

Open Geospatial Consortium

Submission Date: <yyyy-mm-dd>

Approval Date: <yyyy-mm-dd>

Publication Date: <2020-09-17>

External identifier of this OGC® document: <http://www.opengis.net/doc/IS/CityGML-1/3.0>

Internal reference number of this OGC® document: 20-010

Version: 0.9

Category: OGC® Conceptual Model

Editors: Thomas H. Kolbe, Tatjana Kutzner, Carl Stephen Smyth, Claus Nagel, Carsten Roensdorf,
Charles Heazel

OGC City Geography Markup Language (CityGML) Part 1: Conceptual Model Standard

Copyright notice

Copyright © 2020 Open Geospatial Consortium

To obtain additional rights of use, visit <http://www.opengeospatial.org/legal/>

Warning

This document is not an OGC Standard. This document is distributed for review and comment. This document is subject to change without notice and may not be referred to as an OGC Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: OGC® Standard

Document subtype: Conceptual Model

Document stage: Draft

Document language: English

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

Table of Contents

1. Introduction	9
2. Scope	10
3. Conformance	12
3.1. Conceptual Models	12
3.2. Implementation Specifications	12
3.3. Conformance Classes	12
4. References	14
5. Terms and Definitions	15
6. Conventions	17
6.1. Identifiers	17
6.2. UML Notation	17
6.3. Conceptual Modelling	19
7. Overview of CityGML	21
7.1. Modularisation	21
7.2. General Modelling Principles	22
7.3. Representation of Spatial Properties	24
7.4. CityGML Core Model: Space Concept, Levels of Detail, Special Spatial Types	27
7.5. Appearances	33
7.6. Modelling Dynamic Data	33
7.7. Extending CityGML	35
8. CityGML UML Model	37
8.1. Structural Overview of Requirements Classes	37
8.2. Core	38
8.3. Appearance	52
8.4. City Furniture	57
8.5. City Object Group	60
8.6. Dynamizer	63
8.7. Generics	69
8.8. Land Use	75
8.9. Point Cloud	77
8.10. Relief	79
8.11. Transportation	82
8.12. Vegetation	92
8.13. Versioning	95
8.14. Water Body	98
8.15. Construction	102
8.16. Bridge	110
8.17. Building	117

8.18. Tunnel	126
9. CityGML Data Dictionary	134
9.1. ISO Classes	134
9.2. Core	143
9.3. Appearance	169
9.4. CityFurniture	177
9.5. CityObjectGroup	179
9.6. Dynamizer	182
9.7. Generics	194
9.8. LandUse	203
9.9. PointCloud	205
9.10. Relief	206
9.11. Transportation	211
9.12. Vegetation	231
9.13. Versioning	236
9.14. WaterBody	240
9.15. Construction	244
9.16. Bridge	263
9.17. Building	272
9.18. Tunnel	286
10. Application Domain Extension (ADE)	295
10.1. General Rules for ADEs	295
10.2. Defining New ADE Model Elements	295
10.3. Augmenting CityGML Feature Types with Additional ADE Properties	296
10.4. Encoding of ADEs	298
10.5. Requirements and Recommendations	298
Annex A: Abstract Test Suite (Normative)	301
A.1. Introduction	301
A.2. Conformance Class Core	301
A.3. Conformance Class Appearance	304
A.4. Conformance Class CityFurniture	305
A.5. Conformance Class CityObjectGroup	306
A.6. Conformance Class Dynamizer	306
A.7. Conformance Class Generics	307
A.8. Conformance Class LandUse	308
A.9. Conformance Class PointCloud	309
A.10. Conformance Class Relief	309
A.11. Conformance Class Transportation	310
A.12. Conformance Class Vegetation	311
A.13. Conformance Class Versioning	312
A.14. Conformance Class WaterBody	312

A.15. Conformance Class Construction	313
A.16. Conformance Class Bridge	314
A.17. Conformance Class Building	315
A.18. Conformance Class Tunnel	315
Annex B: Revision History	317
Annex C: Changelog for CityGML 3.0	318
Annex D: Glossary	319
11. ISO Concepts	321
12. Abbreviated Terms	325
Annex E: Bibliography	327

i. Abstract

CityGML is an open conceptual model for the storage and exchange of virtual 3D city models. It is defined through a Unified Modeling Language (UML) object model. This UML model builds on the ISO Technical Committee 211 (ISO/TC211) conceptual model standards for spatial and temporal data. Building on the ISO foundation assures that the man-made features described in the city models share the same spatio-temporal universe as the surrounding countryside within which they reside.

The aim of the development of CityGML is to reach a common definition of the basic entities, attributes, and relations of a 3D city model. This is especially important with respect to the cost-effective sustainable maintenance of 3D city models, allowing the reuse of the same data in different application fields.

ii. Keywords

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, CityGML, 3D city models

iii. Preface

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

More content to be provided.

iv. Submitting organizations



This is the official CityGML logo. For current news on CityGML and information about ongoing projects and fields of research in the area of CityGML see <http://www.citygml.org> and <http://www.citygmlwiki.org>

NOTE

PNG interlace method not supported for PDF generation. Find a more compatible OGC logo for "image::images/OGC_Logo.png[]"

This Document was submitted to the Open Geospatial Consortium (OGC) by the members of the CityGML Standards Working Group of the OGC. Amongst others, this comprises the following

organizations:

- To be provided
- ...

v. Submitters

All questions regarding this submission should be directed to the editors or the submitters:

Table 1. Submission Contact Points

Name	Institution	E-mail
Thomas H. Kolbe	Chair of Geoinformatics, Technical University of Munich, Germany	thomas.kolbe@tum.de
Tatjana Kutzner	Chair of Geoinformatics, Technical University of Munich, Germany	kutzner@tum.de
Carl Stephen Smyth	OpenSitePlan, USA	steve@opensiteplan.org
Claus Nagel	virtualcitySYSTEMS, Germany	cnagel@virtualcitysystems.de
Carsten Roensdorf	Ordnance Survey, Great Britain	Carsten.Roensdorf@os.uk
Charles (Chuck) Heazel	HeazelTech LLC	cheazel@heazeltech.com

vi. Participants in development

In addition to the Editors of the specification the following individuals contributed to the CityGML 3.0 development:

