.
- If a bit is reset (0), the respective LOD is not available.

Figure 17-18 depicts an example with four LOD references.

*Figure 17-18 Example: Inner node with four LOD references*

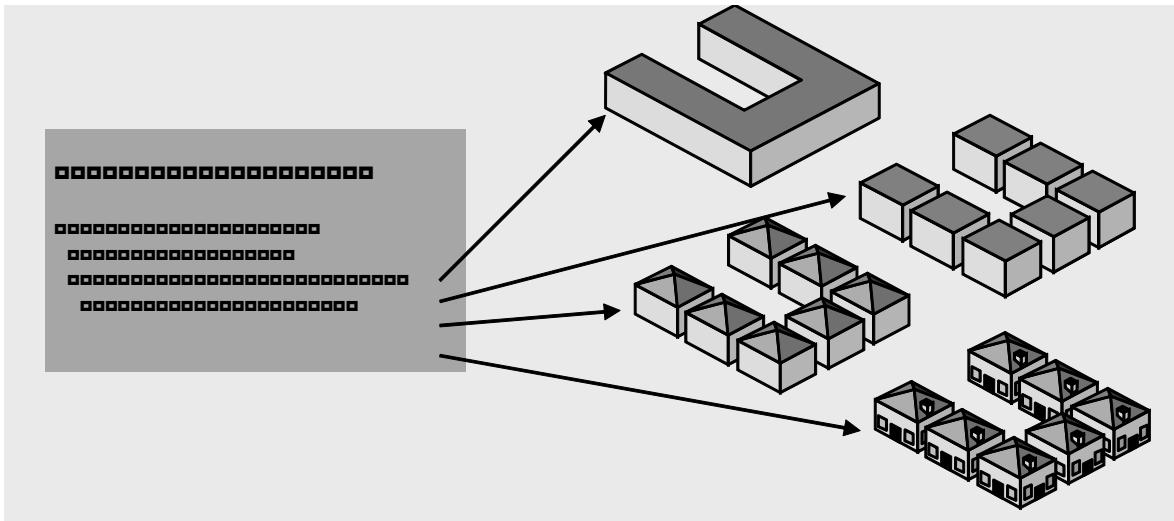
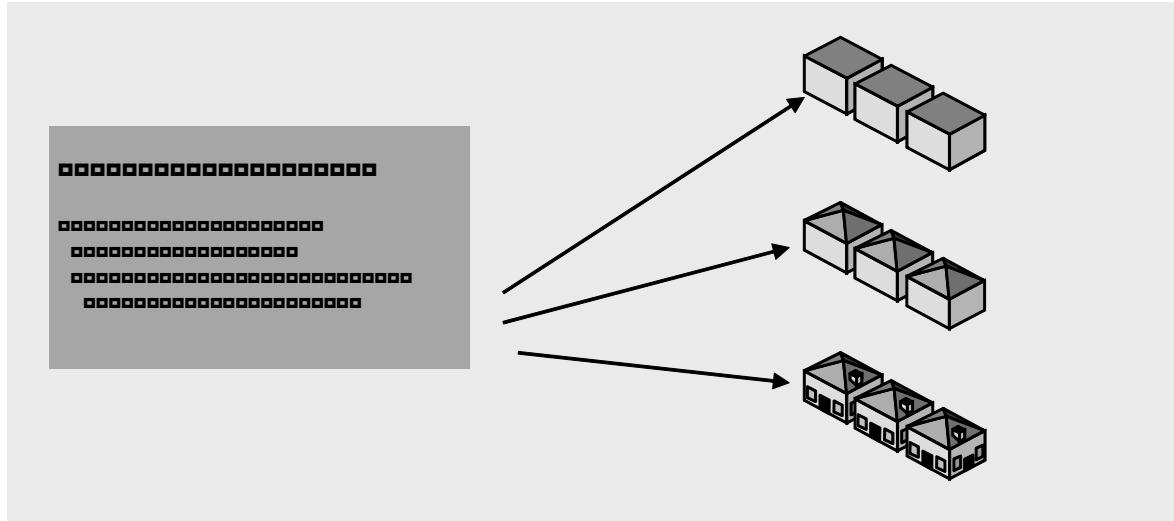


Figure 17-19 depicts the same example for a child node. In this case, not all LOD representations exist. The block of seven houses from Figure 17-18 is split into smaller blocks, and the spatial node in Figure 17-19 only refers to three buildings. The U shape representation in LOD 0 does not exist.

Figure 17-19 Spatial node without LOD 0 reference



## 17.6 References to Name Building Block

The following flexible attributes are used for references from the 3D Objects building block to the Name building block:

- NAMED\_OBJECT\_REFERENCE
- NAMED\_OBJECT\_BASE\_REF
- NAMED\_OBJECT\_RELATIVE\_REF
- FLOOR\_NUMBER

## 17.7 3D Objects and Building Footprints

3D objects can be available for area features of the Basic Map Display building block. If available, these 3D objects can replace a building footprint or a 2.5D city model. For more information, refer to 11.12 3D Objects Representing Building Footprints on page 226.

Rendering of both, 3D objects and 2D building footprints/2.5D city models should be avoided, however, as this may lead to unwanted visual effects.

## 17.8 References to 3D Objects

3D objects are referenced by means of references set to the spatial node that represents the respective 3D object in the spatial tree.

There are two ways to reference 3D objects:

- References to single houses (`objectHierarchyLevel` is set to 0, `heightOverGround` is set to 0)

For references to single houses, the longitude and latitude information is sufficient to uniquely identify the spatial node. `objectHierarchyLevel` and `heightOverGround` can be set to default value (0). This is depicted in Figure 17-21.

---

**Note** This simple referencing method can be used, if the following conditions are met:

- The coordinates of the referencing object, for example a POI, are located within the bounding box of the referenced spatial node.
  - No other spatial node exists with object hierarchy level 0 and a bounding box with a center point closer to the coordinates of the referencing object.
- 

- References to groups of buildings or parts of a building

For references to 3D objects that are not single houses, the `objectHierarchyLevel` information is required in addition to the coordinate information. This is required for the following references:

- References to a group of buildings, such as a university campus
  - References to a part of a building, such as a clock tower of a church
  - References to objects in a parent-child relation, whose bounding boxes share the same center point
- 

**Note** This referencing method can be used if the following conditions are met:

- The coordinates of the referencing object, for example a POI, are located within the bounding box of the referenced spatial node.
  - No spatial node with the same object hierarchy level and a bounding box center point closer to the POI exists.
-

Figure 17-20 3D object references by means of coordinates and object hierarchy level information

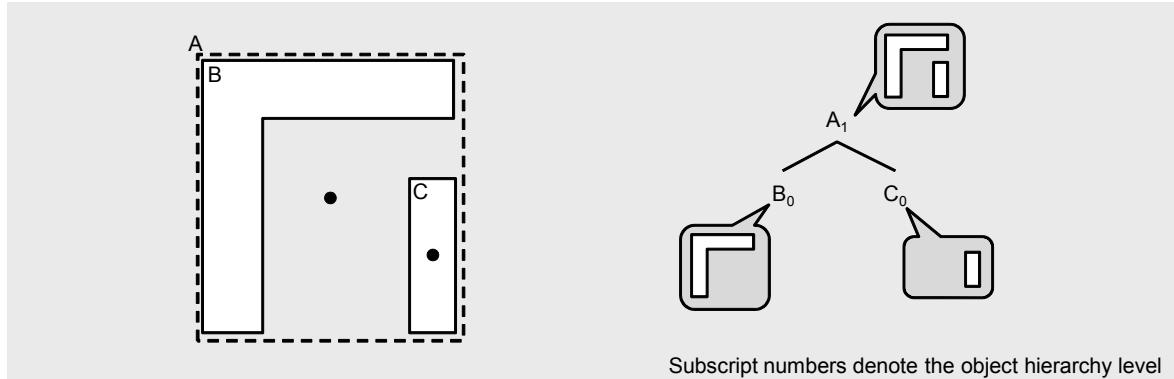


Figure 17-21 3D object references by means of coordinate information only

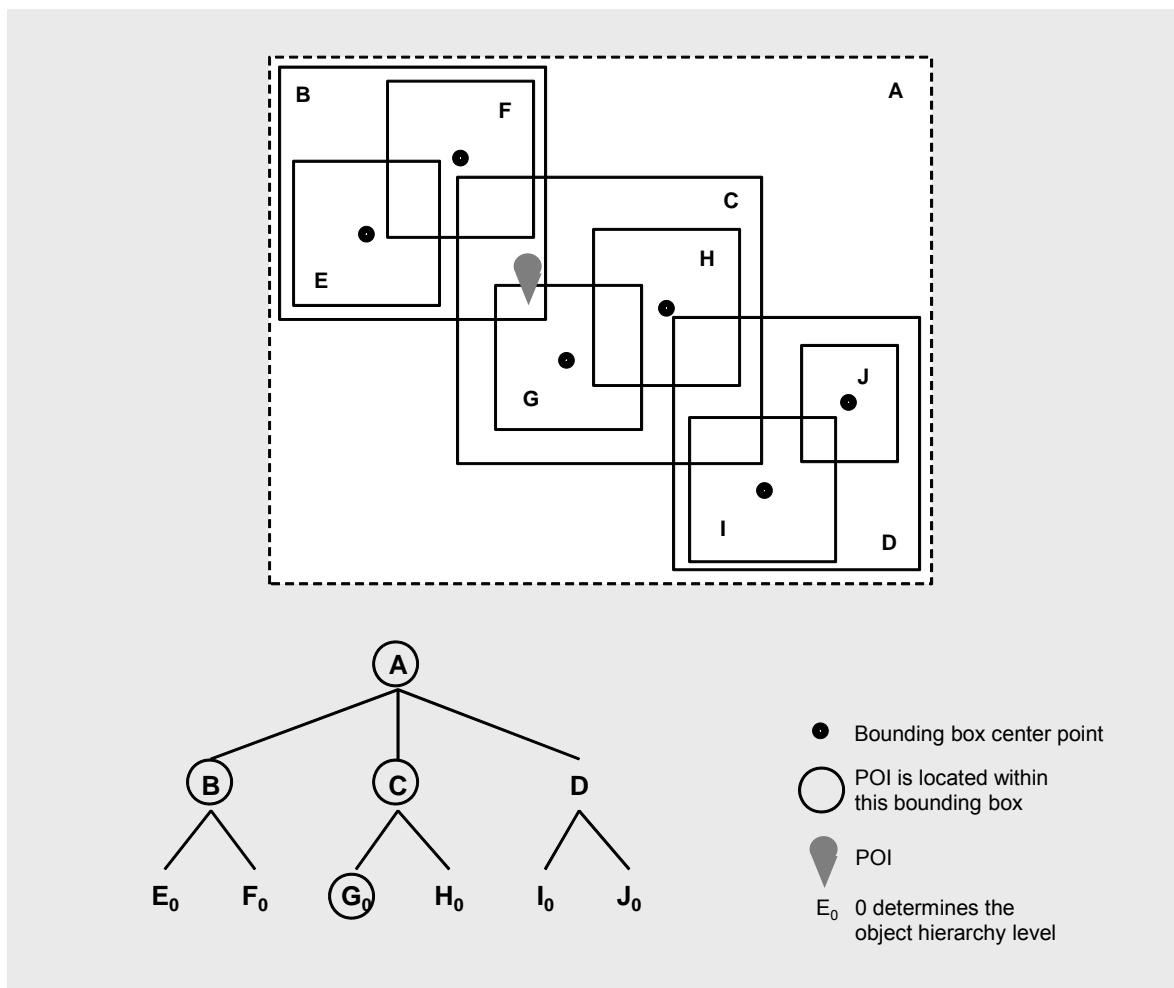


Figure 17-22 3D object references by means of coordinate and distance to center point

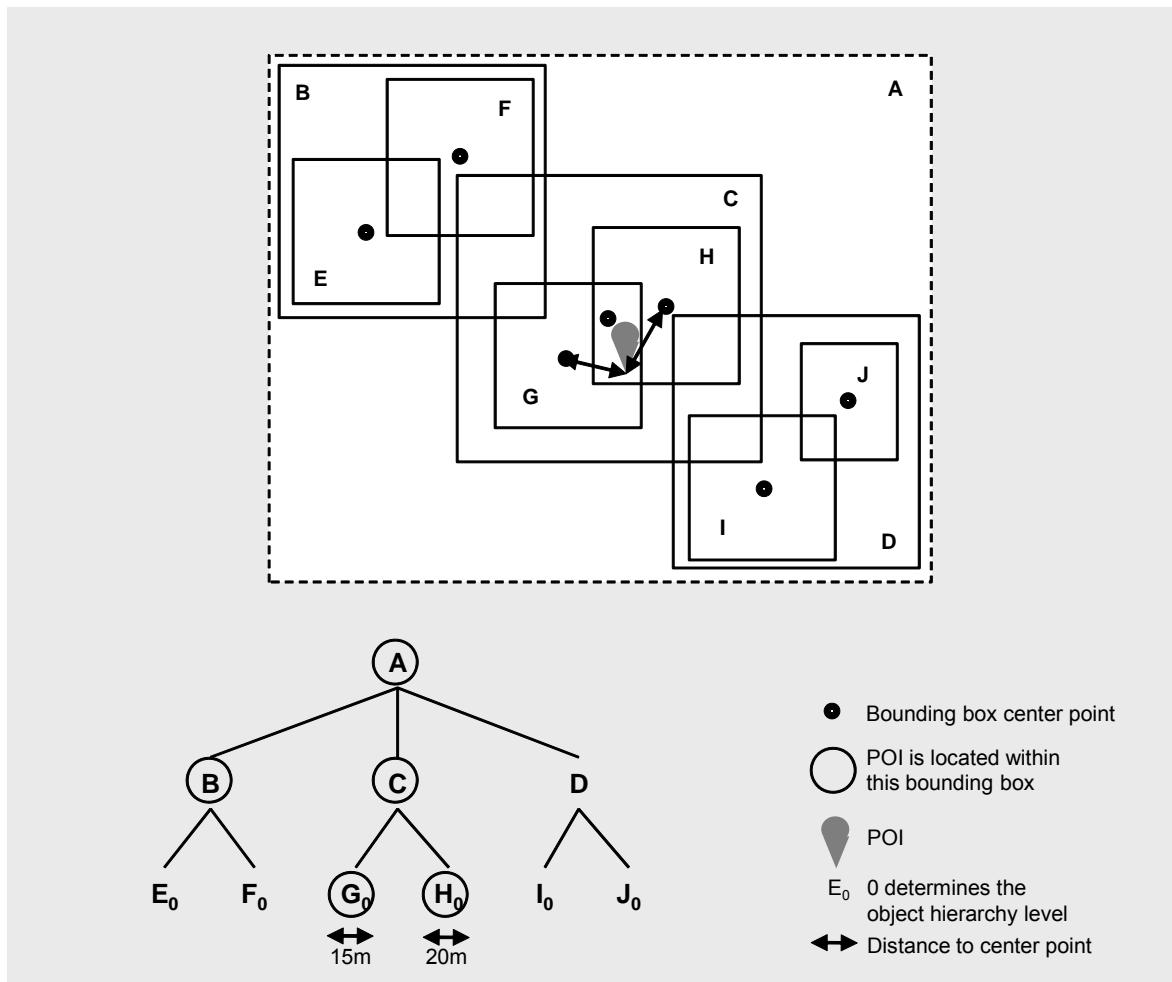
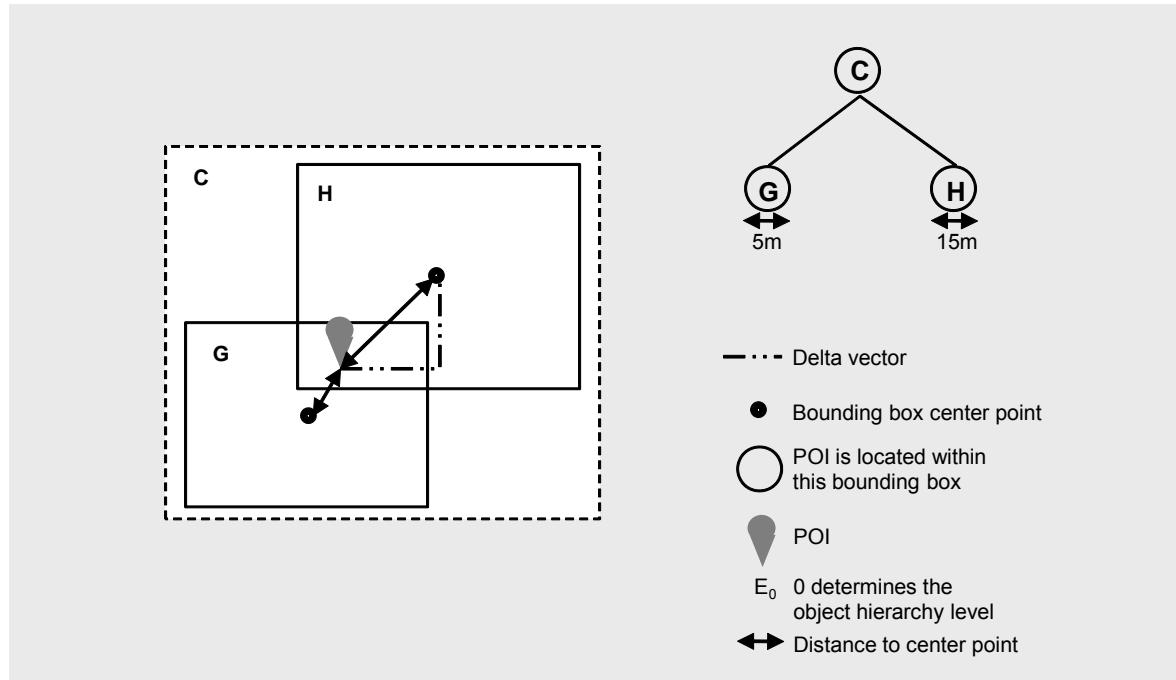


Figure 17-23 3D object references – delta vector



#### Related Topics:

- *NDS – Compiler Interoperability Specification, 14.3 References into 3D Objects Building Block* on page 206
- *NDS – Compiler Interoperability Specification, 10.9 References from POIs to 3D Objects* on page 190

## 17.9 Landmark Icons

In 3D maps, 3D models of landmarks can be displayed near the center or the viewing point. Rendering 3D models for landmarks further away, however, requires too much processing time. In these cases, the 3D models may be replaced by 2D icons.

Landmark icons are stored as flexible attributes and referenced by the `iconSetId` in the `landmarks3DIconsTable`. The `SpatialTreeNode` contains a simple attribute list. If the Boolean `hasAttributes` is set to TRUE, a 2D landmark icon exists and can replace the 3D model of the respective landmark stored with the spatial node.

## 18 Junction View Building Block

Route calculation results in a sequence of links on level 13 representing the calculated route from the starting point to the destination. The maneuver generator in the application uses the route to supply additional information about the route to the driver, for example, the picture of a roundabout with symbolized arrows and/or audio commands, for example, “leave the roundabout at the 3rd exit”.