Table 2. Participants in Development

Name	Institution
Giorgio Agugiaro	3D Geoinformation Group, Delft University of Technology, the Netherlands
Christof Beil	Chair of Geoinformatics, Technical University of Munich, Germany
Filip Biljecki	Department of Architecture, National University of Singapore, Singapore
Kanishk Chaturvedi	Chair of Geoinformatics, Technical University of Munich, Germany
Volker Coors	Stuttgart University of Applied Sciences, Germany
Emmanuel Devys	Institut national de l'information géographique et forestière (IGN), France
Jürgen Ebbinghaus	AED-SICAD, Germany
Heinrich Geerling	Architekturbüro Geerling, Germany
Gilles Gesquière	LIRIS, University of Lyon, France

Name	Institution
Gerhard Gröger	CPA ReDev GmbH, Germany
Karl-Heinz Häfele	Institute for Automation and Applied Informatics, Karlsruhe Institute of Technology, Germany
Marc-Oliver Löwner	Institute for Geodesy and Photogrammetry, Technische Universität Braunschweig, Germany
Diana Moraru	Ordnance Survey, Great Britain
Friso Penninga	Geonovum, the Netherlands
Helga Tauscher	Faculty of Spatial Information, HTW Dresden - University of Applied Sciences, Germany
Linda van den Brink	Geonovum, the Netherlands
Heidi Vanparys	Danish Agency for Data Supply and Efficiency, Denmark
Sisi Zlatanova	Faculty of Built Environment, University of New South Wales, Australia

The table only lists persons that were involved in the development of CityGML 3.0. CityGML 3.0 is based on extensive previous work that was done for CityGML 2.0. For persons involved in the previous work, please refer to the CityGML 2.0 specification.

vii. Acknowledgements

The editors wish to thank the Special Interest Group 3D (SIG 3D) of the initiative Geodata Infrastructure Germany (GDI-DE) which originally started the development of CityGML, the CityGML Standards Working Group and the 3D Information Management (3DIM) Working Group of the OGC as well as all contributors of change requests and comments.

Chapter 1. Introduction

An increasing number of cities and companies are building virtual 3D city models for different application areas like urban planning, mobile telecommunication, disaster management, 3D cadastre, tourism, vehicle and pedestrian navigation, facility management and environmental simulations. Furthermore, in the implementation of the European Environmental Noise Directive (END, 2002/49/EC) 3D geoinformation and 3D city models play an important role.

In recent years, most virtual 3D city models have been defined as purely graphical or geometrical models, neglecting the semantic and topological aspects. Thus, these models could almost only be used for visualisation purposes but not for thematic queries, analysis tasks, or spatial data mining. Since the limited reusability of models inhibits the broader use of 3D city models and may not justify the costs associated with maintaining city models, a more general modelling approach had to be taken in order to satisfy the information needs of the various application fields.

CityGML is a common semantic information model for the representation of 3D urban objects that can be shared over different applications. The latter capability is especially important with respect to the cost-effective sustainable maintenance of 3D city models, allowing governments and companies to reap the benefits of their investment in 3D city models by being able to put the same models into play in different application fields. The targeted application areas explicitly include city planning, architectural design, tourist and leisure activities, environmental simulation, mobile telecommunication, disaster management, homeland security, real estate management, vehicle and pedestrian navigation, and training simulators.

CityGML is an open conceptual model for the storage and exchange of virtual 3D city models. It is defined through a Unified Modeling Language (UML) object model. This UML model builds on the ISO Technical Committee 211 (ISO/TC211) conceptual model standards for spatial and temporal data. Building on the ISO foundation assures that the man-made features described in the city models share the same spatio-temporal universe as the surrounding countryside within which they reside.

CityGML defines the classes and relations for the most relevant topographic objects in cities and regional models with respect to their geometrical, topological, semantical, and appearance properties. “City” is broadly defined to comprise not just built structures, but also elevation, vegetation, water bodies, city furniture, and more. Included are generalisation hierarchies between thematic classes, aggregations, relations between objects, and spatial properties. CityGML is applicable for large areas and small regions, and can represent the terrain and 3D objects in different levels of detail simultaneously. Since both simple, single scale models without topology and few semantics as well as very complex multi-scale models with full topology and fine-grained semantical differentiations can be represented, CityGML enables lossless information exchange between different geographic information systems and users.

Chapter 2. Scope

This document is an OGC Conceptual Model Standard to specify the representation of virtual 3D city and landscape models. The CityGML 3.0 Conceptual Model is expected to be the basis for a number of future Encoding Standards in which subsets of the Conceptual Model can be implemented. These Encoding Standards will enable both storage and exchange of data. Support for GML encoding is expected as a minimum, though additional encodings in formats such as JSON and database schemas will be highly desirable.

In contrast to the previous CityGML 2.0, the 3.0 Conceptual Model is NOT implemented as an application schema of the Geography Markup Language. However, a preliminary encoding in GML version 3.2 is available as an informative resource separately from this specification to allow implementation and testing.

The target of the conformance classes specified in this document are

- CityGML encoding specifications that provide encodings for the UML conceptual model specified in this document, and
- Additional UML models that can be created by users to extend this conceptual model as Application Domain Extensions.

CityGML models comprise of georeferenced 3D vector data along with the semantics associated with the data. In contrast to other 3D vector formats, CityGML is based on a rich, general purpose information model in addition to geometry and appearance information that allows for the integration of a variety of source data to come together in a City Model. To enable the use of CityGML in specific domain areas, CityGML has historically provided an extension mechanism to enrich the data with identifiable features and properties, preserving semantic interoperability. As it is recognised that an implementable expansion mechanism might have dependencies on the encoding language, the CityGML 3.0 Conceptual Model specifies high level requirements rather than a full extension model.

Targeted application areas explicitly include urban and landscape planning; architectural design; tourist and leisure activities; environmental, energy and mobility simulations; mobile telecommunications; disaster management; homeland security; vehicle and pedestrian navigation; training simulators and mobile robotics.

The future CityGML 3.0 Encoding Standards will be implementable source formats for 3D portraying or transformation into dedicated portrayal formats such as the OGC i3s or the OGC 3D Tiles community standards, KML/COLLADA or glTF. The OGC 3D Portrayal Service may be used for content delivery.

Features of the CityGML 3.0 Conceptual Model:

- Geospatial Information Model (ontology) for urban landscapes based on the ISO 19100 family
- Representation of 3D geometries, based on the ISO 19107 model, independent of data encodings, as well as of 3D point clouds
- Grouping into space hierarchies, including concepts like stories/floors within buildings

- Representation of object surface characteristics (e.g. textures, materials)
- Representation of dynamic, i.e. time-dependent, properties of city models
- Taxonomies and aggregations
 - Digital Terrain Models as a combination of triangulated irregular networks (TINs), regular rasters, break and skeleton lines, mass points
 - Sites (currently buildings, other constructions, bridges, and tunnels)
 - Vegetation (areas, volumes, and solitary objects with vegetation classification)
 - Water bodies (volumes, surfaces)
 - Transportation facilities (graph structures, 3D space, and 3D surface data)
 - Land use (representation of areas of the earth's surface dedicated to a specific land use)
 - City furniture
 - Generic city objects and attributes
 - User-definable (recursive) grouping
- Multiscale model with 4 well-defined consecutive Levels of Detail (LOD), applicable to both interior and exterior:
 - LOD0 – Highly generalised model
 - LOD1 – Block model / extrusion objects
 - LOD2 – Realistic, but still generalised model
 - LOD3 – Highly detailed model
- Multiple representations in different LODs simultaneously; generalisation relations between objects in different LODs
- Ability to combine different interior and exterior LoDs, including representation of floor plans
- Optional topological connections between feature (sub)geometries
- Enables a variety of different encoding specifications, including GML and JSON
- Extension of the conceptual model through code lists, generic objects and Application Domain Extensions (ADEs)
- With CityGML 3.0, ADEs become platform-independent models on a conceptual level that can be mapped to multiple and different target encodings. ADEs are implemented as UML models that extend the conceptual model in this specification. This includes a mechanism that favours the insertion of additional feature properties into any defined feature class through 'hooks' over subtyping of features. This means that the existing feature classes can be used and additional properties from one or more ADEs can easily be supported in different encodings
- Ability to specify an ADE that can be further extended

Chapter 3. Conformance

This standard defines a [Conceptual Model](#) which is independent of any encoding or formatting techniques. The [Standardization Targets](#) for this standard are:

1. [Conceptual Models](#) (extended versions of this conceptual model)
2. [Implementation Specifications](#) (encodings of this conceptual model)

3.1. Conceptual Models

A Conceptual Model standardization target is a version of the CityGML 3.0 Conceptual Model tailored for a specific user community. This tailoring can include:

1. Omission of one or more of the optional Packages
2. Reduction of the multiplicity for an attribute or association
3. Restriction on the valid values for an attribute
4. Additional concepts documented through ADEs.

Of these options, actions #1, #2, and #3 can be performed when creating an implementation specification. Only action #4 requires an extension of the CityGML Conceptual Model. These extensions SHALL be accomplished using the ADE mechanism described in [Section 10](#) Application Domain Extensions (ADE).

Extensions of the CityGML Conceptual Model SHALL conform with the ADE Conformance Class.

3.2. Implementation Specifications

Implementation Specifications define how a Conceptual Model shall be implemented using a specific technology. Conformant Implementation Specifications provide evidence that they are an accurate representation of the Conceptual Model. This evidence shall include implementations of the abstract tests specified in [Annex A](#) (normative) of this document.

Since this standard is agnostic to the implementing technologies, the specific techniques to be used for conformance testing cannot be specified. It is the responsibility of the Implementation Specifications to provide evidence of conformance which is appropriate for the implementing technologies. This evidence should be provided as an annex to the Implementation Specification document.

3.3. Conformance Classes

This standard identifies seventeen (17) conformance classes. One conformance class is defined for each Package in the UML model. Each conformance class is defined by one requirements class. The tests in [Annex A](#) are organized by Requirements Class. So an implementation of the *Core* conformance class must pass all tests specified in Annex A for the *Core* requirements class.

Of these seventeen conformance classes, only the *Core* conformance class is mandatory. All other

conformance classes are optional. In the case where a conformance class has a dependency on another conformance class, that conformance class shall also be implemented.

The CityGML Conceptual Model is defined by the CityGML UML model. This specification is a representation of that UML model in document form. In the case of a discrepancy between the UML model and this document, the UML model is authoritative.

Chapter 4. References

The following normative documents contain provisions that, through reference in this text, constitute provisions of OGC 20-010. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of OGC 20-010 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.

- IETF: RFC 2045 & 2046, Multipurpose Internet Mail Extensions (MIME). (November 1996),
- IETF: RFC 3986, Uniform Resource Identifier (URI): Generic Syntax. (January 2005)
- INSPIRE: D2.8.III.2 Data Specification on Buildings – Technical Guidelines. European Commission Joint Research Centre.
- ISO: ISO 19101-1:2014, Geographic information - Reference model - Part 1: Fundamentals
- ISO: ISO 19103:2015, Geographic Information – Conceptual Schema Language
- ISO: ISO 19105:2000, Geographic information – Conformance and testing
- ISO: ISO 19107:2003, Geographic Information – Spatial Schema
- ISO: ISO 19109:2015, Geographic Information – Rules for Application Schemas
- ISO: ISO 19111:2019, Geographic information – Referencing by coordinates
- ISO: ISO 19123:2005, Geographic information — Schema for coverage geometry and functions
- ISO: ISO 19156:2011, Geographic information – Observations and measurements
- ISO: ISO/IEC 19505-2:2012, Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 2: Superstructure
- ISO/IEC 19507:2012, Information technology — Object Management Group Object Constraint Language (OCL)
- ISO: ISO/IEC 19775-1:2013 Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) — Part 1: Architecture and base components
- Khronos Group Inc.: COLLADA – Digital Asset Schema Release 1.5.0
- OASIS: Customer Information Quality Specifications - extensible Address Language (xAL), Version v3.0
- OGC: The OpenGIS® Abstract Specification Topic 5: Features, OGC document 08-126
- OGC: The OpenGIS™ Abstract Specification Topic 8: Relationships Between Features, OGC document 99-108r2
- OGC: The OpenGIS™ Abstract Specification Topic 10: Feature Collections, OGC document 99-110

Chapter 5. Terms and Definitions

This document uses the terms defined in Sub-clause 5.3 of [OGC 06-121r8], which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this Standard.