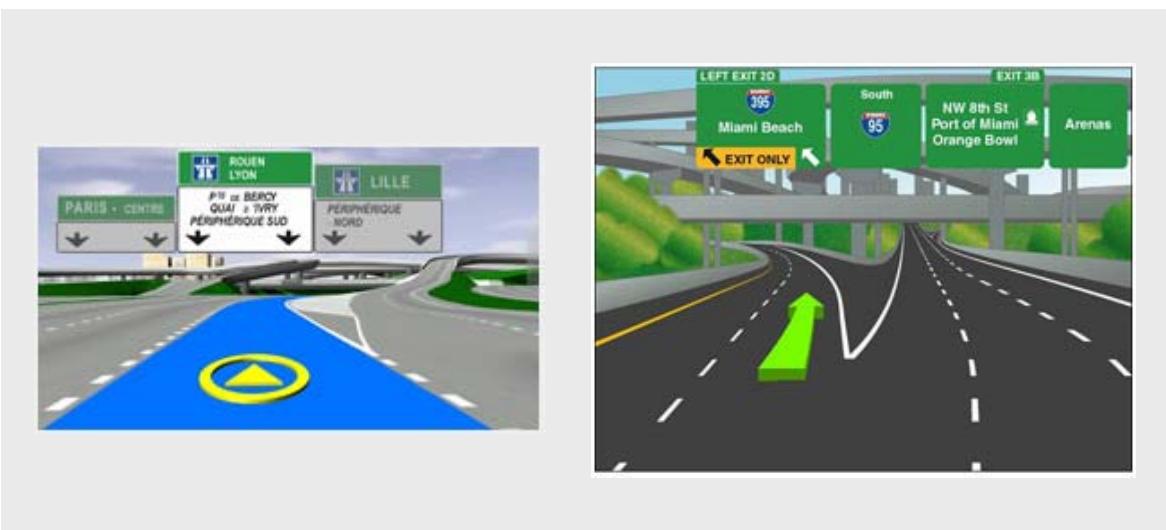
Junction views give the driver realistic visual information of a junction and the required maneuver. For this purpose, NDS provides a standalone database for junction view information. The flexible attribute `HAS_JUNCTION_VIEW` (DataScript location: `nds.common.flexibleattributes`) can be assigned to links and indicates that junction view information is available for the respective link.

Figure 18-1 illustrates an example of a junction view image; Figure 18-2 shows examples of three-dimensional junction views.

*Figure 18-1 Junction view image*



*Figure 18-2 3D junction view*



The junction view image may either be stored completely in the database or may be generated on-the-fly by the navigation system by laying several individual transparent images contained in the database on top of each other, for example, background images, junction images, sign board images, and arrow images.

For more information on generating junction views, refer to *NDS – Compiler Interoperability Specification, 15 Junction Views* on page 211.

## 18.1 Building Block Structure and Content

The Junction View building block is an optional building block, and its data logically belongs to the Routing building block (see Chapter 9 *Routing Building Block* on page 107). The junction view database is organized relationally with the following tables:

- **JvFromLinkTable:** This table contains the mapping between routing information (`fromLink`) and the corresponding images (`imageID`). The position on the link (`position`) defines where an image (or several images) has to be displayed. A sequence number (`seqNr`) defines the drawing order for a group of overlayed images.
- **JvFrom2ToLinkTable:** This table contains the mapping between routing information defining a path (`fromLink` and `toLink`) and the corresponding images (`imageID`) illustrating the path, for example, arrows indicating the lanes to use. Each path has an assigned number (`pathNr`). If several paths lead from `fromLink` to `toLink`, the `pathNr` is used to distinguish the paths. A link list (`intermediateLinksList`) defines a set of links between `fromLink` and `toLink`. This link list is optional. A sequence number (`seqNr`) defines the drawing order.

The application derives junction IDs from the combination of `fromLink` and `toLink`.

- **JvImageTable:** The images used to compose junction views are stored as BLOBs in this table. Attributes enable the application to find the suitable picture for the current situation, for example, day or night view of the junction (see Figure 18-3). The BLOB may also contain a 3D model of the junction (known as *3D junction view*).

---

**Note** By default, references from the Junction View building block to links in the Routing building block use the feature ID. However, these references may also be modeled by means of a global feature ID. In this case, the flexible attribute `GLOBAL_FEATURE_ID` must be assigned to the related link. See also Section *Identifying Features* on page 50.

Using global feature IDs enables references to Routing building blocks of different NDS products and thus reuse of the junction view database for several NDS product databases.

---

The `imageMirrored` flag is stored in the 2D junction view tables. If the flag is set, it indicates that the application should display a mirrored version of the respective junction view image at runtime.

DataScript location: `nds . jv . main > JvFromLinkTable` and `JvFrom2ToLinkTable`

Table 18-1 shows the attributes that are assigned to each image.

*Table 18-1 Attributes for junction view images*

Attribute	Description
<code>imageType</code>	Defines the image content, for example junction, background, signboard, arrow
<code>dayLight</code>	Indicates whether the image can be used for day or night or whether it can be generally used
<code>weather</code>	Indicates the weather situation of the image, for example rain, snow, sunshine
<code>colorDepth</code>	Indicates the color depth of the image, for example whether 8, 16 or 32 bits are used to encode the image color
<code>jvImageFormat</code>	Images have to be stored either in SVG or PNG. 3D models of junctions have to be stored as 3D or digital terrain features according to the definitions of the NDS 3D and DTM building blocks.
<code>width</code>	Image width in pixel
<code>height</code>	Image height in pixel
<code>position</code>	Indicates the position of a junction view image This information is required to distinguish between several junction view images for the same link.

*Figure 18-3 Day view (left) and night view (right), example*





## 19 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 objects 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 Section 10.6 *Next-valid-character Trees* on page 186).

The FTS building block is an optional building block.

Full-text search denotes a technique for searching computer-stored documents. The user supplies search strings. A search engine uses these search strings to match them with the words in documents. A full-text search index structure basically consists of an n-to-m relation between the search strings (so-called *terms*) and the documents. For a given query, the full-text search library typically returns all document IDs which contain the terms supplied in the query.

---

**Example** A user enters the following search strings: **Otto 1 Volger**.

Based on these search strings, the full-text search returns just a few hits for a complete European database, among them the NAVTEQ office in Sulzbach, which is located at Otto-Volger-Straße 1.

---

For the current version of NDS, full-text search is implemented for documents based on POI and Name data. The implementation allows applications to support a convenient one-shot destination input for POIs and addresses. Based on the search strings **McDonald**, **Leopold**, **Schwabing**, for example, the application supplies a list of all McDonald's restaurants located in the Leopoldstraße in the city district Schwabing.

The source data for the documents is taken from a given instance of a POI or Name building block, the so-called *source building block*. An NDS database can contain more than one instance of a source building block and, therefore, it is also possible to have more than one instance of the Full-text Search building block in an NDS database.

FTS data refers to the source data via a document ID. The uniqueness of this ID is only guaranteed within the scope of a single building block. Therefore it is not possible to have only a single instance of a Full-text Search building block for an NDS database with more than one source building block.

---

**Note** Full-text search differs from fuzzy search. In addition to the full-text search tables in the Full-text Search building block, Double Metaphones may be used in search documents in order to extend the FTS building block and cover fuzzy search use cases. For detailed information, refer to *Fuzzy Search Documents with Double Metaphones* on page 351.

---



---

**Note** The full-text search supported by NDS bases on the SQLite full-text search module fts3. For information on the use of fts3, refer to <http://www.sqlite.org/cvstrac/wiki?p=ftsUsage>.

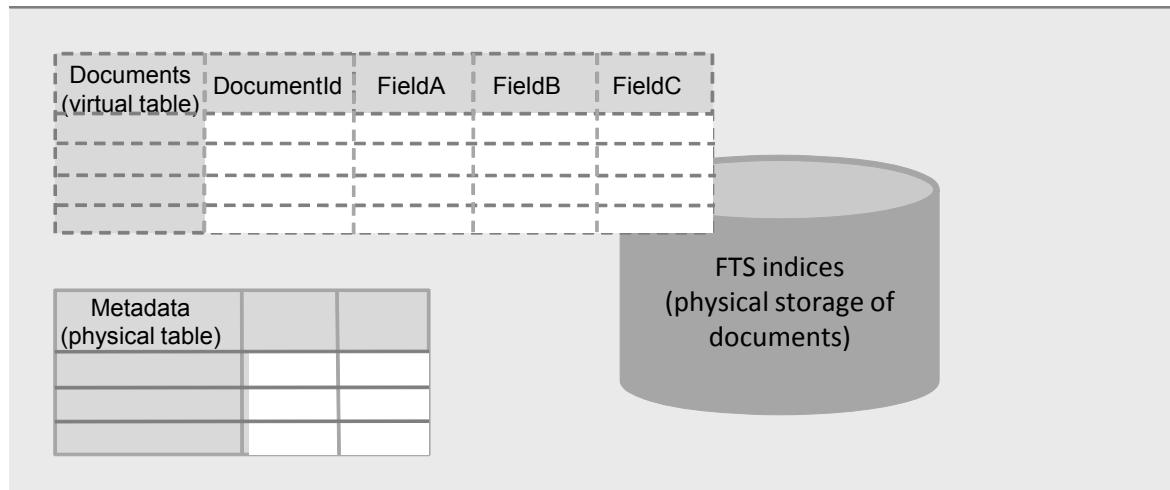
---

## 19.1 Building Block Structure and Content

The Full-text Search building block consists of full-text search (FTS) documents and metadata. The FTS documents are stored in the FTS indices. Both, the application and the compiler do not directly access the FTS indices in order to work with the FTS documents, but access the indices via an SQLite virtual table (DataScript location: `nds.fts.poi > PoiFtsTable`, as well as `nds.fts.name > RoadFtsTable` and `crossroadFtsTable`). The FTS metadata is stored in an SQLite table.

Figure 19-1 depicts the structure of the Full Text Search building block.

Figure 19-1 Structure of the Full Text Search building block



The full-text search indices are created during compilation by the fts3 engine. The full-text search indices are managed by the fts3 engine, and not by the application or the compiler. For more information on the full-text search indices, refer to *NDS – Compiler Interoperability Specification, 16.3 SQLite fts3 Index* on page 216.

The update region concept of NDS also applies to the FTS building block. Therefore, data contained in a given instance of a Full-text Search building block is restricted to a specific update region and is generated using input data taken from the same update region.

## Full-text Search Documents

Each full-text search (FTS) document represents one NDS feature, for example, a POI from an instance of a POI building block or a road named object from the Name building block. The compiler generates the FTS documents from the NDS features and passes them to the FTS engine which stores them in the FTS index.

Each FTS document consists of a unique document ID and a specific number of fields.

The document ID is used as follows:

- To identify documents within a result set of a query
- As a reference to the feature of the source building block, for example, a reference to the POI represented by the FTS document

The fields are used to store specific attributes of the FTS document. The FTS document representing a POI, for example, can contain the *Category* field to store the POI category. The application can use the field data for the definition of queries for the full-text search. The field list of an FTS document defines the schema for the SQLite virtual table. This is illustrated in Figure 19-1. For examples regarding FTS documents, refer to Section 19.2 *Examples for Search Documents* on page 348.

## Full-text Search Metadata

The Full-text Search building block contains the following metadata:

- Building block ID to identify the building block containing the source data for the full-text search (source building block).

---

<b>Note</b>	The Full-text Search building block and the corresponding source building block must belong to the same product database. To ensure this, the update region ID of the source building block is equivalent to the update region ID of the Full-text Search building block. Therefore, it is not necessary to store it explicitly.
-------------	--

---

- FTS engine-specific metadata. For more information, refer to *NDS – Compiler Interoperability Specification*, 16.5 *Metadata of the SQLite fts3 Module* on page 218.

DataScript location of the full-text search metadata: `nds.fts.metadata > FtsMetadataTable`.

## Full-text Search Index

The full-text search index consists of terms (tokens). The index is a so-called inverted index: Each term refers to all documents to which the term is linked via the list of document IDs. During compilation, the entries are written into the index.

For more information, refer to *NDS – Compiler Interoperability Specification, 16.5 Metadata of the SQLite fts3 Module* on page 218. For general information on indexing processes for full-text search, refer to [http://en.wikipedia.org/wiki/Full\\_text\\_search](http://en.wikipedia.org/wiki/Full_text_search).

## 19.2 Examples for Search Documents

Each FTS document consists of a unique document ID and a specific number of fields. The fields are used to store specific attributes of the FTS document. The application can use the field data for the definition of queries for the full-text search. The field list of an FTS document defines the schema for the SQLite virtual table.

---

<b>Note</b>	More fields for other attributes may be added at any time in order to extend the full-text search.
-------------	--

---

For more information on virtual tables for full-text search, refer to *NDS – Compiler Interoperability Specification, 16.3 SQLite fts3 Index* on page 216.

### Full-text Search Documents for POI Features

POI-relevant information, such as the number of stars, categories or special dishes for restaurant POIs, cannot be expressed in rigid, hierarchical structures. Therefore, full-text search is especially suitable for this kind of information.

Table 19-1 shows the columns of an FTS document for a POI, and at the same time the schema of the virtual `PoiFtsTable`.

*Table 19-1 Columns of the full-text search virtual table for POI features*

<b>Column</b>	<b>Description</b>
<code>docId</code>	Full-text search document ID The FTS document ID uniquely identifies a POI FTS document. The value must be set to the POI ID.
<code>poiName</code>	Full name of the POI
<code>poiCategory</code>	IDs of the POI categories
<code>poiStreetName</code>	Street name of the POI address
<code>poiHouseNumber</code>	House number of the POI address
<code>poiPostalCode</code>	Postal code of the POI address
<code>poiCityName</code>	Name of the city in which the POI is located
<code>stateName</code>	Name of the state in which the POI is located
<code>countryName</code>	Name of the country in which the POI is located
<code>suburbName</code>	Name of the suburb in which the POI is located
<code>poiPhone</code>	International phone number for the POI
<code>poiEmail</code>	E-mail address of the POI
<code>poiUrl</code>	URL of the POI's website
<code>searchTags</code>	Space separated list of optional search tags. The content of these hidden tags is not standardized. The following list gives examples: <ul style="list-style-type: none"><li>– Street base name</li><li>– City exonyms</li><li>– City district</li><li>– Known-as area</li><li>– POI descriptions</li><li>– IDs of the standard NDS database, for example, named object ID, route link ID</li></ul>

The fields of the `PoiFtsTable` are populated with data from the source building block. If no data is available for a field, the value for the corresponding field is set to an empty string. The POI ID and the name of the POI are mandatory data in the POI building block and, therefore, the `docId` and `poiName` fields are always filled. This guarantees that POIs can at least be found by their names.

### Full-text Search Documents for Named Object Features

Full-text based search can also be used to find addresses via road or crossroad named object features of the Name building block.

Table 19-2 shows the columns of an FTS document for a road named object feature, and at the same time the schema of the virtual `RoadFtsTable`.

Table 19-2 Columns of the full-text search virtual table for road named objects

Column	Description
docId	Full-text search document ID The FTS document ID uniquely identifies an FTS document for a road named object. The value must be set to the named object ID of the road named object feature.
roadName	Full name of the road named object
cityName	Name of the city that the road belongs to (represented by a CONTAINED_IN relation between road named object and city named object)
stateName	Name of the state that the road belongs to
countryName	Name of the country that the road belongs to
postalCode	Postal code of the road
suburbName	Name of the suburb that the road belongs to
searchTags	Space-separated list of optional search tags. The content of these hidden tags is not standardized. The following list gives examples: <ul style="list-style-type: none"> <li>- street base name</li> <li>- alternative names of the road named object</li> <li>- city exonyms</li> <li>- city district</li> <li>- known-as areas</li> </ul>

Table 19-3 shows the columns of an FTS document for a crossroad named object feature, and at the same time the schema of the virtual `crossroadFtsTable`.

Table 19-3 Columns of the full-text search virtual table for crossroad named objects

Column	Description
docId	Full-text search document ID The FTS document ID uniquely identifies a FTS document of a crossroad named object. The value must be set to the named object ID of the crossroad named object feature.
street1Name	Full name of the first street belonging to the crossroad named object
street2Name	Full name of the second street belonging to the crossroad named object
cityName	Name of the city that the crossroad belongs to (represented by a CONTAINED_IN relation between crossroad named object and city named object)
stateName	Name of the country that the crossroad belongs to
countryName	Name of the country that the crossroad belongs to
postalCode	Postal code of the crossroad
suburbName	Name of the suburb that the crossroad belongs to

Column	Description
searchTags	Space-separated list of optional search tags. The content of these hidden tags is not standardized. The following list gives examples: <ul style="list-style-type: none"> <li>- street base name</li> <li>- alternative names</li> <li>- known-as areas</li> </ul>

## Fuzzy Search Documents with Double Metaphones

*Double Metaphones* is a search algorithm based on phonetic algorithms, which extends the FTS building block with fuzzy search capabilities. The Double Metaphones algorithm is an extension of the phonetic search algorithms Soundex and Metaphones. Phonetic algorithms encode words with a similar pronunciation by the same representation.

The basic concept of Double Metaphones is to create one or two keys that represent the pronunciation of a string. The first key in the Double Metaphones is called *primary key* and represents the American pronunciation conventions, while the *secondary key* or alternate key represents the native pronunciation of a foreign word. In this way the Double Metaphone algorithm tries to account for myriad irregularities in English of Slavic, Germanic, Celtic, Greek, French, Italian, Spanish, Chinese, and other origins. For more information, see also [http://en.wikipedia.org/wiki/Double\\_Metaphone](http://en.wikipedia.org/wiki/Double_Metaphone) and <http://www.ddj.com/dept/cpp/184401251>.

Double Metaphones may be integrated in the Full-text Search building block in additional full-text search tables (DataScript location `nds.fts.name > RoadFtsDoubleMetaphTable` and `crossroadFtsDoubleMetaphTable`, as well as `nds.fts.poi > PoiFtsDoubleMetaphTable`). In order to store Double Metaphones, the same virtual table structures as for POIs or named objects are used. However, instead of the POI, road or crossroad names, these virtual tables contain two metaphone keys (primary and secondary key) separated by spaces in the corresponding columns. Also, there are no columns for postal codes, URLs, or house numbers, as Double Metaphones are not used for numbers. For detailed information on the tables containing Double Metaphones, refer to the *NDS – Physical Model Description*.

A double metaphone key character mostly is a consonant; vowels are largely eliminated, see Table 19-4.

Table 19-4 Example of the double metaphone keys for the name „Villingen“

Search string	Primary key	Secondary key
VILLINGEN	FLNIN	FLNKN
FILLINGEN	FLNIN	FLNKN
PHILLINGEN	FLNIN	FLNKN
WILLINGEN	ALNIN	FLNKN

To compare two words for a phonetic match, the application compares the primary and secondary keys of the first word with the primary and secondary keys of the second (see Table 19-5).