For the purposes of this document, the following additional terms and definitions apply.

2D data

geometry of features is represented in a two-dimensional space

NOTE In other words, the geometry of 2D data is given using (X,Y) coordinates.

[INSPIRE D2.8.III.2, definition 1]

2.5D data

geometry of features is represented in a three-dimensional space with the constraint that, for each (X,Y) position, there is only one Z

[INSPIRE D2.8.III.2, definition 2]

3D data

Geometry of features is represented in a three-dimensional space.

NOTE In other words, the geometry of 2D data is given using (X,Y,Z) coordinates without any constraints.

[INSPIRE D2.8.III.2, definition 3]

application schema

A set of [conceptual schema](#) for data required by one or more applications. An application schema contains selected parts of the base schemas presented in the ORM Information Viewpoint. Designers of application schemas may extend or restrict the types defined in the base schemas to define appropriate types for an application domain. Application schemas are information models for a specific information community.

OGC Definitions Register at <http://www.opengis.net/def/glossary/term/ApplicationSchema>

codelist

A value domain including a code for each permissible value.

conceptual model

model that defines concepts of a universe of discourse

[ISO 19101-1:2014, 4.1.5]

conceptual schema

1. formal description of a [conceptual model](#)

[ISO 19101-1:2014, 4.1.6]

2. base schema. Formal description of the model of any geospatial information. <>application-schema-definition,Application schemas> are built from conceptual schemas.

OGC Definitions Register at <http://www.opengis.net/def/glossary/term/ConceptualSchema>

Implementation Specification

levels of detail

quantity of information that portrays the real world

NOTE The concept comprises data capturing rules of spatial object types, the accuracy and the types of geometries, and other aspects of a data specification. In particular, it is related to the notions of scale and resolution.

[INSPIRE Glossary]

life-cycle information

set of properties of a spatial object that describe the temporal characteristics of a version of a spatial object or the changes between versions

[INSPIRE Glossary]

Chapter 6. Conventions

6.1. Identifiers

The normative provisions in this document are denoted by the URI

<http://www.opengis.net/spec/CityGML-1/3.0>

All requirements and conformance tests that appear in this document are denoted by partial URIs which are relative to this base.

6.2. UML Notation

The CityGML standard is presented in this document in diagrams using the Unified Modeling Language (UML) static structure diagram (see Booch et al. 1997). The UML notations used in this standard are described in the diagram in [Figure 1](#).

[UML Notation] | *images/UML_Notation.png*

Figure 1. UML notation (see ISO TS 19103, Geographic information - Conceptual schema language).

All associations between model elements in CityGML are uni-directional. Thus, associations in CityGML are navigable in only one direction. The direction of navigation is depicted by an arrowhead. In general, the context an element takes within the association is indicated by its role. The role is displayed near the target of the association. If the graphical representation is ambiguous though, the position of the role has to be drawn to the element the association points to.

The following stereotypes are used in this model:

- «ApplicationSchema» denotes a conceptual schema for data required by one or more applications. In the CityGML conceptual model, every CityGML module is defined as separate application schema to allow for modularisation.
- «FeatureType» represents features that are similar and exhibit common characteristics. Features are abstractions of real-world phenomena and have an identity.
- «TopLevelFeatureType» denotes features that represent the main components of the conceptual model. Top-level features may be further semantically and spatially decomposed and substructured into parts.
- «Type» denotes classes that are not directly instantiable, but are used as an abstract collection of operation, attribute and relation signatures. The stereotype is used in the CityGML conceptual model only for classes that are imported from the ISO standards 19107, 19109, 19111, and 19123.
- «ObjectType» represents objects that have an identity, but are not features.
- «DataType» defines a set of properties that lack identity. A data type is a classifier with no operations, whose primary purpose is to hold information.
- «Enumeration» enumerates the valid attribute values in a fixed list of named literal values. Enumerations are specified in the CityGML schema.
- «BasicType» defines a basic data type.

- «CodeList» enumerates the valid attribute values. In contrast to Enumeration, the list of values is open and, thus, not given inline in the CityGML UML Model. The allowed values can be provided within an external code list.
- «Union» is a list of attributes. The semantics are that only one of the attributes can be present at any time.
- «Property» denotes attributes and association roles. This stereotype does not add further semantics to the conceptual model, but is required to be able to add tagged values to the attributes and association roles that are relevant for the encoding.
- «Version» denotes that the value of an association role that ends at a feature type is a specific version of the feature, not the feature in general.

In order to enhance the readability of the CityGML UML diagrams, classes are depicted in different colors. The following coloring scheme is applied:

Class defined in this Requirements Class

Classes painted in yellow belong to the Requirements Class which is subject of discussion in that clause of the specification in which the UML diagram is given. For example, in the context of [Section 8.2](#), which introduces the *CityGML Core* module, the yellow color is used to denote classes that are defined in the *CityGML Core* Requirements Class. Likewise, the yellow classes shown in the UML diagram in [Section 8.17](#) are associated with the *Building* Requirements Class that is subject of discussion in that chapter.

Class defined in another Requirements Class

Classes painted in blue belong to a Requirements Class different to that associated with the yellow color. In order to explicitly denote to which Requirements Class these classes belong, their class names are preceded by the UML package name of that Requirements Class. For example, in the context of the *Building* Requirements Class, classes from the *CityGML Core* and the *Construction* Requirements Classes are painted in blue and their class names are preceded by *Core* and *Construction*, respectively.