Table 19-5 Matching scheme for primary and secondary keys

Match between keys of the two words	Rank of match
Primary key = primary key	Strongest match
Secondary key = primary key	Normal match
Primary key = secondary key	Normal match
Secondary key = secondary key	Minimal match

## 19.3 Use of the Full-text Search Building Block

An application can use the Full-text Search building block by providing a search string as input for the FTS engine. The search string must comprise at least the following elements, separated by a space:

- Terms (tokens). Two types of terms exist:
  - *Unqualified terms*, that is, terms that can be applied to all fields of an FTS document. Example: Restaurant
  - *Qualified terms*, that is, terms that can only be applied to a specific field. Example: poiCategory:32
- The search terms can contain a wildcard character (\*) at the end of the term, representing zero or more characters.
- Phrases, that is, a group of terms enclosed in quotes
- Logical operators, such as AND, OR, NOT

The search string instructs the FTS engine to search for the following:

- Exact match for given fields
- Partial match, that is, specific words contained within given fields
- Prefix search for a word in given fields
- Combination of words, phrases, and prefixes, for a set of fields

Based on the search string, the FTS engine returns a result set with the IDs of the FTS documents matching the search string. The application can then use the resulting list of FTS document IDs to retrieve the FTS documents. The application can apply further filtering or can load the corresponding NDS features from the source building block.

### Use Cases for Full-text Search

Based on the structure and content of the Full-text Search building block as described in Section 19.1 *Building Block Structure and Content* on page 346 and Section 19.3 *Use of the Full-text Search Building Block* on page 352, the following use cases are possible:

- Search for a POI with the given full name
- Search for POIs containing the given word in the POI name
- Search for POIs of the given POI category and the given words in the name

- Search for POIs of the given POI category in a given city
- Search for POIs at a given address in a given city
- Search for a road in a given city
- Search for a crossroad using the base names of the crossing roads

The list is not complete, many more combinations are possible.



## 20 Icons

Icons are used in NDS as a graphical representation of points in a map, for example POIs, or for depicting an item classification in a list, for example the POI category selection list.

In NDS, icons are optional and can be provided for several purposes, such as:

- Display of road numbers
- POIs
- Named object classes
- Selection criteria
- City centers
- Point features
- TMC events (TMC icon table)
- Landmark icons (see Section 17.9 *Landmark Icons* on page 340)
- Additional icons for bannered routes

Icons shall be stored as PNG bitmaps and must be provided in several resolutions. They may additionally be stored as scalable vector data in SVG format. The following information is assigned to each icon:

- Scale range: Defines the recommended scale for displaying the icon
- Usage type flags (DataScript location: `nds.overall.icondata.IconTable`): Define the specific situations in which an icon shall be displayed. For POI categories, for example, different icons may be displayed for location input and map display. The following flags are defined:
  - 2D (2D icon, typically an ideographic representation of a category, brand or other classification)
  - 3D (3D icon, typically used for special icons of a POI, which can also be available as a 3D landmark, for example the dome in Cologne)
  - Night view
  - Day view
  - Highlighted version
  - Map display
  - Result-list display
  - On route/off route (specifies whether an icon is located on the guided route (on route), or not (off route))

Icons in PNG or SVG formats are stored as BLOBs in one of the instances of the `IconTable` (DataScript location: `nds.overall.icondata > IconTable`). The POI icons, for example, are stored in the `poiIconTable` (DataScript location: `nds.all > poiIconTable`), see also Section 13.5 *Representing POIs as Icons* on page 270.

The icon tables contain icon-specific information, such as:

- Icon name
- Icon size
- Icon format (for example, PNG, SVG, SPRITE, NDS\_3D)
- Icon data, defining specific icon data depending on the format, such as `IIcon3D` or `IIconSpriteRef`; the default value is `image`.
- Optional icon information in the data structure `AdditionalIconAttributes`, for example text bounding box, font for road number icons, etc.

## 20.1 Icon Sets

Similar icons are grouped into icon sets. For a POI category, for example, icons may be available for different scales and use types. These icons are grouped into one icon set, which is assigned to the corresponding POI category. Icon sets are defined by the `iconSetId` and stored in the `IconTable`).

Icon sets are used by different building blocks:

- Name building block: Icons for selection criteria, location input, names, and named object classes
- Basic Map Display building block: Icons for road numbers, selection criteria, and city centers (see Section 11.6 *Icons* on page 213)
- POI building block: Icons for POI categories, POI attributes, and POIs
- Traffic Information building block: Icons for TMC events
- 3D Objects building block: Icons for 3D landmarks

The application can supersede the icons stored in the NDS database and display customized icons instead.

## 20.2 Icon Collections

An icon collection collects all icons of an NDS database sharing the same design, for example, all icons of the same brand. The icon collections that are available in an NDS database are stored in the `IconCollectionTable` (DataScript location: `nds.overall.icondata`) with the following information:

- Collection ID: The collection ID uniquely identifies an icon collection. The default icon collection has ID 0. The collection ID is referenced in the `IconTable`.
- Description of the icon collection: Short description of the purpose of the collection, for example, an icon collection for a specific car brand which shall be used in navigation databases for the respective car manufacturer.

## 20.3 Default Icons and Application-specific Icons

The application is responsible to provide an icon for those features for which no custom icon is available in the NDS database. Also, if specific icons shall be used instead of icons provided by the NDS database, the application needs to provide these specific icons. This is necessary if, for example, the icons shall be harmonized within the entire HMI application.

If the application cannot identify a type defined in the NDS database, for example if a POI category has no standard category or is not connected to any standard category by means of the category graph, the application shall use a default icon to display the type.

## 20.4 Icon Sprites

Icon sprites provide the possibility to assemble icons in one set, thus enabling the application to load them into video RAM at the same time.

Icon sprites are modeled in NDS as follows (DataScript location: `nds.overall.icondata`):

- The icon format type **SPRITE** indicates that several icons are assembled in one set.
- The icon sprite reference (**iconSpriteRef**) contains the ID of the icon assemblage in the icon sprite table as well as position information (x and y coordinate values).
- The icon sprite table (**IconSpriteTable**) stores the assembled icons in one big set.

---

**Note** The icon format of the icons stored in the icon sprite table should be PNG (**iconFormat** type = 0).

---

## 20.5 Additional Icons for Banned Routes

Special routes form a loop or a spur of a more dominant route (so-called parent or mainline route). These special routes, also called child or auxiliary routes, are distinguished from the parent route by means of auxiliary words or suffix letters on the route shield or an adjacent sign, which are called „banner“. Some examples of banned routes: Bypass route, Truck route, Spur route, Business route, City route.

The icons indicating banned routes are modeled in NDS as follows:

The **AdditionalIconRefTable** (DataScript location: `nds.overall.metadata`) contains the additional icon types (**addIconType**) that are required to indicate the banned road type per region (**regionId**) and administrative road class (**adminRoadClass**).

The **numberPrefixId** in the road number data (`nds.name.namestring > RoadNumber`) refers to an ID in the **RoadNumberClassPrefixTable** where the administrative road class and region ID for the road number can be retrieved. The values from the **addIconType** array are then used together with this information to find the additional icon in the **AdditionalIconRefTable**.

The following additional icon types are defined:

- RENDER (text string defined in additionalString is rendered)
- NORTH
- SOUTH
- EAST
- WEST
- ALT (used for alternate routes)
- BYPASS
- BUSINESS
- BUSINESS\_LOOP
- BUSINESS\_ROUTE
- CITY
- CONNECTOR
- LINK
- LOOP
- SPUR
- TOLL
- SCENIC
- TRUCK
- TEMP

## 20.6 3D Icons

NDS provides data structures for rendering 3D icons (3D objects or animated icons) into a 2D map. The application can rotate these icons on the map, if required.

The choice NDS\_3D in `nds.overall.icondata > IconTable > IconData` indicates the availability of a 3D icon. 3D icons have the same structure as 3D objects (see Chapter 17 *3D Objects Building Block* on page 309). The `Icon3d` structure contains the geometry information for the 3D icon and the information whether the icon shall be rotated with the map. To always display a train icon parallel to its rail tracks, for example, `rotate` is set to TRUE.

## A      Glossary

### **Address retrieval**

Component in a navigation system that collects address-relevant information like city, zip code, road, house number or country for routing or POI features.

### **Advanced Map Display**

Data and functions for advanced display features, such as 3D objects or terrain models. NDS provides separate building blocks for this type of data:

- Orthoimages
- Digital terrain model
- 3D objects

### **Aggregation feature**

Real-world objects that share a name in the same language can be represented in the database by an aggregation feature. An aggregation feature is an umbrella object representing all named object features that have the same name.

### **Application**

A program or a series of programs that uses a database for navigation, map display or other map-based services. Typical applications in the context of Navigation Data Standard are in-car or hand-held navigation systems.

### **Area**

A two-dimensional object like a lake or woodland, which is represented in the database by an area feature in the Basic Map Display building block. Area features can have complex shapes, which can be approximated by a polygon.

### **Base link**

Road segment between two intersections, represented in the database by the link feature in the Routing building block.

A base link is a road segment that is located on one tile only and exists only on level 13 of the Routing building block. The road geometry of a base link is defined by a list of shape points assigned to the base link as an attribute.

See also Route link.

### **Basic Map Display building block**

The Basic Map Display building block groups together the features for map display applications. It contains the map features necessary for rendering two-dimensional maps together with their geometrical data.

**Binding**

Tie between certain values that belong together.

**Building block**

Main component of a database that complies with Navigation Data Standard. Each database is composed of several building blocks addressing specific functional aspects of the standard. There are separate building blocks, for example, for routing, map display and names.

A building block is either mandatory – like the Routing building block and the Name building block – or optional like the Basic Map Display building block which is necessary only for applications that support map display.

**Character code chart**

See <http://unicode.org/charts>.

**Cipher key**

Sequence of symbols that a user of a given cipher system must possess in order to use the system. Without a key, a user cannot encrypt data or decrypt data.

**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.

**Compiler**

A program or a series of programs that creates a database which complies with Navigation Data Standard.

**Contours**

Lines of equal elevation in a terrain. Contours can be used for map shading. In Navigation Data Standard, they are modeled as a line or an area feature in the Basic Map Display building block.

**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.

**Database**

Collection of navigation data that a navigation system operates on at a given time. The content and structure of a database is standardized by the current specification with the aim that an application is able to work with any database that complies with Navigation Data Standard.

**Destination location**

Linear or point location specifying a specific location on the map as a destination, for example, for routing.

**EGM96**

Earth Gravitational Model from 1996, used in NDS as standard reference system for representing absolute heights. EGM96 specifies the height above a so-called geoid which corresponds to the height above sea level. See also <http://en.wikipedia.org/wiki/EGM96>.

**Exonym**

Alternate name for a place, for example, for a city or a country, that is not used by the local inhabitants, neither in an official nor in any local language.

**Expression**

Combination of values, variables, operators, and functions that are evaluated according to specific rules in a particular programming language, which computes and then returns another value.

**External POI**

POIs that are not part of NDS and not delivered on the same data carrier.

**Feature**

Database representation of a real-world object relevant for a navigation system. All real-world objects can be represented by either one or more features in one of the levels of a building block.

A feature is the smallest entity within a database that can be addressed independently.

**Feature class**

Database classification for real-world objects. All real-world objects represented in one of the building blocks are instantiated in a feature class. The class defines the structures that may be used for describing the features in the class, that is attributes to define the specifics of different features in one class as well as references to other features.

**Filter attribute**

Attributes assigned to POIs that can be used for searching a POI.

**GDF**

Geographic data files – interchange file format for geographic data. Defined in ISO14825:2011. See also [http://en.wikipedia.org/wiki/Geographic\\_Data\\_Files](http://en.wikipedia.org/wiki/Geographic_Data_Files).

**Geocoding**

Process of assigning geographic identifiers like latitude and longitude to a feature, for example, to a house number that is part of an address.

Reverse geocoding is the inverse process: the feature is given and the geographic identifier is retrieved.

**Integrated POI building block**

POI building block that forms an integral part of an NDS database and contains references to other building blocks of the NDS database.

**Intersection**

Point where two or more roads meet or cross or a road ends. An intersection is represented in the database by a feature in the Routing building block.

**Level**

Partitioning of data in a building block. Depending on its type, a building block can have up to 16 levels and can contain replicated data in multiple levels. Higher levels are represented by lower level numbers and correspond to larger map scales. Lower levels are represented by higher level numbers and contain more detailed information.

The concept of levels can be employed, for instance, to quickly calculate long-distance routes and for a fast display of small-scale maps.

**Line**

A one-dimensional object like a river or a railway line, which is represented in the database by a line feature in the Basic Map Display building block.

**Link**

Road segment between two intersections, represented in the database by a feature in the Routing building block. In the real world, a link represents a road or a carriageway.

**Location**

Place connected to or part of the road network.

**Location reference**

Label assigned to a location. With a single location referencing method, one reference must define unambiguously exactly one location in the location referencing system. The reference is the string of data which is passed between different implementations of a location referencing system to identify the location.

**Map matching**

Component in a navigation system that uses GPS or other sensors to acquire position data for locating the user's position within the road network on the basis of the digital map represented in the database.

**Map scale**

Ratio between the displayed length of a map feature and the actual length of the corresponding real-world objects. A map scale of 1:10 000 indicates that 1 cm on the map corresponds to 100 m in the real world.

A map scale is called larger than another map scale when objects appear larger on the given map, for example, 1:10 000 is larger than 1:500 000.

**Metadata**

Metadata is data about data. It refers to information that is used to describe specific content.

In Navigation Data Standard, metadata can either refer to the data of a specific building block, of several building blocks or of all building blocks. Applications in the context of Navigation Data Standard use metadata to adapt to variable database content. The definition of metadata supports the independence of application software from navigational data, thus creating flexibility between systems.

**Name building block**

The Name building block groups together the naming data for other building blocks, for example, for the Routing and the Basic Map Display building blocks, and defines index structures and the input paths available for location input.

The Name building block must exist in every database that complies with Navigation Data Standard.

**Name features**

Two types of name features are available: Named objects and name strings.

**Name rotation**

Compound names can be stored using rotation markers. By this the input of name variants is supported. Example: "Bosch-Road, Robert-" (rotated name) and "Robert-Bosch-Road".

See also Rotation marker.

**Name string feature**

Verbal description of a real-world object represented by a character string or a number or a mixture of both. It defines exactly one name for one or several Named object feature.

Name string features are defined in the Name building block.

See also Named object feature.

**Named object feature**

Object representing a real-world object, for example, a country, a city, a road, a house, or a lake.

Named object features are defined in the Name building block.

**Navigation Data Standard (NDS)**

A standardized physical format for navigation systems jointly developed by NDS e.V.

**Navigation system**

Technical system which has the capabilities of supporting human beings in finding their way from one location to another.

**NDS e.V.**

A registered society of car manufacturers, system suppliers and map suppliers aimed at jointly developing a standardized physical format for navigation systems, the Navigation Data Standard.

**Next-valid-character tree**

Data structure of the Name building block that facilitates fast selection of names from large lists during location input.

An NVC tree is referred to by one or more named object features. It is made up of nodes and edges. The nodes represent characters that can be entered. For a given node, the edges lead to the node's child nodes. The child nodes represent possible continuations after selecting previous characters.

**Non-integrated POI building block**

POI building block that does not contain references to other building blocks of the NDS database.

**POI**

Point of interest. Service with a defined location.

**POI building block**

The POI building block groups together the features for Points of Interests, such as restaurants, sights, or car dealers.

The POI building block is stored in its own database. There are two possible types of the POI building block: integrated and non-integrated.

**POI category**

Classification of POIs according to their type. There are two types of POI categories:

- Super-categories: Summarize POI categories under a generic term
- Leaf categories: Identify POI categories that do not have any subsidiary categories

Example: The POI super-category „vehicle“ contains all POI categories thematically related to vehicles, such as gas stations, car dealers, car repair, car wash.

**Point**

A zero-dimensional object like a mountain peak or a city center, which is represented in the database by a point feature in the Basic Map Display building block.

**Polygon**

A shape in the plane defined by a finite number of points joined by straight lines. Depending on the context, the term is used to designate a curve or the area enclosed by a curve.

In Navigation Data Standard, a polygon is defined as a bounded and closed two-dimensional geometrical shape bounded by a finite number of straight lines. To represent an area, for example, polygons also form disconnected shapes, shapes with holes or shapes touching themselves. To avoid these cases that are problematic for drawing and filling algorithms, a polygon is often represented as a union of several simple elementary polygons which are connected and have no self-intersections or holes.

**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.

**Proximity search**

Search of an address within a given distance or area.

**Road geometry line**

Means for describing the geometry information of a road in Navigation Data Standard. Geometry information is needed, for example, for map matching, route guidance, or map display.

In the database, a road geometry line is represented by a feature in the Routing building block. The feature has all the attributes necessary for the graphical representation of roads.

**Rotation marker**

Special character that can be used to describe possible rotations of compound names.

See also Name rotation.

**Route calculation**

Component in a navigation system that finds the best route from a start position to a destination according to the user's preferences, traffic conditions, and so on.

**Route guidance**

Component in a navigation system that guides the user from the start position of a route to its destination. Route guidance always compares the current vehicle position with the route computed by route calculation and passes on corresponding maneuver descriptions to the user.

**Route link**

Road segment between two intersections, represented in the database by a link feature in the Routing building block.

A route link is a purely topological routing feature. It can exist on all levels of the Routing building block. On level 13, route links are used exclusively for links that extend over more than one tile as opposed to base links that are located on one tile only. Road geometry of route links is implemented via road geometry line features.

See also base link and road geometry line.

**Routing building block**

Building block that groups together the features for routing applications. It must exist in every database that complies with Navigation Data Standard.

**Selection graph**

Directed acyclic graph defining the index structures and the input paths available for location input in a given database. Input paths describe in which sequence which type of locations, for example, countries, cities, roads, etc. can be selected during location input.

The selection graph is stored as metadata in the Name building block.

**South-west rule**

Navigation Data Standard defines a tile as follows: If  $(x_1, y_1)$  is the south-west corner of a tile and  $(x_2, y_2)$  is its north-east corner, then all points  $(x, y)$  with  $x_1 \leq x < x_2$  and  $y_1 \leq y < y_2$  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.

**Speller tree**

Synonym for next-valid-character tree.

**Tile**

Geographic partition of the earth; tiles form approximately rectangular territorial sections. The union of the maximum number of tiles available on each building block level covers the whole surface of the Earth. All tiles on one level are bounded by tile borders of constant latitude and constant longitude, so the size of the tiles on one level is always equal in terms of WGS 84 coordinate units (fixed tiling scheme).

In NDS, tiles represent a unit for storing data in the database. That means a tile contains the data of all the features located within the geographic borders defined by the tile. The features are encoded in DataScript in a compact way and stored in the database as one BLOB (binary large object).

**Tile element**

Any object stored in a tile, for example, feature, named object, names.

**Tiling scheme**

Means for geographically partitioning the earth with the purpose of identifying, grouping and locating features in building blocks with geographical references. NDS uses a fixed tiling scheme, meaning that all tiles on one level are bounded by tile borders of constant latitude and constant longitude, so the size of the tiles on one level is always equal in terms of WGS 84 coordinate units.

**Transcription**

Phonetic mapping of a name from one language to another.

See also Transliteration.

**Transition**

Concept in Navigation Data Standard used to model the way from one link to another through an intersection, thus describing the topological connection of links through intersections.

In the database, a transition is represented as a property of the intersection to which it belongs.

**Transliteration**

Mapping of a name into another writing system, for example, from the Latin to a Cyrillic alphabet. The main objective of transliteration is to be able to reconstruct the original text.

See also Transcription.

**Triangulation**

In Navigation Data Standard, polygons are segmented into triangles. This process is called triangulation. The resulting primitives can be effectively rendered by 3D engines.

**Unicode**

Industry standard used in Navigation Data Standard for consistently representing and manipulating text.