**Class defined in ISO 19107,
ISO 19111 or ISO 19123**

Classes painted in green are defined in the ISO standards 19107, 19111, or 19123. Their class names are preceded by the UML package name, in which the classes are defined.

[ColorScheme grey]

Classes painted in grey are defined in the ISO standard 19109. In the context of this standard, this only applies to the class *AnyFeature*. *AnyFeature* is an instance of the metaclass *FeatureType* and acts as super class of all classes in the CityGML UML model with the stereotype «FeatureType».

Notes and OCL constraints

The color white is used for notes and OCL constraints that are provided in the UML diagrams.

The example UML diagram in [Figure 2](#) demonstrates the UML notation and coloring scheme used throughout this specification. In this example, the yellow classes are associated with the *CityGML Building* module, the blue classes are from the *CityGML Core* and *Construction* modules, and the green class depicts a geometry element defined by ISO 19107.

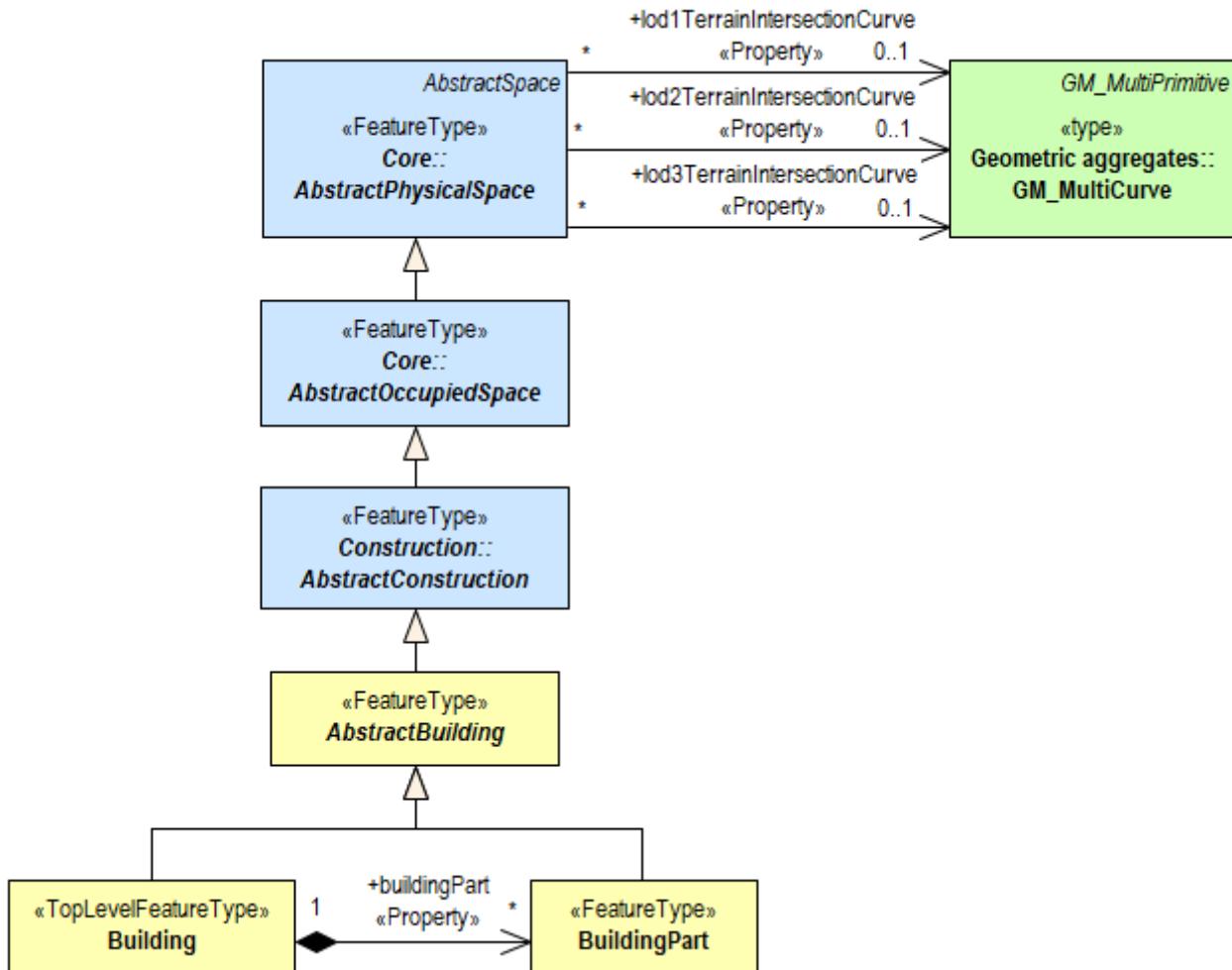


Figure 2. Example UML diagram demonstrating the UML notation and coloring scheme used throughout the CityGML specification.

6.3. Conceptual Modelling

ISO 19101 defines universe of discourse to be a view of the real or hypothetical world that includes everything of interest. That standard then defines conceptual model to be a model that defines concepts of a universe of discourse.

The scope of this CityGML Conceptual Model Standard establishes the limits of the universe of discourse for this Standard. The next task is to discover and standardize the concepts within this scope. CityGML will potentially support numerous diverse application software packages covering multiple disciplines and facility life cycle phases. Each conceivably can have its own universe of

discourse and their own set of concepts.

The goal of this CityGML Conceptual Model Standard is to establish and document a common set of concepts that spans the applications supported. This does not attempt to redefine application concepts, but merely present a common set of concepts from and to which their concepts can be understood and mapped.

GML and JSON encodings are planned and other encodings are anticipated. Each encoding addresses a specific information community and set of application software packages. However, with the increasing desire to share information between communities and applications having a common conceptual model across all of these encodings is highly advantageous.

An added benefit of the development of a conceptual model results from the rigor involved in achieving consensus. After numerous iterations, the end result is consistent, cohesive, and complete. Updating a conceptual model is far easier than rewriting software code. Further, the iterations help to flesh out details as well as to unearth differences in individual conceptualizations.