**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.

**UTF-8**

Variable-length character encoding for Unicode that is used in Navigation Data Standard.

**WGS 84 coordinate system**

Standard coordinate reference system used in Navigation Data Standard.

## B NDS Database Supplier IDs

The NDS database supplier ID is a unique identifier for a party that compiles and supplies data in NDS format (DataScript location: `nds.overall.version.NdsDatabaseSupplierTable`).

NDS e.V. defines and maintains these identifiers. Obsolete IDs will be retired and not reused.

Table B-1 shows the NDS database suppliers and their respective IDs.

Note	The NDS database supplier ID is mandatory for NDS databases.
------	--

*Table B-1 List of NDS database supplier IDs*

NDS Database Supplier	ID
Aisin AW	10
Alpine Electronics	11
BMW	12
Clarion	13
Daimler	14
Denso	15
Elektrobit	16
Harman	17
Mapscape	18
Mitsubishi Electric	19
Navigon	20
NavInfo	21
NAVTEQ	22
Panasonic	23
PTV (retired)	24
Robert Bosch	25
RITU (retired)	26
TechniSat	27
TomTom	28
Volkswagen	29
Zenrin	30
AutoNavi	31
EMG	32
TeleNav	33



## C Overview of Database Tables

The following figures show the database tables for each building block in an NDS database.

<b>Note</b>	The database tables reflect DataScript version 2.1.1.8 and will be updated in the next release of the <i>NDS – Format Specification</i> .
-------------	---

Figure C-1 Product data

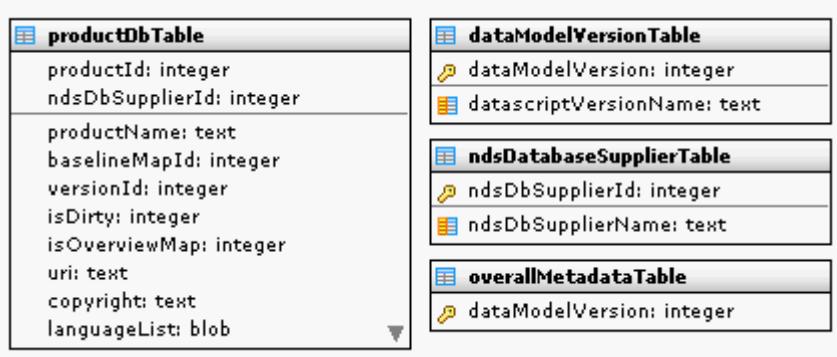


Figure C-2 Overall data

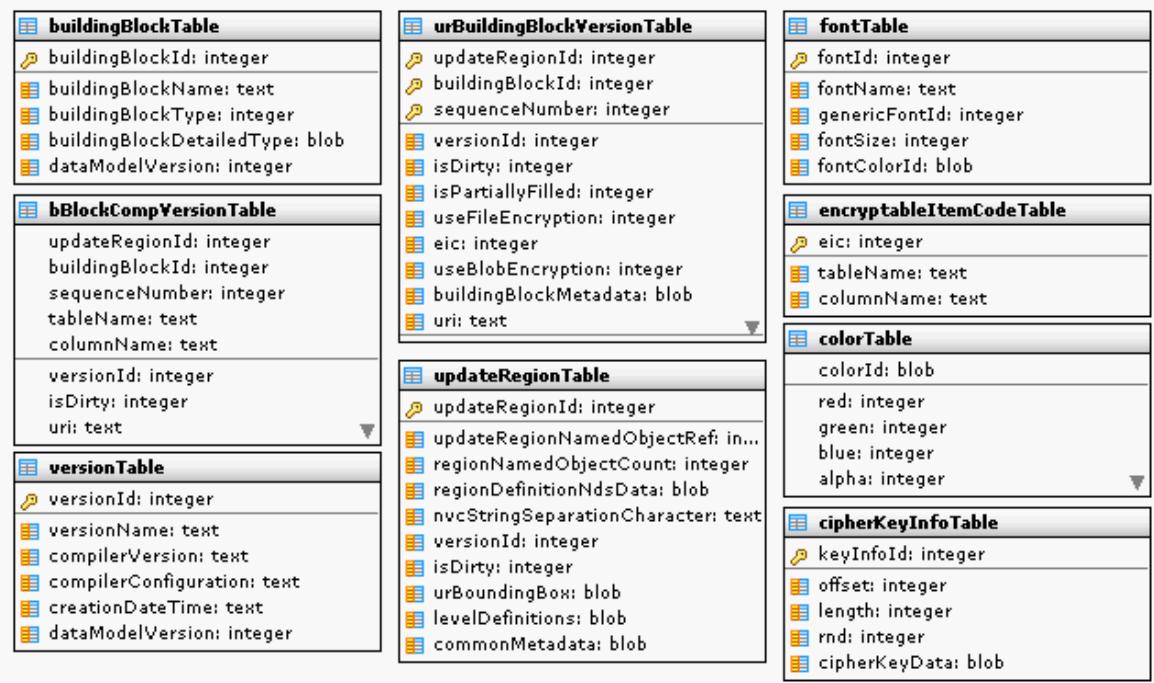


Figure C-3 3D Objects building block

<b>bodyGeometryBlobTable</b>	<b>spatialSubTreeTable</b>
blobId: integer	spatialSubTreeId: integer
bodyGeometryBlob: blob	spatialTreeBlob: blob
<b>object3DTextureMapTable</b>	<b>templateBodyGeometryTable</b>
textureMapId: integer	templateBodyGeometryId: integer
versionId: integer	templateBodyGeometry: blob
compressionType: integer	
cipherKeyInfoId: integer	
textureImageFormat: integer	
image: blob	
subImageArray: blob	
	<b>path3DTileTable</b>
	id: integer
	compressionType: integer
	cipherKeyInfoId: integer
	ndsData: blob

Figure C-4 Basic Map Display building block

<b>bmdIconCollectionTable</b>	<b>bmdTileTable</b>
collectionId: integer	id: integer
versionId: integer	versionId: integer
collectionName: text	compressionType: integer
	cipherKeyInfoId: integer
	ndsData: blob
	tilePatternId: integer
<b>bmdIconTable</b>	<b>bmdTilePatternTable</b>
iconId: integer	tilePatternId: integer
collectionId: integer	versionId: integer
iconSetId: integer	compressionType: integer
scaleLevelId: integer	cipherKeyInfoId: integer
usageTypeMask: integer	ndsData: blob
additionalAttributes: blob	
iconDisplayArrangement: integer	
iconDrawingPriority: integer	
versionId: integer	
compressionType: integer	
cipherKeyInfoId: integer	
iconName: text	
pixWidth: integer	
pixHeight: integer	
iconFormat: integer	
image: blob	
	<b>bmdScaleLevelTable</b>
	scaleLevelId: integer
	minScaleDenominator: integer
	maxScaleDenominator: integer
	<b>bmdFeatureClassHierarchyTable</b>
	parentClass: integer
	childClass: integer

Figure C-5 Digital Terrain Model building block

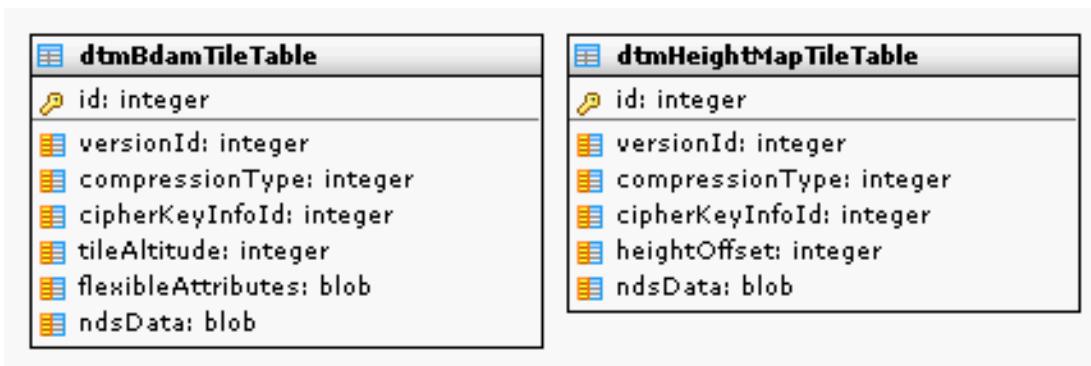


Figure C-6 Full-text Search building block

<b>poiFtsTable</b>	<b>roadFtsTable</b>	<b>crossroadFtsTable</b>
<p>docId: integer</p> <p>poiName: text</p> <p>poiCategory: text</p> <p>poiHouseNumber: text</p> <p>poiStreetName: text</p> <p>poiPostalCode: text</p> <p>poiCityName: text</p> <p>poiStateName: text</p> <p>poiCountryName: text</p> <p>poiSuburbName: text</p> <p>poiPhone: text</p> <p>poiEmail: text</p> <p>poiUrl: text</p> <p>searchTags: text</p> <p>mortonCode: text</p>	<p>docId: integer</p> <p>roadName: text</p> <p>cityName: text</p> <p>stateName: text</p> <p>countryName: text</p> <p>postalCode: text</p> <p>suburbName: text</p> <p>searchTags: text</p> <p>mortonCode: text</p>	<p>docId: integer</p> <p>street1Name: text</p> <p>street2Name: text</p> <p>cityName: text</p> <p>stateName: text</p> <p>countryName: text</p> <p>postalCode: text</p> <p>suburbName: text</p> <p>searchTags: text</p> <p>mortonCode: text</p>
<b>poiFtsDoubleMetaphTable</b>	<b>roadFtsDoubleMetaphTable</b>	<b>crossroadFtsDoubleMetaphTable</b>
<p>docId: integer</p> <p>poiNameDmKeys: text</p> <p>poiCategoryDmKeys: text</p> <p>poiStreetNameDmKeys: text</p> <p>poiCityNameDmKeys: text</p> <p>stateNameDmKeys: text</p> <p>countryNameDmKeys: text</p> <p>suburbNameDmKeys: text</p> <p>poiEmailDmKeys: text</p> <p>searchTagsDmKeys: text</p> <p>mortonCode: text</p>	<p>roadNameDmKeys: text</p> <p>cityNameDmKeys: text</p> <p>stateNameDmKeys: text</p> <p>countryNameDmKeys: text</p> <p>suburbNameDmKeys: text</p> <p>searchTagsDmKeys: text</p> <p>mortonCode: text</p>	<p>street1NameDmKeys: text</p> <p>street2NameDmKeys: text</p> <p>cityNameDmKeys: text</p> <p>stateNameDmKeys: text</p> <p>countryNameDmKeys: text</p> <p>suburbNameDmKeys: text</p> <p>searchTagsDmKeys: text</p> <p>mortonCode: text</p>
<b>ftsRoadNameMetadataTable</b>	<b>ftsCrossroadNameMetadataTable</b>	<b>ftsCrossroadNameColMetaTable</b>
<p>buildingBlockId: integer</p> <p>ftsTokenizer: integer</p> <p>searchTagsDescription: text</p>	<p>buildingBlockId: integer</p> <p>ftsTokenizer: integer</p> <p>searchTagsDescription: text</p>	<p>columnName: text</p> <p>dataAvailability: integer</p>
<b>ftsPoiMetadataTable</b>	<b>ftsRoadNameColMetaTable</b>	<b>ftsCrossroadNameColMetaTable</b>
<p>buildingBlockId: integer</p> <p>ftsTokenizer: integer</p> <p>searchTagsDescription: text</p>	<p>columnName: text</p> <p>dataAvailability: integer</p>	<p>columnName: text</p> <p>dataAvailability: integer</p>
<b>ftsPoiColMetaTable</b>		
<p>columnName: text</p> <p>dataAvailability: integer</p>		

Figure C-7 Junction View building block

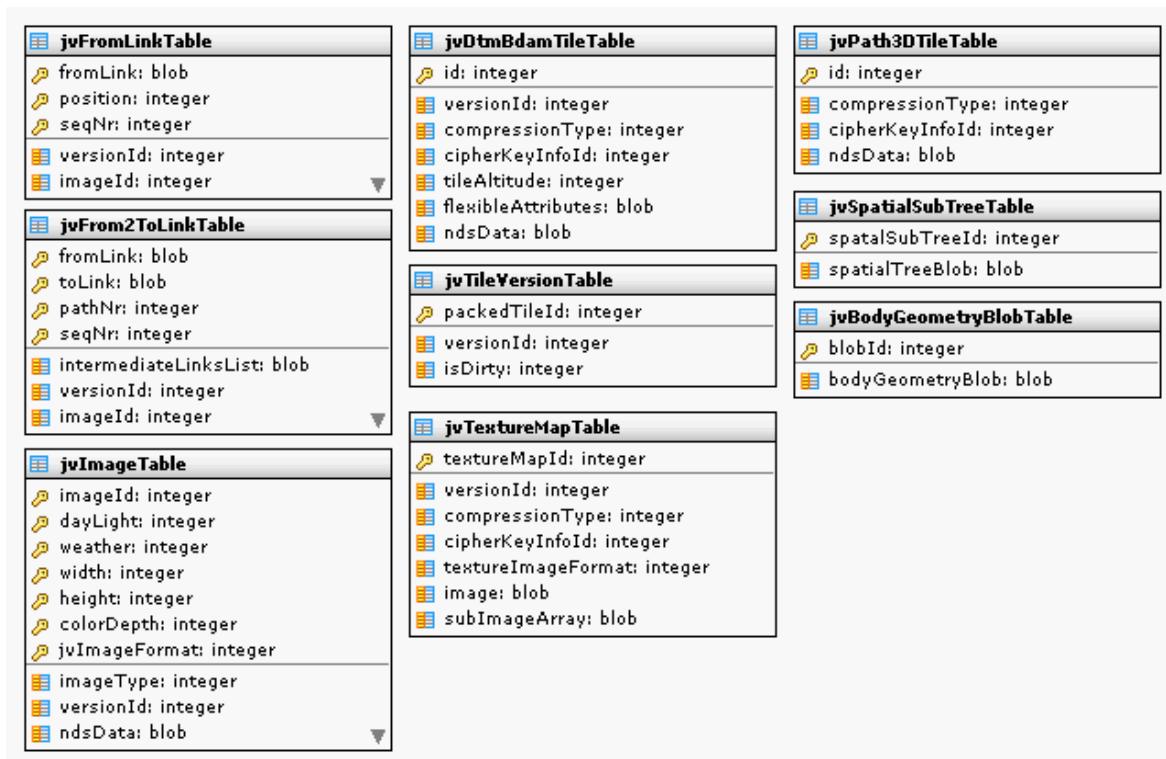


Figure C-8 Name building block

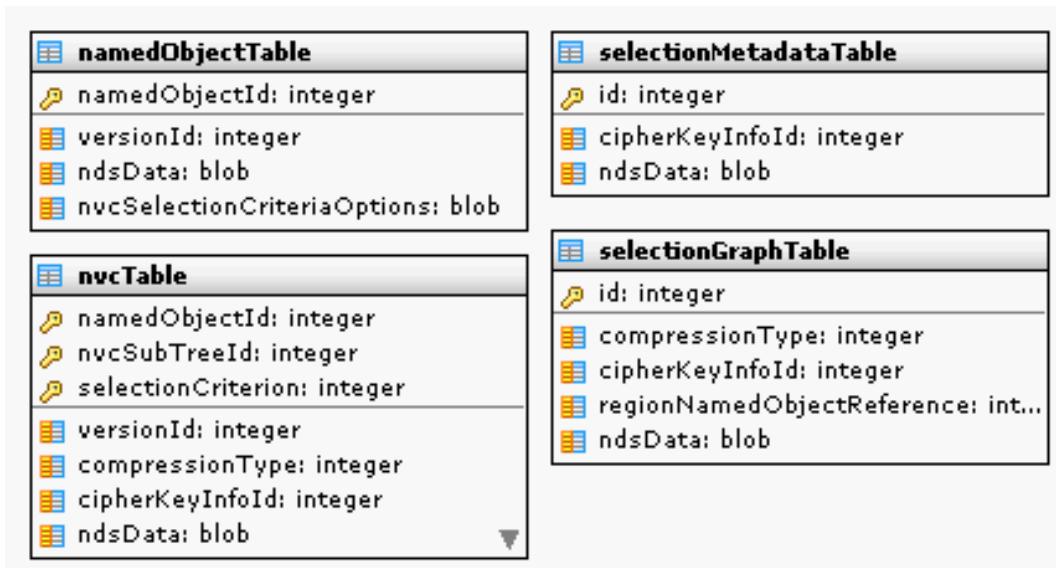


Figure C-9 Orthoimages building block

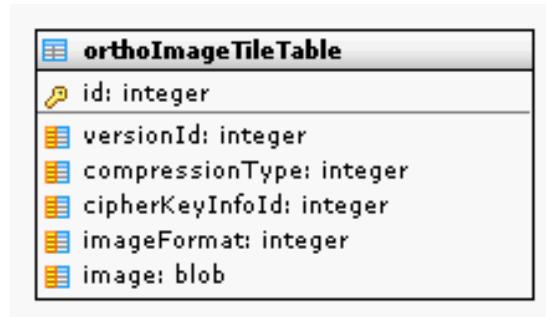


Figure C-10 POI building block

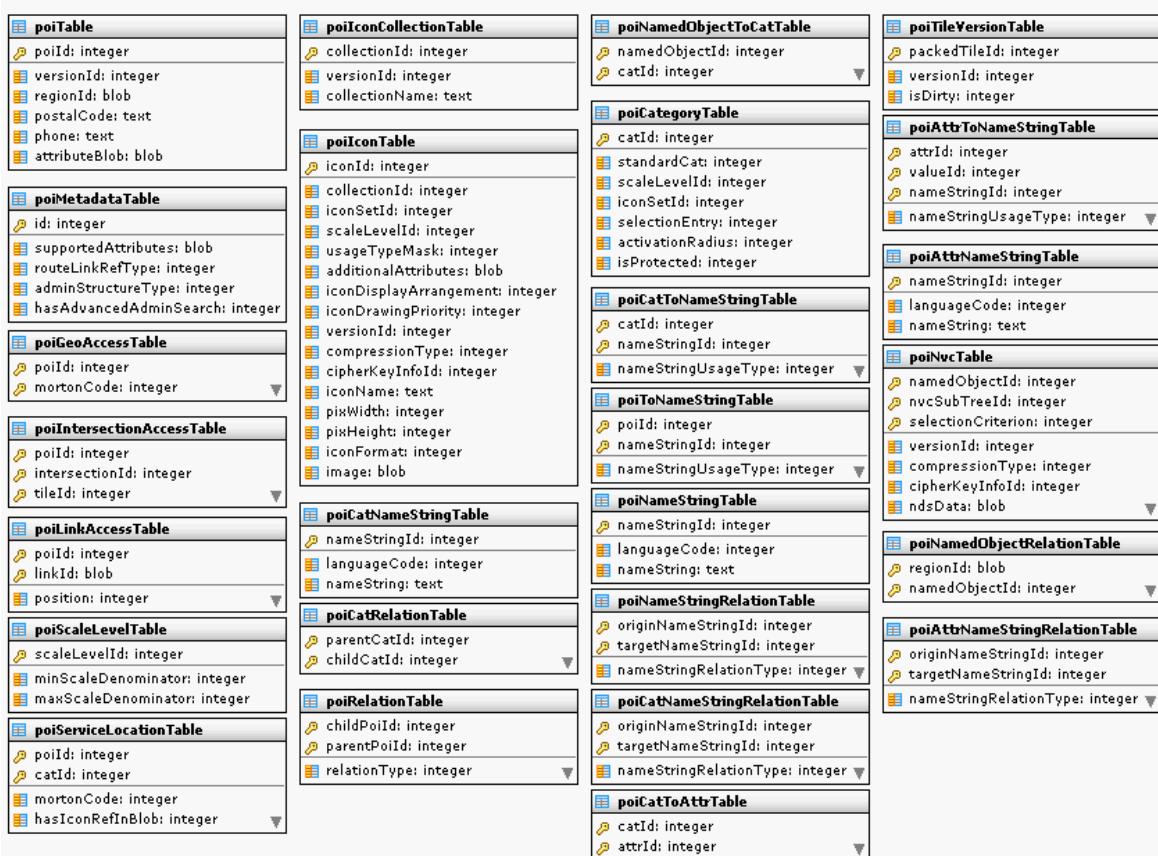


Figure C-11 Routing building block

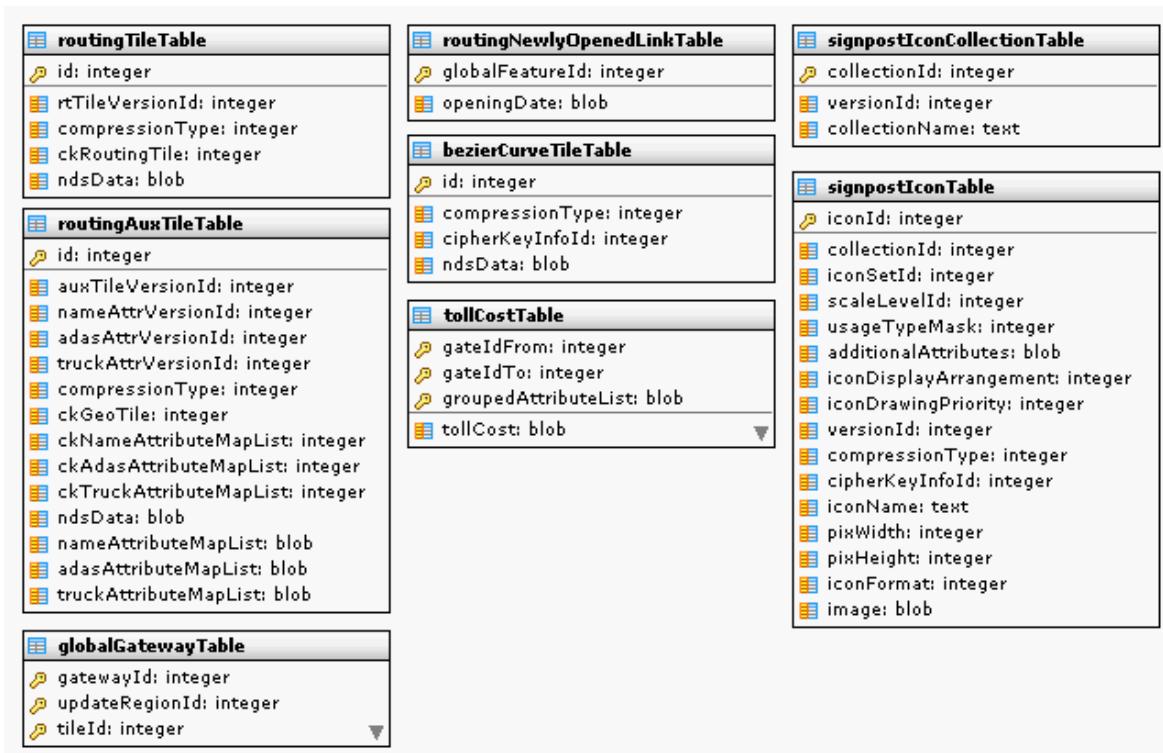
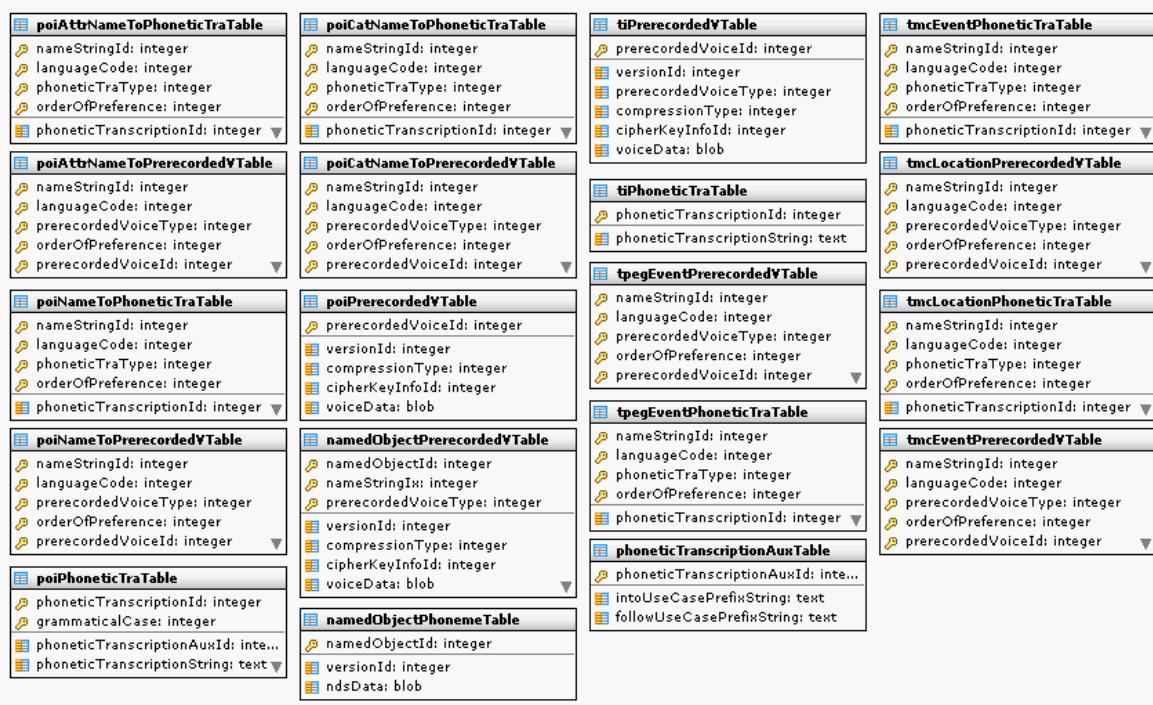
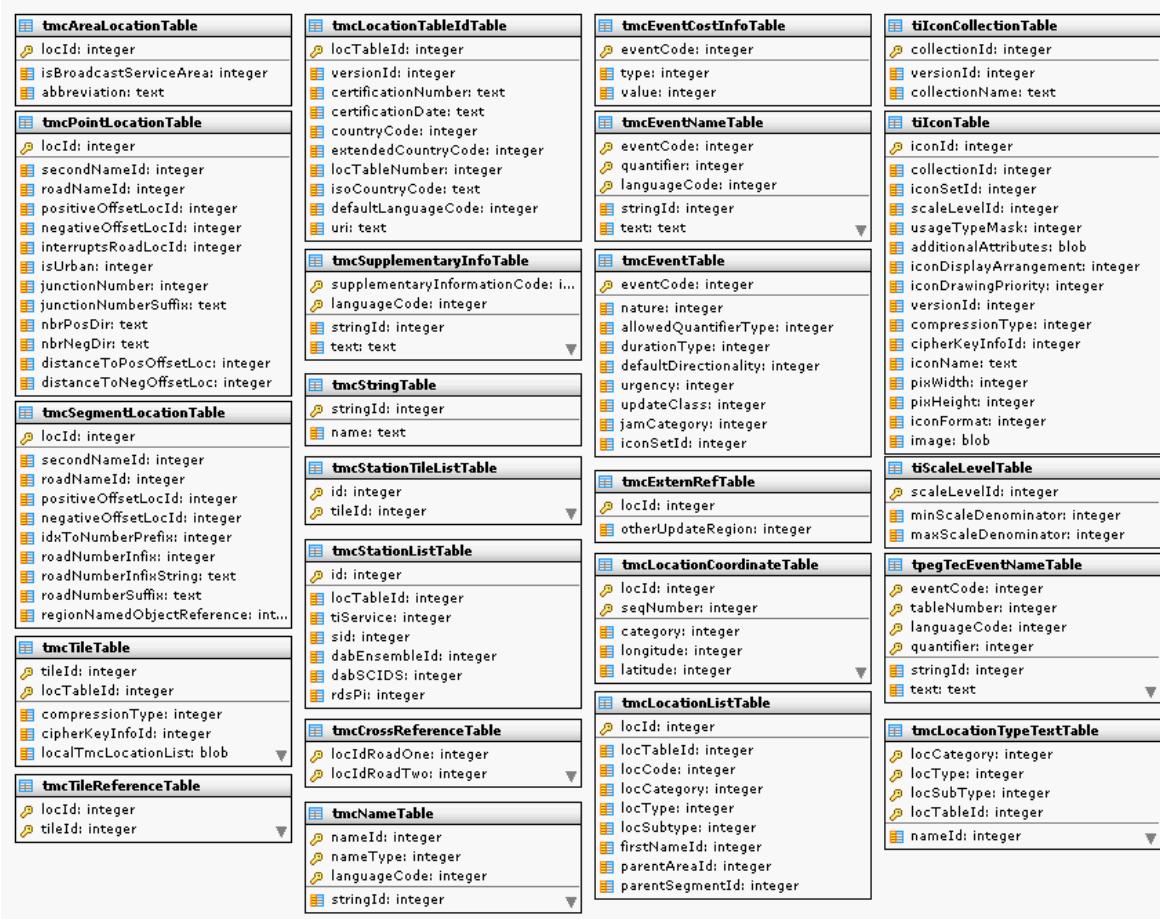


Figure C-12 Speech building block



*Figure C-13 Traffic Information building block*





## D List of Flexible Attributes

NDS provides many flexible attributes, which can be used to model complex real-world situations.

---

**Note**

The following overviews are work in progress and subject to change.

---

Figure D-1 Flexible attributes in NDS, part 1

19.12.2011, 21:08  
Page 1 of 8

## List of Flexible Attributes

Type Code	Attribute Name	Properties	Routing BB Features	Routing BB Layer	Routing or BMD Level	Name BB	BMD BB	Points	Map Lines	Areas	Region metadata	Spotted node 3D Object	Remarks
0 00	VALIDITY_RANGE	hexadecimal decimal bidirectional grouped only											bi-directional reference if used for both directions
1 01	TIME RANGE OF DAY								x				time
2 02	DATE RANGE OF YEAR							x	x				time
3 03	DAYS OF WEEK							x	x				time
4 04	DAY OF MONTH							x	x				time
5 05	DAY OF YEAR							x	x				time
6 06	MONTHS OF YEAR							x	x				time
7 07	TIME DURATION							x	x				time
8 08	TRANSITION MASK 2-4 VAL							x	x				
9 09	TRANSITION MASK 5-VAL							x	x				
10 10	TRANSITION MASK 6-8 VAL							x	x				
11 11	TRANSITION MASK 9-16 VAL							x	x				
12 12	OC TIME RANGE OF WEEK							x	x				
13 13	OD SPECIAL TRANSITION CODE							x	x				
14 14	OD SPECIAL TRANSITION CODE							x	x				
15 15	OD RIGHT OF WAY REGULATION							x	x				
16 16	TRAFFIC_LIGHTS							x	x				
17 17	SIGN POSITION							x	x				
18 18	CONSUMPTION UP EXCES SLOPE							x	x				
19 19	CONSUMPTION DOWN EXCESS SLOPE							x	x				
20 20	CONSUMPTION SPEED VARIATION							x	x				
21 21	CONSUMPTION SPEED DEPENDENCY							x	x				
22 22	AVERAGE SLOPE							x	x				
23 23	LINK PERCENTAGE							x	x				
24 24								x	x				
25 25								x	x				
26 26								x	x				
27 27								x	x				
28 28								x	x				
29 29								x	x				
30 30								x	x				
31 31								x	x				
32 32								x	x				
33 33								x	x				
34 34								x	x				
35 35								x	x				
36 36								x	x				
37 37								x	x				
38 38								x	x				
39 39								x	x				
40 40								x	x				
41 41								x	x				

*Figure D-2 Flexible attributes in NDS, part 2*

Type Code	Type	Attribute Name	Attribute Properties	Routing BB Features	Routing BB Layer	Routing or BMD Level	Name BB	BMD BB	Remarks
hexdema	hexdema	2A WEIGHT_PER_AXLE_IMPERIAL	x	x	x	x	x	x	physical conditions
		2B WEIGHT_PER_AXLE_METRIC	x	x	x	x	x	x	physical conditions
		2C VEHICLE_HEIGHT_IMPERIAL	x	x	x	x	x	x	physical conditions
		2D VEHICLE_HEIGHT_METRIC	x	x	x	x	x	x	physical conditions
		2E VEHICLE_WIDTH_IMPERIAL	x	x	x	x	x	x	physical conditions
		2F VEHICLE_WIDTH_METRIC	x	x	x	x	x	x	physical conditions
		30 LENGTH_IMPERIAL	x	x	x	x	x	x	physical conditions
		31 LENGTH_METRIC	x	x	x	x	x	x	physical conditions
		32 NUM_AXLES	x	x	x	x	x	x	physical conditions
		50	x	x	x	x	x	x	
		51	x	x	x	x	x	x	
		52	x	x	x	x	x	x	
		53	x	x	x	x	x	x	
		54	x	x	x	x	x	x	
		55	x	x	x	x	x	x	
		56	x	x	x	x	x	x	
		38 FREQUENTLY USED VEHICLE TYPES	x	x	x	x	x	x	
		57	x	x	x	x	x	x	
		58	x	x	x	x	x	x	
		59	x	x	x	x	x	x	
		60	x	x	x	x	x	x	
		61	x	x	x	x	x	x	
		62	x	x	x	x	x	x	
		63	x	x	x	x	x	x	
		64	x	x	x	x	x	x	
		40 SIGNPOST_TOWARD_INFO_REFERENCE	x	x	x	x	x	x	
		41 SIGNPOST_DIRECTIONAL_INFO_REFERENCE	x	x	x	x	x	x	
		42 SIGNPOST_ICON_REFERENCE	x	x	x	x	x	x	
		43 SIGNPOST_EXIT_NUMBER	x	x	x	x	x	x	
		44 SIGNPOST_LINK_REFERENCE	x	x	x	x	x	x	
		45	x	x	x	x	x	x	
		46	x	x	x	x	x	x	
		47	x	x	x	x	x	x	
		48	x	x	x	x	x	x	
		49	x	x	x	x	x	x	
		50	x	x	x	x	x	x	
		51	x	x	x	x	x	x	
		52	x	x	x	x	x	x	
		53	x	x	x	x	x	x	
		54	x	x	x	x	x	x	
		55	x	x	x	x	x	x	
		56	x	x	x	x	x	x	
		57	x	x	x	x	x	x	
		58	x	x	x	x	x	x	
		59	x	x	x	x	x	x	
		60	x	x	x	x	x	x	
		61	x	x	x	x	x	x	
		62	x	x	x	x	x	x	
		63	x	x	x	x	x	x	
		64	x	x	x	x	x	x	
		65	x	x	x	x	x	x	
		66	x	x	x	x	x	x	
		67	x	x	x	x	x	x	
		68	x	x	x	x	x	x	
		69	x	x	x	x	x	x	
		70	x	x	x	x	x	x	

Figure D-3 Flexible attributes in NDS, part 3

19.12.2011, 21:08  
Page 3 of 8

## List of Flexible Attributes

Type Code	Attribute Name	Attribute Properties	Routing BB Features	Routing BB Layer	Routing or BMD Level	Name BB	BMD BB	Region metadata	Spatter node 3D Object	Remarks
24	4A WIDE_LANE_MASK	hexadecimal decimal	75 4B TOURIST_ROUTE_TYPE	*	*	*	*	*	*	shall-only-be-used-in-LANE_SEPARATORS-between-
76	4C WIDE_SEPARATOR	*	*	*	*	*	*	*	*	shall-only-be-used-in-LANE_SEPARATORS-between-
77	4D HAS_JUNCTION_VIEW	*	*	*	*	*	*	*	*	to be grouped with validity range
78	4E NUM_TOLL_BOOTH	*	*	*	*	*	*	*	*	to be grouped with other conditions, e.g. toll parent type, provider, currency, change
79	4F TOLL_BOOTH_MASK	*	*	*	*	*	*	*	*	
80	50 TOLL_VIGNETTE	*	*	*	*	*	*	*	*	toll
81	51 TOLL_ENTRY	*	*	*	*	*	*	*	*	
82	52 TOLL_EXIT	*	*	*	*	*	*	*	*	
83	53 TOLL_CHARGE	*	*	*	*	*	*	*	*	
84	54 CURRENCY	*	*	*	*	*	*	*	*	
85	55 TOLL_GATE_ID	*	*	*	*	*	*	*	*	
86	56 TOLL_PAYMENT_PROVIDER_ID	*	*	*	*	*	*	*	*	
87	57 TOLL_PAYMENT_TYPE	*	*	*	*	*	*	*	*	
88	58 NUM_LANES	*	*	*	*	*	*	*	*	
89	59 LANE_CONNECTIVITY_ALONG_LINK	*	*	*	*	*	*	*	*	*VALIDITY_RANGE* specifies the location of the lane connection.
90	5A LANE_CONNECTIVITY_ACROSS_INTERSECTION	*	*	*	*	*	*	*	*	
91	5B LANE_MARKING	*	*	*	*	*	*	*	*	
92	5C LANE_SEPARATORS	*	*	*	*	*	*	*	*	
93	5D LANE_PASSING	*	*	*	*	*	*	*	*	
94	5E LANE_TURNING	*	*	*	*	*	*	*	*	
95	5F LANE_USED_IN_BOTH_DIRECTIONS	*	*	*	*	*	*	*	*	
96	60 PHYSICAL_WIDTH_METRIC	*	*	*	*	*	*	*	*	
97	61 CURVATURE	*	*	*	*	*	*	*	*	
98	62 TYPE_OF_PAVEMENT	*	*	*	*	*	*	*	*	Routing or guidance

Figure D-4 Flexible attributes in NDS, part 4

Type Code	Attribute Name	Properties	Routing BB Features	Routing BB layer	Routing BB layer	Name	BB	BB	BB	BB	BB	3D Object	Spacial node	3D Object	Remarks
hexadecimal decimal															what is the use case for this?
99	63 OVERPASS														Only L13 or super levels too?
100	64 UNDERPASS														what is the use case for this?
101	65 SLOPE_ARRAY														Only L13 or super levels too?
102	66 ELEVATION_DELTA														for countour lines and building heights
103	67 FERRY_TYPE														must be grouped at least with a LANE_MASK indicating which lane has the divider
104	68 PHYSICAL_WIDTH_IMPERIAL														
105	69 PHYSICAL_LANE_DIVIDER														
106	6A TOLL_PAYMENT_PROVIDER_DEFINITION	grouped only													
107	6B	bidirectional													
108	6C	bidirectional													
109	6D DIGITIZATION_STATUS	grouped only													
110	6E ATTRIBUTION_STATUS	bidirectional													
111	6F	bidirectional													
112	70														
113	71														
114	72 AVERAGE_SPEED														used for eco routing for grouping with vehicle types
115	73 URBAN														always grouped with LENGTH_ALONG_LINK
116	74 LINK_TYPE														always grouped with LENGTH_ALONG_LINK
117	75 TUNNEL														always grouped with LENGTH_ALONG_LINK
118	76 FERRY														always grouped with LENGTH_ALONG_LINK
119	77 TOLL														always grouped with LENGTH_ALONG_LINK
120	78 BRIDGE														always grouped with LENGTH_ALONG_LINK
121	79 NUM_TOLL_GATES														
122	7A NUM_TRAFFIC_LIGHTS														