Perhaps the greatest benefit of the standards activity is the ability to communicate the resultant model. This is in part due to using a standardized conceptual modelling language like UML and the agreed OGC and ISO/TC211 conventions for using UML. The eventual outcome of being able to provide formal documentation for what is meant by each concept is invaluable in understanding the subsequent encodings and applications.

This will be the first OGC conceptual model standard without accompanying encodings. Yet the model is presented in a manner consistent with the formalisms adopted for writing OGC standards. This standard follows the [OGC Specification Model standard for modular specifications](#) and is consistent with the OGC Naming Authority conventions and recommendations. The target of this Standard are the encoding standards which will follow and not the application software that will implement these encodings. Requirements for the encodings are explicit and grouped into Requirements Classes. Accompanying Conformance Classes are included to determine if an encoding conforms to the conceptual model.

UML has been used as the conceptual modelling language in this Standard. Class Diagrams have been created and inserted as Figures. The boxes in these diagrams (officially “Classifiers” in UML) typically represent classes, data types, enumerations, code lists, unions, etc. and this terminology is used throughout the Standard. However, since this is a Conceptual Model, these should all be interpreted to be “concepts”. For each Requirements Class, an introductory diagram is included which contains all of the concepts relevant to that Requirements Class.

Though redundant with the UML diagrams, all of the classes, class attributes, and associations are repeated in the Data Dictionary in [Chapter 9](#). If these differ, the UML takes precedence.

Chapter 7. Overview of CityGML

CityGML is an open conceptual model for the storage and exchange of virtual 3D city and landscape models. This document defines the conceptual schema for the most relevant entities of the urban space like buildings, roads, railways, tunnels, bridges, city furniture, water bodies, vegetation, and the terrain. The conceptual schema specifies how and into which parts and pieces physical objects of the real world should be decomposed and classified. All objects can be represented with respect to their semantics, 3D geometry, 3D topology, appearances, and their changes over time. Different spatial representations can be provided for each object (outdoor and indoor) in four predefined Levels of Detail (LOD 0-3). The CityGML 3.0 Conceptual Model ([Chapter 8](#)) is formally specified using UML class diagrams, complemented by a data dictionary ([Chapter 9](#)) providing the definitions and explanations of the object classes and attributes. It is the basis for multiple encoding standards, which map the concepts (or subsets thereof) onto exchange formats or database structures for data exchange and storage.

While CityGML can be used for 3D visualization purposes, its special merits lie in applications that go beyond visualization like decision support, urban and landscape planning, urban facility management, Smart Cities, navigation (both indoor and outdoor), Building Information Modelling (especially for as-built documentation), integration of city and BIM models, assisted and autonomous driving, and simulations in general (cf. [Kolbe 2009](#)). A comprehensive overview on the many different applications of virtual 3D city models is given in [[Biljecki et al. 2015](#)]. Many of the applications already use and some even require using CityGML.

In CityGML all 3D city objects can easily be enriched with thematic data. For example, street objects can be enriched with information about traffic density, speed limit, number of lanes etc., or buildings can be enriched by information on the heating and electrical energy demand, numbers of households and inhabitants, the appraised building value etc. Even building parts like individual roof or wall surfaces can be enriched with information e.g. about solar irradiation and thermal insulation parameters. For many application domains specific extensions of the CityGML conceptual model have already been created (cf. [Biljecki et al. 2018](#)).

7.1. Modularisation

The CityGML conceptual model provides models for the most important types of objects within virtual 3D city and landscape models. These feature types have been identified to be either required or important in many different application areas. However, implementations are not required to support the overall CityGML conceptual model in order to be conformant to the standard, but may employ a subset of constructs according to their specific information needs. For this purpose, modularisation is applied to the CityGML conceptual model.

[ModuleOverview] | *images/ModuleOverview.png*

Figure 3. CityGML 3.0 module overview. The vertical boxes show the different thematic modules. Horizontal modules specify concepts that are applicable to all thematic modules.

The CityGML conceptual model is thematically decomposed into a *Core module* and different kinds of *extension modules* as shown in [Figure 3](#). The Core module (shown in green) comprises the basic concepts and components of the CityGML conceptual model and, thus, must be implemented by any

conformant system. Each red coloured module covers a specific thematic field of virtual 3D city models. CityGML introduces the following eleven thematic extension modules: *Building*, *Bridge*, *Tunnel*, *Construction*, *CityFurniture*, *CityObjectGroup*, *LandUse*, *Relief*, *Transportation*, *Vegetation*, and *WaterBody*. All three modules *Building*, *Bridge*, and *Tunnel* model civil structures and share common concepts that are grouped within the *Construction* module. The five blue coloured extension modules add specific modelling aspects that can be used in conjunction with all thematic modules. The *Appearance* module contains the concepts to represent appearances (like textures and colours) of city objects. The *PointCloud* module provides concepts to represent the geometry of city objects by 3D point clouds. The *Generics* module defines the concepts for generic objects, attributes, and relationships. *Versioning* adds concepts for the representation of concurrent versions, real world object histories and feature histories. The *Dynamizer* module contains the concepts to represent city object properties by time series data and to link them with sensors, sensor data services or external files.

Each CityGML encoding can decide to only support a subset of CityGML modules. If a module is supported by an encoding, all concepts shall be mapped. However, the encoding specification can define so-called *null mappings* to restrict the use of specific elements of the conceptual model in an encoding. Null mappings can be expressed in an encoding specification for individual feature types, properties, and associations defined within a CityGML module. This means that the corresponding element will not be included in the respective encoding.

Note that also CityGML applications do not have to support all modules. Applications can also decide to only support a specific subset of CityGML modules. For example, when an application only has to work with building data, only the modules *Core*, *Construction*, and *Building* would have to be supported.

7.2. General Modelling Principles

7.2.1. Semantic Modelling of Real-World Objects

Real-world objects are represented by geographic features according to the definition in ISO 19109. Geographic features of the same type (e.g. buildings, roads) are modelled by corresponding feature types that are represented as classes in the Conceptual Model. The objects within a 3D city model are instances of the different feature types.