Figure D-5 Flexible attributes in NDS, part 5

Type Code	Attribute Name	Properties	Routing BB Features	Routing BB Layer	Routing or BMD Level	Level 15	Upper Levels	Name BB	BMD BB	Region BB	Spatter node	3D Objec	Remarks
123	7B) NUM_GIVE_RIGHT_OF_WAY	hexadecimal					x						To be clarified if we need that
124	7C) LENGTH_ALONG_LINK						x						Metadata only
125	7D) ICON_REFERENCE						x						bind only
126	7E) ROTATED_ICON_REFERENCE						x						
127	7F												
128	80												
129	81												
130	82												
131	83												
132	84												
133	85) ROAD_Z_LEVEL						x						do we need an additional attribute layer on L13 of routing BB for attributes being more relevant only?
134	86) DRAWING_ORDER						x						do we need an additional attribute layer on L13 of routing BB for attributes being more relevant only?
135	87) DAYTIME_RUNNING_LIGHT						x						do we need an additional attribute layer on L13 of routing BB for attributes being more relevant only?
136	88						x						
137	89) HIERARCHY						x						
138	8A) HIERARCHY						x						
139	8B						x						
140	8C						x						
141	8D						x						
142	8E						x						
143	8F						x						
144	90						x						
145	91						x						
146	92) POSTAL_CODE_NAMED_OBJECT_RELATIVE_REF						x						
147	93) ADMIN_HIERARCHY_NAMED_OBJECT_RELATIVE_REF						x						
148	94) POSTAL_CODE_NAMED_OBJECT_REFERENCE						x						
149	95) ADMIN_HIERARCHY_NAMED_OBJECT_REFERENCE						x						
150	96) NAMED_OBJECT_REFERENCE						x						
151	97) NAMED_OBJECT_BASE_REF						x						
152	98) NAMED_OBJECT_RELATIVE_REF						x						
153	99) CONTINUED_TURN_RESTRICTION						x						
154	9A) TITLE_OF_CONTINUED_TURN_RESTRICTION						x						
155	9B) CONTINUED_TURN_RESTRICTION						x						
156	9C) NAMED_OBJECT_POSITION						x						
157	9D) FLOOR_NUMBER						x						
158	9E) NAMED_OBJECT_CLASS						x						
159	9F) EXTENDED_POST_CODE_POSITION						x						
160	A0) TOP_SELECTION_ENTRY						x						
161	A1) TOP_SELECTION_INDICATOR						x						

Figure D-6 Flexible attributes in NDS, part 6

Type Code	Attribute Name	Properties	Routing BB Features	Routing BB layer	Routing BB Level	Name BB	BMD BB	3D Object	Spacial node	Region metadata	Remarks
hexdezimal dezimal											
162 A2											
163 A3											
164 A4											
165 A5											
166 A6											
167 A7											
168 A8											
169 A9											
170 AA											
171 AB											
172 AC	POPULATION										
173 AD	GATEWAY										
174 AE	KM_MILL_INDICATOR										
175 AF	MULT_DIGITIZED										
176 B0	REGION_NAMED_OBJECT_REFERENCE	grouped only bidirectional unidirectional									
177 B1	NON_DEFAULT_TRAFFIC_SENSE										
178 B2	DETACHED_FROM_TERRAIN										
179 B3	STUBBLE										
180 B4	LOCAL_TMC_LOCATION										
181 B5	HOUSE_NUMBER_RANGE										
182 B6	PROFILE_BASE_AVG_SPEED										
183 B7	PROFILE_BASE_ELEVATED_ROAD										
184 B8	IS_ELEVATED_ROAD										
185 B9	CO_BUILDING_HEIGHT										
186 BA	C1_GROUND_HEIGHT										
187 BB	ATTRIBUTE_POINT_LIST										
188 BC	ODD_OR_EVEN_DAYS										
189 BD	WEEKDAY_IN_MONTH										
190 BE	TIME_ZONE_REFERENCE										
191 BF	RACE_TRACK										
192 CG	BUILDING_HEIGHT										
193 CI	GROUND_HEIGHT										
194 C2	BASE_ELEVATION										
195 C3	ROOF_COLOR										
196 C4	WALL_COLOR										
197 C5	WARNING_SIGN										
198 C6	UNDER_CONSTRUCTION										

Figure D-7 Flexible attributes in NDS, part 7

19.12.2011, 21:08  
Page 7 of 8

## List of Flexible Attributes

Type Code	Attribute Name	Properties	Routing BB Features	Routing BB Layer	Routing or BMD Level	Name BB	BMD BB	Region metadata	Spatter node	3D Objec	Remarks
199	C7 AREA_TYPE	hexadecimal			x				x		BMD and 3d Objects
200	C8 IS_COUNTRY_CAPITAL								x		BMD only
201	C9 RAILWAY CROSSING										
202	CA PEDESTRIAN CROSSING										
203	CB FEATURE_TYPE										for default mechanisms assigning to the whole file
204	CC1 WEATHER										
205	CD1 LEGAL_SPEED_LIMIT_RANGE										
206	CE SINGLE_OR_DUAL_LANE										
207	CF SEASONAL_CLOSED								x		
208	DG FERRY_TRAVEL_TIME								x		
209	DI TIME_DOMAIN								x		
209	DI CONCRETE_NAMED_OBJECT								x		
210	D2 TRAFFIC_ZONE								x		
211	D3 FREQUENT_ACCIDENTS								x		
212	D4 HAS_3D_OBJECT								x		
213	D5 NOT_USEFUL_FOR_BEGINNERS								x		
214	D6 GLOBAL_FEATURE_ID								x		
215	D7 FERRY_TRAVEL_TIME								x		
216	D8 SHORTCUT								x		
217	D9 BYPASS								x		
218	DA SIDE_STREET								x		
219	DB HAS_VCS								x		
220	DC EXTENDED_CLOTHOID_META_DATA								x		
221	DD EXTENDED_CLOTHOID								x		
222	DE EXTENDED_GRADIENT_META_DATA								x		
223	DF EXTENDED_GRADIENT								x		
224	E0 ENHANCED_GEOMETRY								x		
225	E1 ADRS_ACCURACY								x		
226	E2								x		
227	E3 START_OR_END_DESTINATION_ROAD_ONLY								x		
228	E4 BORDER_IS_DISPUTED_BY								x		
229	E5 BORDER_IS_ACCEPTED_BY								x		
230	E6								x		
231	E7								x		
232	E8								x		
233	E9								x		
234	EA								x		
235	EB								x		

*Figure D-8 Flexible attributes in NDS, part 8*



# E Index

- 0-9
- 2.5D city models 223
  - 3D landmarks for POI 258
  - 3D object
    - reference from a POI 258
  - 3D Object feature 312
    - properties 313
  - 3D objects
    - performance 309
    - replacing footprints 226, 336
  - 3D Objects building block 309
  - 3D Object feature 312
  - aggregated feature class for spatial node 320
  - applying textures 334
  - body geometry type 326
  - bounding box 315
  - color gradient 332
  - content 309
  - data 310
  - elevation 324
  - feature classes 320
  - feature properties 313
  - geometry information 322
  - identifying 3D objects 320
  - level of detail 325
  - material 331
  - normal in 3D space 323
  - object level hierarchy 319
  - reference to geometry
  - information 335
  - references 322
  - references to 3D objects 336
  - references to Name building block 336
  - render group 328
  - scaling factor for elevation 324
  - spatial extension of 3D object 315
  - spatial index structure 311
  - spatial node 312
  - structure 309
  - template body geometry 327
  - template buildings 326
  - textures 331, 333
  - triangle strip 326
  - vector in 3D space 323
- A
- access location for named objects 172
  - access relation between POIs 265
  - Access-at-road relation 174
  - ADAS
    - attribute 134
    - attribute layer 83
    - curvature profile 136
    - enhanced attributes for elevation profiles 143
    - enhanced geometry (clothoids) 137
    - enhanced geometry for intersections 145
    - slope profile 136
  - ADAS attributes 134
  - additional icons 78
    - data 78
  - address format 79
    - example (Germany) 79
  - address points 151, 199
  - address retrieval 163, 196
  - administrative road numbers 77
  - advanced driver assistance, see ADAS 134
  - advanced map display
    - 3D objects 309
  - aerial images (see Orthoimages) 305
  - AES-128 103
  - aggregation feature 171
  - ALERT-C standard 232
  - algorithm
    - encryption 103
    - Routing 108
  - anchor point of tiles 95
  - application
    - icons 357
  - Application Data Containers (ADC) 249

architecture of database 43  
architecture of NDS 43  
area feature 211  
    fixed attributes 212  
area features  
    reference from a POI 258  
    replace by 3D objects 226  
area location (TMC) 233  
articles for TTS 283, 286  
ASR 281  
attribute 49  
    ADAS 134  
    area feature 212  
    attribute group 81  
    attribute layer (definition) 82  
    condition attribute 82  
    currency 88  
    data quality 86  
    definition 52  
    detached from terrain 222  
    extension 83  
    grouping 81  
    intersections 127  
    line feature 210  
    link feature 117  
    name string feature 177  
    named object feature 171  
    POI 256  
    point feature 209  
    road geometry line feature 123  
    Routing 109  
    spatial extension 83  
    validity range 83  
attribute group (definition) 81  
attribute layer  
    ADAS 83  
    attribute point list 83  
    definition 82  
    Routing 109  
attribute point 83  
    rules for use with shape points 83  
attribute point list 83  
Attributes  
    ADAS 134  
attributes 81  
    fixed 81

    flexible 81  
attribution status 86  
auxiliary route 357

**B**

bannered routes 357  
base link feature 114  
    geometry 120  
Basic Map Display 201  
    building block 201  
    content 202  
    data 202  
    drawing order 215  
    drawing style 213  
    features 207  
    flexible attributes 87  
    labeling hints 226  
    levels 203  
    metadata 205  
    point feature 208  
    polygon type 211  
    structure 202  
    z level 216  
Batched Dynamic Adaptive Meshes (BDAM) 294  
Batched Dynamic Adaptive Meshes, levels 298  
BDAM, see Batched Dynamic Adaptive Meshes 294  
bit stream 55  
bitmask for filtering POI attributes 262  
BLOB 49  
    BLOB encryption 101, 103  
body geometry  
    references to geometry information 335  
body geometry type 326  
bounding box  
    overlapping 321  
bounding box  
    rotation 317  
    spatial extension of 3D object 315  
building block 44  
    3D Objects 309  
    Basic Map Display 201

- content index
  - 99
- definition 46
- dependencies Routing to BMD 130
- Digital Terrain Model 289
- extension 57
- Full-text Search 345
- Junction View 341
- mapping of tables to building blocks 72
- metadata (definition) 53
- Name 163
- Orthoimages 305
- POI 253
- Routing 107
- Speech 281
- Traffic Information 229
- building footprints and 3D 226, 336
  
- C**
- capital relation 174
- car light required 76
- cases, grammatical 283
- change case marker 195
- character code chart 72
  - unicode 72
- character code list, see character code
- chart 72
- child route 357
- cipher algorithm 103
- cipher key structure 104
- City 209
- city model
  - 2.5D 223
- city model 2.5D 223
- class-specific metadata (Name) 170
- clipping at tile borders (general concept) 96
- clothoid 137
- coding
  - coordinates 89
  - Morton code 91
- collection (icons) 356
- color gradient 332
- common metadata 53
  
- D**
- data encryption 101
- data model version table 62
- data quality attributes 86
- database
  - architecture 43
  - attributes 49
  - building block (definition) 46
  - content 49
  - data types 49
  - definition (NDS) 43
  - encryption 101
  - features 49
  - features (definition) 49
  - file structure 59
  - interfaces 43
  - levels (definition) 47
  - metadata 49
  - metadata (definition) 53
  - objects 49
  - partitioning 44

- product database (definition) 45  
SQLite version 43  
storing content 49  
structure 44  
subdivisions 44  
update region 45  
DataScript 53  
decryption 105  
default icons 357  
definite articles for TTS 286  
dependencies  
    Basic Map Display to Names 206  
    Basic Map Display to Routing 206  
    Basic Map Display to Traffic Information 207  
    Names to Basic Map Display 166  
    Names to Routing 166  
    Names to Speech 166  
    POI to 3D 258  
    POI to BMD 258  
    POI to Routing 257  
    POI to Speech 258  
    Routing to BMD 112, 130  
    Routing to Junction View 112  
    Routing to Names 112, 129  
    Routing to Traffic Info 112  
    Traffic to Speech 231  
detached from terrain attribute 222  
diacritic character 71  
diacritic transliteration indicator 71  
Digital Audio Broadcasting (DAB) 229  
Digital Terrain Model  
    BDAM content 302  
    BDAM data 302  
    BDAM structure 302  
    building block 289  
    height map structure 290  
    height maps 290  
    height maps content 290  
    height maps data 290  
digital terrain model  
    BDAM 294  
digitization status 86  
double metaphones 351  
    fuzzy search 351  
downward reference (Routing) 132  
drawing order 215  
    road geometry lines 125  
drawing style 213  
driver assistance, see ADAS 134
- E**
- eco routing 151  
    consumption speed curve 154  
    CSC 152  
    metadata 153  
    overview of factors 152  
    slope 152, 159  
    speed variation 152, 157  
EGM 96 324  
EGM96 89  
elevation 324  
    scaling factor 324  
elevation profile (ADAS) 143  
EN ISO 14819 232  
encryptable item 105  
encryption 101  
    algorithm 103  
    BLOB 103  
    cipher algorithm 103  
    cipher key structure 104  
    database 101  
    decryption 105  
    encryptable item 105  
    file encryption 102  
    scope 101  
enhanced ADAS attributes (elevation) 143  
enhanced ADAS geometry (clothoids) 137  
enhanced geometry for intersections 145  
enhancement of NDS 57  
event code 229  
event code (TMC) 232  
event information (TMC)  
    lookup table 239  
event information (TPEG) 249  
    lookup table 250  
Extension building block 57

**F**

feature 49  
 3D Object 312  
 aggregation 171  
 area feature 211  
 Basic Map Display 207  
 definition 49  
 feature attributes (general concept) 52  
 feature ID 50  
 global feature ID 51  
 line feature 209  
 link feature 113  
 linking to traffic information 237  
 name string feature 177  
 named object feature 167  
 POI feature 259  
 point feature 208  
 referencing features 51  
 road geometry line 123  
 road geometry line feature 122  
 Routing building block 112  
 structure 50  
 feature class  
 3D Objects 320  
 aggregated feature class for spatial node 320  
 area feature class 211  
 definition 49  
 examples (figure) 50  
 intersection feature class 126  
 line feature class 209  
 link feature class 113  
 multi-level hierarchy 49  
 named object feature class 167  
 POI feature class 259  
 point feature class 208  
 real-world object 49  
 road geometry line feature 122  
 feature ID 50  
 components 50  
 global ID 51  
 schema 50  
 file encryption 101, 102  
 filter POI secondary attributes 262

fixed attribute  
 area feature 212  
 line feature 210  
 links 117  
 name string feature 177  
 named objects 171  
 point feature 209  
 road geometry lines 123  
 fixed attributes 81  
 flexible attribute  
 attribution status 86  
 Basic Map Display 87  
 digitization status 86  
 for assigning named objects to Routing and BMD features 88  
 line feature 210  
 links 119  
 named objects 173  
 point feature 209  
 restriction 82  
 road geometry line features 125  
 Routing 87  
 VALIDITY\_RANGE 86  
 flexible attribute map 76  
 car light required 76  
 legal speed limit 76  
 warning sign icons 76  
 flexible attributes 81  
 in NVC trees 191  
 footprints and 3D 226, 336  
 Frequency Modulation (FM) 229  
 Full-text Search 345  
 content 346  
 data 346  
 fuzzy search 351  
 metadata 347  
 named object features 349  
 POI 348  
 structure 346  
 fuzzy search  
 double metaphones 351

**G**

generic profile attributes 145  
 geographic data (partitioning) 89

geographical extent 172  
geometry  
  base link feature 120  
  link features 120  
geometry information  
  3D Objects 322  
global feature ID 51  
global metadata  
  definition 53  
grammatical cases 283  
grouping of attributes 81  
guidance 107  
guidance-point relation (POI) 266

## H

height maps 290  
  compression 294  
  content 290  
  data 290  
  levels 291  
  structure 290

## I

icon  
  bannered routes 357  
  basic map display 213  
  default icons 357  
  definition in metadata 355  
  description of icon collection 356  
  for road numbers 77  
  icon collection 356  
  icon collection ID 356  
  icon set 356  
  icon table 355  
  PNG 355  
  POIs as icons 270  
  road number 356  
  scale range 355  
  sprites 357  
  usage type flag 355  
icon set 356  
icon sprites 357  
index  
  content indices for update regions 99

inter-building block reference 51, 129  
inter-level reference 51, 132  
international waters (country code) 70, 180  
intersection  
  attributes 127  
  connection between links 127  
  enhanced geometry (ADAS) 145  
  feature 127  
  feature class 126  
  number of links 127  
  position 127  
  reference 128  
  references of intersection  
  features 128  
  transition 127

inter-tile reference 52, 128  
intra-tile reference 52  
ISO 70  
ISO 15924 alpha 4 script code 70  
ISO 3166-1 alpha 3 country code 70  
ISO 639-3 language code 70  
ISO 8601 68  
ISO country code  
  for country named objects 180  
  international waters 70, 180  
  metadata 70

## J

Junction View 341  
  content 342  
  data 342  
  structure 342

## L

labeling hints 226  
language  
  in NVC trees 193  
  language code 177  
  language definition 70  
  language names 70  
  name string feature 178  
  selection 70  
language definition 70