In order to distinguish and reference individual objects, each object has unique identifiers. In CityGML 3.0 each geographic feature has the mandatory *featureID* and an optional *identifier* property. The *featureID* is used to distinguish all objects and their possibly existing multiple versions of the same real-world object. The *identifier* is identical for all versions of the same real-world object and can be used to reference specific objects independent from their actual object version. The *featureID* must at least be unique within the same CityGML dataset, but it is generally recommended to use globally unique identifiers like UUID values or identifiers maintained by an organisation like a mapping agency. It is also recommended to provide globally unique and stable identifiers for the *identifier* attribute. This means these identifiers should remain stable over the lifetime of the real-world object.

CityGML feature types typically have a number of spatial and non-spatial properties (also called attributes) as well as relationships with other feature or object types. Note that a single CityGML

object can have different spatial representations at the same time, for example, different geometry objects representing the feature's geometry in different levels of detail or as different spatial abstractions.

Many attributes have simple, scalar values like a number or a character string. However, some attributes are complex, i.e. they do not just have a single property value. In CityGML the following types of complex attributes occur:

- *qualified attribute values* – e.g. a measure consists of the value and a reference to the unit of measure, or e.g. for relative and absolute height levels the reference level has to be named, too
- *code list values* for enumerative attributes; in addition to the value a link to the code list definition should be provided
- attributes consisting of a *tuple of different fields and values* – e.g. addresses, space occupancy, and others
- attribute value consisting of a *list of numbers*, for example, to represent coordinate lists or matrices

In order to support history, CityGML 3.0 introduces bitemporal timestamps for all objects. Besides the attributes *creationDate* and *terminationDate* from CityGML 2.0, which refer to the time period in which a specific version of an object is an integral part of the 3D city model, all objects now can additionally have the attributes *validFrom* and *validTo*, which represent the lifespan a specific version of an object has in the real-world. With these two time intervals a CityGML dataset could be queried both for how did the *city* look alike at a specific point in time as well as how did the *city model* look at that time.

The combination of the two types of feature identifiers explained above and bi-temporal timestamps allows that not only the current version of a 3D city model, but also its entire history can be represented in CityGML and possibly exchanged even within a single file.

7.2.2. Class Hierarchy and Inheritance of Properties and Relations

In CityGML, the specific feature types like *Building*, *Tunnel*, or *WaterBody* are defined as subclasses of more general higher-level classes. Hence, feature types build a hierarchy along specialization / generalization relationships where more specialized feature types inherit the properties and relationships of all their superclasses along the entire generalization path to the topmost feature type *AnyFeature*.

7.2.3. Relationships between CityGML objects

In CityGML objects can be related to each other and different kinds of relations are distinguished. First of all, complex objects like buildings or transportation objects are typically consisting of parts. These parts are individual features of their own, and can even be further decomposed. Therefore, CityGML objects can form aggregation hierarchies. Some feature types are marked in the conceptual model with the stereotype «*TopLevelFeatureType*». These constitute the main objects of a city model and are typically the root of an aggregation hierarchy. Only top-level features are allowed as direct members of a *CityModel* object. The information which feature types belong to the top level is required for software packages that want to filter imports, exports, and visualizations according to the general type of a city object (e.g. only show buildings, solitary vegetation objects,

and roads). Application Domain Extensions of CityGML should also make use of this concept, such that software tools can learn from inspecting their conceptual schema what are the main, i.e. the top-level, feature types of the extension.

Some relations in CityGML are qualified by additional parameters, typically to further specify the type of relationship. For example, a relationship can be qualified with a URI pointing to a definition of the respective relation type in an Ontology. Qualified relationships are used in CityGML, among others, for

- general relationships between features – association *relatedTo* between city objects,
- user-defined aggregations using *CityObjectGroup* – this relation allows also for recursive aggregations,
- external references – linking of city objects with corresponding entities from external resources like objects in a cadastre or within a BIM dataset.

The CityGML conceptual model contains many relationships that are specifically defined between certain feature types. For example, there is the *boundary* relationship from 3D volumetric objects to its thematically differentiated 3D boundary surfaces. Another example is the *generalizesTo* relation between feature instances that represent objects on different generalisation levels.

In CityGML 3.0 there are new associations to express topologic, geometric, and semantic relations between all kinds of city objects. For example, it can be expressed that two rooms are adjacent or that one interior building installation (like a curtain rail) is overlapping with the spaces of two connected rooms. It can also be expressed that two wall surfaces are parallel and two others are orthogonal. Also distances between objects could be represented explicitly using geometric relations. In addition to spatial relations logical relations can be expressed.

7.2.4. Definition of the Semantics for all Classes, Properties, and Relations

The meanings of all elements defined in the CityGML conceptual model are normatively specified in the data dictionary in [Chapter 9](#).

7.3. Representation of Spatial Properties

7.3.1. Geometry and Topology

Spatial properties of all CityGML feature types are represented using the geometry classes defined in ISO 19107. Spatial representations can have 0-, 1-, 2-, or 3-dimensional extents depending on the respective feature type and Levels of Detail (LOD; the LOD concept is discussed in [Section 7.4.4](#) and [Section 8.2.5](#)). With only a few exceptions, all geometries must use 3D coordinate values. Besides primitive geometries like single points, curves, surfaces, and solids, CityGML makes use of different kinds of aggregations of geometries like spatial aggregates (*MultiPoint*, *MultiCurve*, *MultiSurface*, *MultiSolid*) and composites (*CompositeCurve*, *CompositeSurface*, *CompositeSolid*). Volumetric shapes are represented in ISO 19107 according to the so-called *Boundary Representation* (B-Rep, for explanation see [Foley et al. 2002](#)) only.

The CityGML conceptual model does not put any restriction on the usage of specific geometry types as defined in ISO 19107. For example, 3D surfaces could be represented in a dataset using 3D

polygons, 3D meshes – i.e. as triangulated irregular networks, or by non-uniform rational B-spline surfaces (NURBS). However, an encoding may restrict the usage of geometry types. For example, curved lines like B-splines or clothoids, or curved surfaces like NURBS could be disallowed by explicitly defining *null encodings* for these concepts in the encoding specification (c.f. [Section 7.1](#) above).