- example 70
  - language names 70
  - latitude 89
  - leaf node
    - reconstructing information 191
  - least significant bit 54
  - left-hand traffic 74
  - legal speed limit 76
  - level 44, 47
    - Basic Map Display 203
    - Batched Dynamic Adaptive Meshes 298
    - height maps 291
    - level definition 47
    - map scales 203
    - Name 165
    - Routing 111
    - tiles 93
  - level metadata 73
    - overview 73
  - level of detail
    - 3D Objects 325
  - line feature 209
    - fixed attributes 210
    - flexible attributes 210
  - linear location (TMC) 233
  - link
    - base link feature 114
  - link feature
    - attributes 117
    - connections in intersections 127
    - feature class 113
    - fixed attributes 117
    - flexible attributes 119
    - geometry 120
    - number in intersections 127
    - references 120
    - references of link features 120
    - route link feature 115
  - linking traffic information to NDS features 237
  - local attribute list (TMC) 245
  - location code 229
  - location code (TMC)
    - primary 235
    - secondary 235
  - location code table (TMC) 233
  - location information (TMC) 233
    - lookup table 241
  - location information (TPEG) 251
  - location input 163
    - Full-text Search 345
    - NVC tree 186
    - POI 278
    - selection criteria 184
    - selection graph 182
  - Location Referencing Container (LRC) 249
  - logical mapping of tables to building blocks 72
  - logical-access relation (POI) 265
  - longitude 89
  - loops of dominant routes 357
- M**
- mainline route 357
  - map display 201
  - map labeling 226
  - map line reference 130
  - map scale 203
  - material
    - ambient lighting 331
    - color gradient 332
    - diffuse lighting 331
    - for 3D Objects 331
    - lighting pairs 331
    - specular lighting 331
    - texture reference 331
    - texture type 331
  - metadata
    - Basic Map Display metadata 205
    - building block-specific metadata 53
    - class-specific metadata (Name) 170
    - definition 53
    - eco routing 153
    - Full-text Search metadata 347
    - global metadata 53
    - Name metadata 165
    - NDS database 49
    - Orthoimages 307
    - POI metadata 255

product-specific 63  
 region-specific 73  
 Routing metadata 111  
 update region-specific 73  
 metaphones 351  
 Morton code 91  
 most significant bit 54  
 multi-level hierarchy (feature classes) 49  
 multipolygons 212

**N**

Name  
 building block 163  
 class-specific metadata 170  
 content 164  
 levels 165  
 metadata 165  
 name data 164  
 structure 164  
 Name building block  
 references from 3D Objects 336

name format  
 zip code range 180  
 name string feature 177  
 attributes 177  
 language attribute 178  
 language code attribute 177  
 name format attribute 177, 178  
 non-printable control and separation characters attribute 178, 182  
 relation type attribute 178, 180  
 use type 178  
 use type attribute 177  
 named object feature  
 access location 172  
 attributes 171  
 feature class 167  
 Full-text Search 349  
 geographical extent 172  
 properties 170  
 references 198  
 references to NVC trees 176  
 relations 173  
 NDS

add-ons 55  
 architecture 43  
 DataScript 53  
 encryption 101  
 enhancements 55  
 extension to the standard 55  
 introduction 41  
 NDS database supplier table 61  
 NDS database, see database 43  
 next-valid-character tree 186  
 normal in 3D space 323  
 numbers 54  
 NVC tree 186  
 change case marker 195  
 compiling named objects 187  
 disambiguation 192  
 flexible attributes 191  
 languages 193  
 mixed-case characters 194  
 reconstructing information from leaf and reference nodes 191

**O**

object level hierarchy 319  
 Orthoimages  
 building block 305  
 content 306  
 data 306  
 format 306  
 metadata 307  
 structure 306  
 tiling scheme 306  
 Overall building block  
 data 73  
 overall metadata table 62

**P**

parent route 357  
 partitioning  
 database 44  
 geographic data 89  
 tiles 92  
 part-of relation (POI) 264  
 phoneme 281

- phonetic transcription 281  
**PhoneticTranscriptionAuxTable** 283  
**POI** 253
  - activate along route 266
  - associated area features 258
  - attributes 256
  - building block 253
  - category 267
  - content 253
  - data 253
  - feature 259**Full-text Search** 348  
 location input 278  
 metadata 255  
 part-of relation 264  
 primary attribute 257, 259  
 references to 3D objects 258  
 region 273  
 relations 263  
 secondary attribute 257, 260  
 secondary attribute (bitmask) 262  
 structure 253  
 vanity 276  
 versioning 279  
 virtual 276  
 virtual tiles 277  
**POI building block**  
 references to 3D objects 336  
**POI category** 267  
**POI feature** 259  
**point feature** 208
  - fixed attributes 209
  - flexible attributes 209**point location (TMC)** 233
  - sequence 234**points of interest (see POI)** 253  
**polygon**
  - multipolygons 212
  - pseudo edges 213
  - type 211**prepositions for TTS** 283, 286  
**prerecorded voice data** 282  
**primary attribute (POI)** 257, 259  
**product database** 44, 45
  - world overview map 61**product database table** 45, 59  
**ProductDbTable** 59  
**profiles**
  - region-specific 79**property, see attribute** 52  
**proprietary extensions to NDS** 55  
**PSF, see NDS** 41
- R**
- RDS (Radio Data System)** 232  
**RDS, see relational DataScript** 53  
**real-world object** 49  
**reference**
  - direction 85
  - inter-building block reference 129
  - inter-level reference 132
  - intersection features 128
  - link features 120
  - map line 130
  - named object features 198
  - patterns for references 51
  - road geometry line features 125
  - route link features 122
  - Routing features 128
  - unidirectional 85**reference node**
  - reconstructing information 191**reference position** 171  
**references**
  - 3D Objects to Name building block 336
  - POIs to 3D objects 336**region (POI)** 273  
**region metadata** 73, 74
  - additional icons 78
  - address format 79
  - descriptive selection criteria names 76
  - flexible attribute map 76
  - overview 74
  - right- or left-hand traffic 74
  - time zones 79**region metatdata**
  - region-specific profiles 79**region-specific data** 73

- region-specific descriptive names of selection criteria 76  
region-specific profiles 79  
relation  
    access (POI) 265  
    access-at-road 174  
    between named objects 173  
    between POIs 263  
    capital 174  
    contained-in 174  
    crossroad-to-road 176  
    for name strings 180  
    guidance-point (POI) 266  
    logical-access (POI) 265  
    part-of relation (POI) 264  
    road-to-house 176  
relational DataScript 53  
render group 328  
    example 328  
    information defined per render group 329  
    winding 328  
rendering of maps 201  
right-hand traffic 74  
road geometry line feature  
    attributes 123  
    feature class 122  
    fixed attributes 123  
    flexible attributes 125  
    references 125  
road geometry lines  
    drawing order 125  
road icons 73  
road number class prefix list  
    example 78  
road number class prefix table 77  
    data 77  
road number icons 77, 356  
road numbers 77  
    transliteration 77  
road signs (banner) 357  
Road-to-house relation 176  
route calculation 107  
route guidance 107  
route guidance attributes 82  
route link feature 115  
    references 122  
Routing 107  
    algorithm 108  
    attribute layer 109  
    attributes 109  
    building block 107  
    content 108  
    data 108  
    dependencies to Basic Map  
    Display 112, 130  
    dependencies to Junction View 112  
    dependencies to Names 112, 129  
    dependencies to Traffic Information 112  
    features 112  
    flexible attributes 87  
    inter-building block reference 129  
    inter-level reference 132  
    inter-tile reference 128  
    levels 111  
    metadata 111  
    references of Routing features 128  
    routing geo tile 110  
    routing tile 110  
    routing tile and routing geo tile 110  
    structure 108  
routing  
    eco routing 151  
    routing features 112  
    routing geo tile 110  
    routing tile 110  
R-tree 311
- S
- satellite images (see Orthoimages) 305  
scale sublevel 204  
secondary attribute (POI) 257, 260  
    bitmask for filtering 262  
selection criteria 184  
selection graph 182  
shape point 83  
    geometry for link features 120  
    rules for use with attribute points 83  
signed numbers 54  
slope 152, 159

- data structures for eco routing 161
- slope profile (ADAS) 136
- spatial index structure 311
  - spatial node 312
  - update region 312
- spatial node 312
  - inner 312
  - leaf 312
  - reference 312
- Speech 281
  - building block 281
  - content 282
  - data 282
  - structure 282
- speech input 281
- speech output 281
- speed profiles 145
- speed variation 152, 157
- SPEED\_PROFILES\_OR\_AVG\_SPEEDS\_PER\_WEEK 146
- sprite (icon) 357
- spur of a dominant route 357
- SQLite version 43
- structure
  - database 44
  - feature 50
  - references 51
- supplementary information code (TMC) 232
  
- T**
- table
  - mapping of tables to building blocks 72
- template body geometry 327
  - 3D vector 327
  - attributes 327
- template buildings 326
- terrain model, see Digital Terrain Model 289
- texture
  - clamp (texture type) 331
  - material reference 331
  - mirror (texture type) 331
  - repeat (texture type) 331
- textures 333
  - applying textures 334
  - for 3D Objects 331
  - properties 333
  - UV mapping 333
- tile
  - anchor point 95
  - clipping at borders 96
  - content index 99
  - levels 93
  - routing and routing geo tile 110
  - tile patterns 214
  - tiling scheme 92
  - tiling scheme 92
    - Orthoimages 306
  - time patterns 145
  - time period attributes 86
  - time zone data 73
  - time zones 79
    - data 80
    - names 80
    - region time zone table 80
- TMC 229
  - ALERT-C standard 232
  - area location 233
  - BLOB 230
  - data design 238
  - determining location for traffic event 235
  - EN ISO 14819 232
  - event code 232
  - event information 232
  - linear location 233
  - local attribute list 245
  - local lists of flexible attributes 230
  - location code table 233
  - location information 233
  - lookup table for event information 239
  - lookup table for location information 241
  - point location 233
  - point location sequence 234
  - primary location code 235
  - Radio Data System (RDS) 232

- relational DB model for event information 240  
relational DB model for location information 242  
secondary location code 235  
supplementary information code 232  
TMC location hierarchy 236  
TMC lookup table 230  
traffic event direction 235  
traffic event extent 235  
TMC concepts 232  
TMC event information 232  
TMC, see Traffic information 229  
TPEG 229, 249  
Application Data Containers (ADC) 249  
data design 251  
event information 249  
location information 251  
Location Referencing Container (LRC) 249  
lookup table for event information 250  
referencing TMC locations 251  
TPEG lookup tables 231  
traffic event (TMC)  
    direction 235  
    extent 235  
Traffic Information 229  
    building block 229  
    content 230  
    data 230  
    metadata 231  
    structure 230  
    TMC concepts 232  
    VICS, see VICS 252  
Traffic Information Building Block  
    Digital Audio Broadcasting (DAB) 229  
    event code 229  
    Frequency Modulation (FM) 229  
    location code 229  
    TMC, see TMC 229  
    TPEG, see TPEG 229  
    VICS 229  
Traffic Message Channel  
    see TMC 229
- transcription 281  
transition 127  
transliteration 71  
    road numbers 77  
transliteration indicator 71  
transliteration languages 70  
transparent enhancement 57  
Transport Protocol Experts Group  
    see TPEG 229  
triangle strip 326  
truck attributes 82  
TTS 281  
    articles 283  
    definite article 283  
    grammatical cases 283  
    prepositions 283  
    prepositions and articles 286  
two's complement representation 54
- U**
- unicode 72  
unidirectional reference 85  
unsigned numbers 54  
update region 44, 45  
    content index  
        99  
    spatial index structure 312  
upward reference (Routing) 132  
usage type flag 355  
UV mapping 333
- V**
- validity range 83, 86  
vanity POI 276  
vector in 3D space 323  
Vehicle Information and Communication System  
    see VICS 252  
versioning of POIs 279  
VICS 252  
virtual POI 276  
virtual tile  
    content index  
        99

POI 277  
voice files 282

## W

warning sign icons 76  
WGS 84 89

world overview map 61

## Z

z level 216  
ZIP code range 180  
zip code range 180



## F Index of DataScript Terms

### A

ACCESS 263, 265  
 ACCESS\_AT\_ROAD 173, 174, 176  
 accessLocationList 172  
 acousticOutput 77  
 ACTIVATION 263, 267  
 activationRadius 267, 269  
 adasAttributeTypeAvailability 111, 134  
 addIconSetId 78  
 addIconType 78, 357  
 AdditionalIconAttributes 356  
 AdditionalIconRefTable 78, 357  
 AddressFormatTable 79  
 AddressPointHouseNumber 200  
 addressSeparator 79  
 ADMIN\_HIERARCHY\_NAMED\_OBJECT\_REFERENCE 88  
 administrativeRoadClass 124  
 AdminRoadClass 117  
 adminRoadClass 78, 357  
 adminStructureType 256, 276  
 aggregatedFeature3DClasses 314, 320  
 ALT 358  
 ALTERNATE\_NAME 178  
 ALTERNATE\_SPELLING 181  
 ANTI\_LABELING\_HINT 227  
 areNameStringsInUpperCase 165  
 attrBitMask 262  
 AttrBitMaskDefinition 262  
 attributeList 173  
 AttributePointList 83  
 ATTRIBUTION\_STATUS 86, 88  
 AVERAGE\_SLOPE 161  
 averageSpeed 118, 123, 156

### B

BASE\_ELEVATION 225, 226  
 baselineMapId 60  
 BASE\_NAME 181  
 baseSpeed 148  
 BBlockCompVersionTable 64, 68

bBlockCompVersionTable 231  
 BG\_INDEXED\_TRIANGLE\_STRIP 323  
 BG\_TEMPLATE\_GEO\_TYPE\_POSITION 328  
 BG\_TEMPLATE\_GEO\_TYPE\_POSITION\_AR  
 RAY 328  
 BG\_TEMPLATE\_GEO\_TYPE\_POSITION\_RO  
 TATION 328  
 BG\_TEMPLATE\_GEO\_TYPE\_POSITION\_RO  
 TATION\_ARRAY 328  
 BG\_TEMPLATE\_GEO\_TYPE\_POSITION\_RO  
 TATION\_SCALE 328  
 BG\_TEMPLATE\_GEO\_TYPE\_POSITION\_RO  
 TATION\_SCALE\_ARRAY 328  
 bmdAttributeTypeAvailability 206  
 bmdFeatureClasses 206, 311, 320  
 bmdFeatureClassHierarchyTable 205  
 bmdMetadata 205, 206  
 bmdTilePatternTable 214  
 BmdTileTable 93  
 bmdTileTable 101  
 bodyGeometries 322  
 BodyGeometry 322  
 BodyGeometryBlob 322, 335  
 bodyGeometryBlobId 322  
 bodyGeometryContent 323  
 BodyGeometryIndexedTriangleStrip 32  
 6, 327  
 bodyGeometryIndexedTriangleStrip 32  
 3, 329  
 bodyGeometryReference 315, 335  
 BodyGeometryType 323, 326  
 bottomHeightOverGround 316  
 BoundingBox 315  
 boundingBox 314  
 boundingRectangle 173  
 bridge 125  
 Building 223  
 BuildingBlockDetailedType 206, 256,  
 289  
 buildingBlockDetailedType 66  
 buildingBlockId 57, 66, 69  
 buildingBlockMetadata 67

buildingBlockName 66  
 BuildingBlockTable 57, 63, 65  
 BuildingBlockType 284  
 buildingBlockType 66  
 BUILDING\_HEIGHT 225, 226  
 BUSINESS 358  
 BUSINESS\_LOOP 358  
 BUSINESS\_ROUTE 358  
 BYPASS 358

**C**

CAPITAL\_RELATION 173, 174  
 categoryId 278  
 catId 269  
 ChangeCaseMask 196  
 characterCharCodeColl 75  
 childCatId 268  
 childSubTree 314  
 cipherKeyInfoId 103, 291  
 CipherKeyInfoTable 64, 102, 103, 104, 307  
 cipherKeyInfoTable 104  
 CITY 358  
 cityName 350  
 ClassID 51  
 CLOTHOID 135  
 colorDepth 343  
 colorId 64  
 ColorTable 64  
 columnName 69  
 compilerConfiguration 67  
 compilerVersion 67  
 complexIntersection 118  
 CONCRETE\_NAMED\_OBJECT 197  
 CONNECTOR 358  
 consecutiveQuads 328  
 CONSUMPTION\_DOWN\_EXCESS\_SLOPE 161  
 CONSUMPTION\_SPEED\_DEPENDENCY 156  
 ConsumptionSpeedDependencyCurve 156  
 ConsumptionSpeedValue 156  
 CONSUMPTION\_SPEED\_VARIATION 159

CONSUMPTION\_UP\_EXCESS\_SLOPE 161  
 CONTAINED\_IN 173, 174, 350  
 containsDetailedCityModel 206  
 CONTINUED\_TURN\_RESTRICTION 109  
 coord 209, 210, 212  
 coordShift 74  
 coordShiftXY 322  
 coordShiftZ 322  
 CoordXYOffset 200  
 copyright 61  
 countryName 349, 350  
 creationDateTime 68  
 crossroadFtsDoubleMetaphTable 351  
 crossroadFtsTable 346, 350  
 CROSSROAD\_TO\_ROAD 173, 176  
 CURRENCY 88, 109  
 CURVATURE 83, 135

**D**

dataElementId 57  
 dataModelVersionId 67, 68  
 DataModelVersionTable 62  
 DATE\_RANGE\_OF\_YEAR 87, 109  
 dayLight 343  
 dayLightOffset 80  
 DAY\_OF\_YEAR 87  
 DAYS\_OF\_MONTH 87  
 DAYS\_OF\_WEEK 87  
 DAYTIME\_RUNNING\_LIGHT 76  
 defaultLanguageCode 193  
 defaultName 80  
 DEFAULT\_OFFICIAL\_NAME 178  
 degreeOfFreedom 139  
 DETACHED\_FROM\_TERRAIN 218, 223  
 DIGITIZATION\_STATUS 86, 88  
 docId 349, 350  
 downExessSlopeThreshold 111, 153, 161  
 DRAWING\_ORDER 210, 215  
 DtmBdamPatternTileTable 303  
 DtmBdamTileTable 93  
 DtmHeightMapTileTable 93, 290  
 dtmImageFormat 74, 290  
 dtmImageHeightResolution 74  
 DtmSurfaceTile 302

DtmSurfaceTin 302

## E

EAST 358  
eic 67  
ELEVATION\_DELTA 225  
elevationDelta 323, 324  
EncryptableItemCodeTable 64  
encryptableItemCodeTable 104, 105  
endAngle 118  
endDayOfWeek 80  
endMonthOfYear 80  
endWeekOfMonth 80  
ENHANCED\_GEOMETRY 119  
ENUMERATION 260  
EQUIPMENT\_AND\_OCCUPANCY 109  
event 240, 250  
events 231  
excessSlope 161  
exitIconSetId 77  
EXIT\_NUMBER 180  
EXONYM 181  
EXPLICIT 126  
EXTENDED\_CLOTHOID 75  
EXTENDED\_CLOTHOID\_META\_DATA 75  
EXTENDED\_GRADIENT 143  
EXTENDED\_POSTAL\_CODE\_POSITION 1  
91  
EXTENSION 57  
externalTileIdList 110  
extTileIdList 173

## F

feature3DClass 314, 320  
FERRY 118  
ferry 125  
FERRY\_TRAVEL\_TIME 87  
FERRY\_TYPE 87, 109  
FIRST\_APPROPRIATE 283, 284  
FIRST\_LETTER\_INPUT 181  
FixedRoadAttributeSet 126  
FixedRoadAttributeSetList 126  
float16 334  
FLOOR\_NUMBER 336

followUseCasePrefixString 286  
fontId 64  
FontTable 64, 214  
format 177, 178  
FREQUENTLY\_USED\_VEHICLE\_TYPES 1  
09  
fromLink 112, 121, 126, 342  
FtsMetadataTable 347  
FULLY\_ATTRIBUTED 86  
FULLY\_DIGITIZED 86  
FunctionalRoadClass 117  
functionalRoadClass 124

## G

GENERATE 177  
genericFontId 64  
GLOBAL\_FEATURE\_ID 51, 52, 254, 342  
GlobalGatewayTable 68, 69  
grammaticalCase 286  
GROUND\_HEIGHT 224, 225, 226  
groupedAttributeList 75  
guidanceAttributeTypeAvailability 111  
GUIDANCE\_POINT 263, 266

## H

HAS\_3D\_OBJECT 226  
hasAccessLocationList 172  
hasAdaptations 62  
hasAdasBasicTypes 75, 134  
hasAdasBezierSpline 75, 134  
hasAdasExtendedTypes 75, 134  
HAS\_ADDITIONAL\_DATA 57  
HAS\_ADDITIONAL\_DATA\_ELEMENT\_ID  
57  
hasAdditionalDataElementId 57  
hasAdvancedAdminSearch 256  
hasAmbientAndDiffuse 331  
hasAttributes 340  
hasBmdTiles 73, 205  
hasBoundingRectangle 173  
hasColorPerVertex 332  
hasControlCharacters 178, 182  
hasCurvedLabelingLink 227  
hasDaylightSaving 80

hasDtmTiles 73  
 hasEcoRoutingAttributes 111, 153  
 HAS\_JUNCTION\_VIEW 112, 121, 126, 341  
 hasMetricSystem 75  
 hasOrthoimageTiles 73  
 hasPoiVirtualTiles 73, 278  
 hasPosition 171  
 hasRegionSpecificNames 185  
 hasRightHandDriving 74  
 hasRoutingTiles 73, 111  
 hasSpecularAndShininess 331  
 hasTextureCoords 329, 334  
 hasTextureCoordsAdditional 334  
 hasTileList 173  
 hasTmcTiles 73  
 HAS\_VICS 229, 252  
 height 343  
 heightOffset 291  
 heightResolution 290  
 HOUSE\_NUMBER 179  
 HOUSE\_NUMBER\_RANGE 179

**I**

Icon3d 358  
 IconCollectionTable 356  
 IconData 358  
 iconDisplayArrangement 271  
 iconDrawingPriority 271  
 iconInMapOutput 77  
 iconSetId 77, 170, 185, 269, 340, 356  
 IconSetTable 355  
 iconSpriteRef 357  
 IconSpriteTable 357  
 IconTable 214, 270, 355, 356, 358  
 id 290  
 image 333  
 ImageFormat 307  
 imageID 342  
 imageType 343  
 insideCityLimitStatus 118, 124  
 INTERCONNECTING\_ROADS 86  
 intermediateLinksList 342  
 intoUseCasePrefixString 286  
 isAggregationFeature 171

IS\_COUNTRY\_CAPITAL 209  
 isDestination 171  
 isDiacriticTransliterationOf 71  
 isDirty 61, 65, 66, 69  
 isInclusive 82, 147  
 isIntegratedPoiBuildingBlock 256  
 isMixedCaseSupported 165  
 isoCountryCode 75  
 isoSubCountryCode 75  
 isOverviewMap 61  
 isPartiallyFilled 66  
 isPoiNvcTree 278  
 isProtected 269  
 isPseudo 213  
 isTransliterationOf 71, 77  
 isTransparent 331

**J**

jamCategory 240  
 JvFrom2ToLink 112, 121, 126  
 JvFrom2ToLinkTable 342  
 JvFromLink 112, 121, 126  
 JvFromLinkTable 342  
 jvImageFormat 343  
 JvImageTable 342

**K**

KM\_MI\_INDICATOR 75

**L**

LABELING\_HINT 227  
 landmarks3DIconsTable 340  
 languageCode 76, 77, 80, 177, 178, 194, 243  
 LanguageDescription 194  
 languageDescription 194  
 LanguageTable 64, 70  
 length 118  
 LENGTH\_ALONG\_LINK 109, 117  
 LevelMetadataTable 73, 111, 205, 278, 290  
 levelNumber 51, 73  
 LICENSE\_PLATE\_ZONE 170

LINK 358  
 LINK\_PERCENTAGE 156  
 LINK\_TYPE 117  
 linkType 125  
 LOAD 177  
 LocalIndex 51  
 LOCAL\_TMC\_LOCATION 230  
 LocalTmcLocation 245  
 location 242  
 locCategory 243  
 locId 243, 244  
 locSubtype 243  
 locTableId 243, 245  
 locType 243  
 LOGICAL\_ACCESS 263, 265  
 longSide 316  
 LOOP 358

**M**

MAJOR\_ROADS\_ONLY 86  
 mapDataReleaseDate 65  
 maximumBasicMapDisplayStringLength 166  
 maximumEventNameStringLength 231  
 maximumLocationInputStringLength 166  
 maximumLocationNameStringLength 231  
 maximumPoiAttributeNameStringLength 256  
 maximumPoiCategoryNameStringLength 256  
 maximumPoiNameStringLength 256  
 maximumRoutingStringLength 166  
 maximumSignpostStringLength 166  
 maxScaleDenominator 61, 74  
 METADATA\_REGION\_REFERENCE 74  
 MINIMUM\_SPEED 88  
 minScaleDenominator 61, 74  
 mixedCaseDescriptionList 165  
 MONTHS\_OF\_YEAR 87  
 mortonCode 259  
 motorway 118  
 MOTORWAY\_INTERSECTION\_NAME 180

**N**

NAME 179  
 nameAttributeMapList 151  
 nameAttributeTypeAvailability 111  
 NamedObject 167, 173  
 NamedObjectAttributes 171, 173  
 namedObjectAttributeTypeAvailability 166  
 NAMED\_OBJECT\_BASE\_REF 88, 336  
 namedObjectClass 79, 173  
 namedObjectClassAttributeTypes 170  
 NamedObjectClassDefinition 165, 170  
 namedObjectClassList 185  
 NamedObjectId 167  
 namedObjectId 167  
 NamedObjectPhonemeTable 167, 284  
 namedObjectPhonemeTable 167  
 NamedObjectPrerecordedVTable 167, 285  
 NAMED\_OBJECT\_REFERENCE 88, 336  
 NamedObjectRelationType 173  
 NAMED\_OBJECT\_RELATIVE\_REF 88, 336  
 NamedObjectString 177, 178  
 NamedObjectTable 165  
 namedObjectTable 167  
 NamedObjectToPrerecordedVTable 167  
 nameFormat 79  
 nameList 185  
 NameMetadata 165, 195, 196  
 nameString 76  
 nameStringId 258  
 NDS\_3D 358  
 ndsData 291  
 NdsDatabaseSupplierTable 61, 104  
 ndsDbSupplierId 60, 61  
 ndsDbSupplierName 61  
 negativeLocId 246  
 NoAggregationFeatureAttributes 173  
 noAggregationFeatureAttributes 171, 172  
 NON\_DEFAULT\_TRAFFIC\_SENSE 88  
 NO\_RELATION 181  
 NORTH 358  
 NOT\_AVAILABLE 177, 283, 284

NOT\_FIELD\_SURVEYED 86  
 numberPrefixId 357  
 numContainedNames 191  
 numFeatureClasses 205  
 numLanguages 194  
 numNames 194  
 numNamesInDefaultLanguage 194  
 numOfChildrenStartWith2 313  
 numVertices 123  
 numXYBits 322, 323  
 numZBits 322  
 NvcGeneralPoiOrNamedObjectReference  
   194  
 NvcGeneralPoiOrNameObjectReference  
   193  
 nvcPoiReference 278  
 NvcSelectionCriteriaOptions 176  
 nvcStringSeparationCharacter 65  
 NvcSubTreeBlob 193  
 nvctree 190  
 NvcTreeAvailabilityType 176  
 nvcTreeNodeAttributeTypeAvailability  
   166  
 NvcTreeTable 176

**O**

objectHierarchyLevel 314, 319  
 OFFICIAL\_ABBREVIATION 180  
 OFFICIAL\_NAME 178  
 onOddDaysOnly 87  
 ordinalNumber 123  
 origin 316, 322  
 origin.height 317  
 OrthoImageMetadata 307  
 OrthoImageTileTable 93, 306  
 OrthoImageType 307  
 OverallMetadataTable 62  
 OVERPASS 88

**P**

parentCatId 268  
 PARENT\_NVC 173  
 parentSegmentId 244  
 PARKING 87

PARTLY\_ATTRIBUTED 86  
 PARTOF 263  
 pathNr 342  
 PATH\_VECTOR\_3D 75  
 phone 260  
 phonetictranscription 231  
 phoneticTranscriptionAuxId 286  
 PhoneticTranscriptionAuxTable 285,  
   286  
 PhoneticTranscriptionFormat 282, 284  
 phoneticTranscriptionId 258  
 PHYSICAL\_WIDTH\_IMPERIAL 87, 135  
 PHYSICAL\_WIDTH\_METRIC 87, 135  
 pluralJunction 118  
 POIATTR\_ACTIVATION 261  
 POIATTR\_ADDRESS 261  
 POIATTR\_BRAND\_NAME 261  
 POIATTR\_FOOD\_TYPE 261  
 POIATTR\_MULTIMEDIA 261, 262  
 PoiAttrNameStringRelationTable 261  
 PoiAttrNameStringTable 261  
 PoiAttrNameToPhoneticTraTable 258,  
   285  
 PoiAttrNameToPrerecordedVTable 258,  
   285  
 POIATTR\_OPENING\_HRS 261  
 PoiAttrToStringTable 261  
 POIATTR\_WEBSITE 261  
 POICAT\_ACCESS\_POINT 265  
 POICAT\_ACTIVATION\_POINT 267  
 poiCategory 349  
 PoiCategoryDescription 278  
 PoiCategoryTable 267, 268  
 POICAT\_GUIDANCE\_POINT 266  
 PoiCatNameStringRelationTable 270  
 PoiCatNameStringTable 270  
 PoiCatNameToPhoneticTraTable 258,  
   284  
 PoiCatNameToPrerecordedVTable 258,  
   285  
 POICAT\_NDSGENERAL 270  
 PoiCatRelationTable 268  
 PoiCatToAttrTable 256  
 PoiCatToStringTable 270  
 poiCityName 349  
 poiEmail 349

PoiFtsDoubleMetaphTable 351  
 PoiFtsTable 346, 348, 349  
 PoiGeoAccess 265  
 poiHouseNumber 349  
 poiIconTable 270, 355  
 PoiIntersectionAccess 254, 265  
 PoiLinkAccess 254, 265  
 PoiMetadata 256  
 PoiMetadataTable 254, 256, 262, 276  
 poiName 349  
 PoiNamedObjectRelationTable 254  
 PoiNameStringRelationTable 260  
 PoiNameStringTable 260  
 PoiNameToPhoneticTraTable 258, 284  
 PoiNameToPrerecordedVTable 258, 285  
 poiPhone 349  
 PoiPhoneticTraTable 258, 285  
 poiPostalCode 349  
 PoiPrerecordedVTable 231, 258, 285  
 PoiRelationTable 264, 266  
 PoiRelationType 263  
 PoiScaleLevelTable 269  
 PoiServiceLocationTable 259, 268, 273  
 poiStreetName 349  
 PoiTable 259, 260, 262, 265, 279  
 PoiToNameStringTable 260  
 poiUrl 349  
 PoiVirtualTileTable 278, 279  
 polygonFlag 213  
 polygonsAreTriangulated 206  
 POPULATION 209  
 position 342  
 positiveLocId 246  
 postalCode 260, 350  
 POSTAL\_CODE\_NAMED\_OBJECT\_REFERE  
 NCE 88  
 PREFERRED 263  
 PREFERRED\_ALTERNATE\_NAME 178  
 prefix 77  
 prefixId 77  
 prefixWithIconOutput 77  
 prerecordedvoice 231  
 PrerecordedVoiceFormat 283  
 ProductDatabase 62, 73  
 ProductDbTable 45, 59, 60

productId 60  
 productName 60  
 profileId 147  
 profileList 146  
 PROHIBITED\_PASSAGE 109, 119  
 pseudo edge (polygons) 213

## Q

QuarterHourTimeOffset 80

## R

referringToExternalFeature 172  
 regionId 74, 76, 77, 78, 80, 259, 273, 357  
 RegionMetadata 146  
 RegionMetadataTable 134  
 regionNamedObjectReference 74  
 RegionProfileTable 79  
 RegionSpecificDescriptiveTable 76  
 RegionTimeZoneTable 80  
 relationType 178, 180  
 releaseMonth 65  
 releaseYear 65  
 RENDER 358  
 RenderGroup 328  
 roadClass 77  
 RoadFtsDoubleMetaphTable 351  
 RoadFtsTable 346, 349  
 RoadGeoLineList 51  
 roadHeightLevel 218  
 roadName 350  
 ROAD\_NUMBER 179  
 RoadNumber 357  
 RoadNumberClassPrefixTable 77, 231, 357  
 roadNumberDisplayClass 77  
 ROAD\_TO\_HOUSE 173, 176  
 ROAD\_Z\_LEVEL 87, 210, 218  
 ROOF\_COLOR 225, 226  
 roofColorIdRef 225  
 rotatedIconRef 213  
 routeLinkFeatureId 123  
 routeLinkRefType 256  
 routingAttributeTypeAvailability 111

RoutingAuxTileTable 93, 120, 129, 151  
 RoutingGeoTile 129  
 RoutingMetadata 111, 153  
 RoutingRoadAttributes 117, 124

**S**

scaleDenominatorList 74  
 scaleLevelId 269  
 scaleSublevel 209, 210, 212  
 SCENIC 109, 125  
 searchTags 349, 350, 351  
 SelectionCriteriaDefinition 165, 176, 184  
 selectionCriterionCode 76  
 selectionEntry 269  
 SelectionGraphDefinition 165  
 SelectionGraphTable 183  
 SelectionMetadataTable 184  
 seqNr 342  
 serviceLocationId 278  
 ShapePointInfo 120  
 ShapePointInfoList 120  
 ShapePointList 120  
 ShapePointType 51, 120  
 shapes 123  
 SharedRoadAttributes 118, 124, 125  
 ShortLeafNode 278  
 shortSide 316  
 SIGNPOST 170  
 SimpleArrayHeightMapData 294  
 SLOPE 87, 125, 135  
 SOUTH 358  
 SpatialTreeInnerNode 312  
 SpatialTreeLeafNode 312  
 SpatialTreeNode 313, 340  
 SpatialTreeRefNode 312  
 SPEED\_LIMIT 88, 109, 119  
 SPEED\_PROFILES\_OR\_AVG\_SPEEDS\_PER\_WEEK 146, 147  
 SPLINE 75  
 SPRITE 357  
 SPUR 358  
 sql\_database 72  
 STANDARD 283, 284  
 StandardCat 267

standardCat 269  
 standardDeviation 139  
 startAngle 118  
 startDayOfWeek 80  
 startMonthOfYear 80  
 startTimeOfDay 80  
 startWeekOfMonth 80  
 stateName 349, 350  
 street1Name 350  
 street2Name 350  
 stringId 241, 243, 251  
 STUBBLE 88  
 SubImage 331  
 subImageArray 334  
 suburbName 349, 350  
 supportedAttributes 256, 260  
 SYNONYM 181

**T**

tableName 69  
 tableNumber 251  
 TELEPHONE\_NUMBER 179  
 TEMP 358  
 TemplateBodyGeometry 327, 329  
 TemplateBodyGeometryId 327  
 TemplateBodyGeometryTable 327  
 TemplateVector3d 327  
 textOutput 77  
 textureImageFormat 333  
 textureMapId 333  
 TextureMapTable 333  
 TextureReference 331  
 threeDAtributeTypeAvailablility 311  
 THREE\_D\_LANDMARK\_REF 258  
 TileContentIndexTable 63, 99  
 tileId 51, 245  
 TileNumber 51  
 TIME\_RANGE\_OF\_DAY 87, 109  
 timeZoneId 80  
 timeZoneNameString 80  
 TimeZoneNameTable 80  
 TIME\_ZONE\_REFERENCE 80, 119  
 TimeZoneTable 80  
 TiPhoneticTraTable 231, 285  
 TiPrerecordedVTable 231, 285

TmcAreaLocationTable 243  
 TmcCrossReferenceTable 244  
 tmcDataType 246  
 tmcDirection 246  
     NEGATIVE\_DIRECTION 246  
     POSITIVE\_DIRECTION 246  
 TmcEventCostInfoTable 241  
 TmcEventNameTable 241  
 TmcEventPrerecordedVTable 231  
 TmcEventTable 240  
 TmcExternRefTable 245  
 tmcLinkType 246  
     ENTRY 246  
     EXIT 246  
     EXTERNAL 246  
     INTERNAL 246  
 TmcLocationCoordinateTable 244  
 TmcLocationListTable 243  
 TmcLocationPrerecordedVTable 231  
 TmcLocationTableIdTable 243  
 TmcLocationTypeTextTable 243  
 TmcNameTable 243  
 TmcPointLocationTable 243  
 TmcSegmentLocationTable 243  
 TmcStationListTable 245  
 TmcStationTileListTable 245  
 TmcSupplementaryInfoTable 241  
 TmcTileReferenceTable 244  
 TmcTileTable 93, 230, 245  
 toLink 112, 121, 126, 342  
 TOLL 118, 358  
 toll 125  
 TOLL\_CHARGE 109  
 TOLL\_VIGNETTE 109  
 topHeightOverGround 316  
 TOTAL\_MAP\_LINE 131  
 TOURIST\_ROUTE\_TYPE 88  
 TpegEventPrerecordedVTable 231  
 TpegTecEventNameTable 251  
 TRANSITION\_MASK\_\* 127  
 TRANSLITERATION 181  
 travelDirection 118, 125  
 TreeInnerNode 190  
 TreeLeafNode 190  
 TreeNodeType 190  
 TreeRefNode 191, 193, 194

TreeShortLeafNode 190, 193, 194  
 truckAttributeTypeAvailability 111  
 TUNNEL 118  
 tunnel 125  
 type 313  
 TYPE\_OF\_PAVEMENT 87, 109, 125  
  
**U**  
 UNDEFINED\_LANGUAGE\_CODE 70, 178  
 UNDER\_CONSTRUCTION 88  
 UNDERPASS 88  
 UNKNOWN 118  
 updateRegionId 65, 66, 69  
 updateRegionNamedObjectRef 65  
 UpdateRegionTable 63, 65  
 upExessSlopeThreshold 111, 153, 161  
 urban 118  
 UrBuildingBlockVersionTable 64, 66,  
     101, 104, 105, 231, 258, 282  
 urBuildingBlockVersionTable 167  
 uri 61, 67, 69  
 usageType 177, 178  
 USE\_ALTERNATE 177  
 useBlobEncryption 67, 101, 102  
 useBoundingRectangle 173  
 usedAdminRoadClasses 75  
 useDefaultLanguage 193, 194  
 useFileEncryption 67, 101, 102  
 utcOffset 80

**V**  
 VALIDITY\_RANGE 83, 86  
 Vector2D 317  
 Vehicle 267  
 versionId 60, 65, 66, 67, 69, 290  
 versionName 67  
 VersionTable 59, 64, 67, 307

**W**  
 WALL\_COLOR 225, 226  
 wallColorIdRef 225  
 weather 307, 343  
 WeekdayInMonth 87

WEST 358  
width 343

Z

ZIP\_CODE\_RANGE 180

Y

Year 87