Note that the conceptual schema of ISO 19107 allows that composite geometries can be given by a recursive aggregation for every primitive type of the corresponding dimension. This aggregation schema allows the definition of nested aggregations (hierarchy of components). For example, a building geometry (*CompositeSolid*) can be composed of the house geometry (*CompositeSolid*) and the garage geometry (*Solid*), while the house's geometry is further decomposed into the roof geometry (*Solid*) and the geometry of the house body (*Solid*). This is illustrated in [Figure 4](#).

[RecursiveAggregation] | *images/RecursiveAggregation.png*

Figure 4. Recursive aggregation of objects and geometries in CityGML (graphic: IGG Uni Bonn).

While CityGML does not employ the topology classes from ISO 19107, topological relations between geometries can be established by sharing geometries (typically parts of the boundary) between different geometric objects. One part of real-world space can be represented only once by a geometry object and is referenced by all features or more complex geometries which are defined or bounded by this geometry object. Thus redundancy can be avoided and explicit topological relations between parts are maintained.

Basically, there are three cases for sharing geometries: First, two different semantic objects may be spatially represented by the same geometry object. For example, if a foot path is both a transportation feature and a vegetation feature, the surface geometry defining the path is referenced both, by the transportation object and by the vegetation object. Second, a geometry object may be shared between a feature and another geometry. For example, a geometry defining a wall of a building may be referenced twice: by the solid geometry defining the geometry of the building, and by the wall feature. Third, two geometries may reference the same geometry, which is in the boundary of both. For example, a building and an adjacent garage may be represented by two solids. The surface describing the area where both solids touch may be represented only once and it is referenced by both solids. As it can be seen from [Figure 4](#), this requires partitioning of the respective surfaces. In general, B-Rep only considers visible surfaces. However, to make topological adjacency explicit and to allow the possibility of deletion of one part of a composed object without leaving holes in the remaining aggregate, touching elements are included. Whereas touching is allowed, permeation of objects is not in order to avoid the multiple representation of the same space.

Another example for sharing geometry objects that are members of the boundaries in different higher-dimensional geometry objects is the sharing of point geometries or curve geometries, which make up the outer and inner boundaries of a polygon. This would allow that each point is only represented once, and different polygons could reference this point geometry. The same applies to the representation of curves for transportation objects like roads, whose end points could be shared e.g. between different road segments to topologically connect them.

Note that the use of topology in CityGML datasets by sharing geometries is optional. Furthermore, an encoding of the CityGML conceptual model might restrict the usage of shared geometries. For example, it might only be allowed to share identical (support) points from different 3D polygons or

only entire polygons can be shared between touching solids (like shown in [Figure 4](#)).

7.3.2. Prototypic Objects / Scene Graph Concepts

In CityGML, objects of equal shape like trees and other vegetation objects, traffic lights and traffic signs can be represented as prototypes which are instantiated multiple times at different locations (see [Figure 5](#)). The geometry of prototypes is defined in local coordinate systems. Every instance is represented by a reference to the prototype, a base point in the world coordinate reference system and a transformation matrix that facilitates scaling, rotation, and translation of the prototype. The principle is adopted from the concept of scene graphs used in computer graphics standards like X3D and COLLADA. Since the ISO 19107 geometry model does not provide support for scene graph concepts, the CityGML class `ImplicitGeometry` has been introduced (for further description see [Section 8.2.5](#)). The prototype geometry can be represented using ISO 19107 geometry objects or by referencing an external file containing the geometry in another data format like X3D or COLLADA.

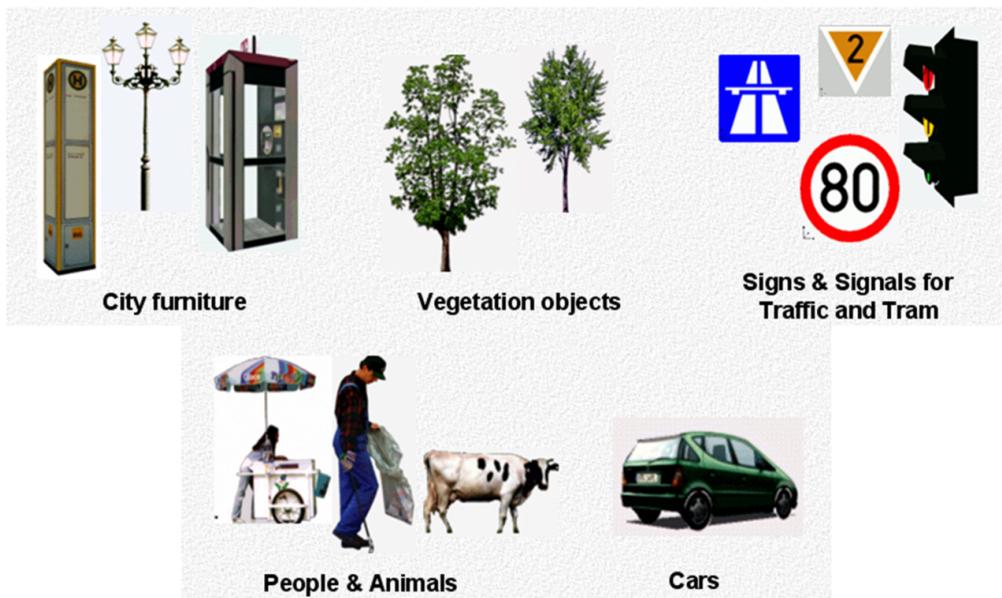


Figure 5. Examples of prototypic shapes (source: Rheinmetall Defence Electronics).

7.3.3. Point Cloud Representation

In addition to the spatial representations defined in the *Core* module, the geometry of physical spaces and of thematic surfaces can now also be provided by 3D point clouds using `MultiPoint` geometry. This allows, for example, to spatially represent the building hull, a room within a building or a single wall surface just by a point cloud. All thematic feature types including transportation objects, vegetation, city furniture, etc. can be spatially represented by point clouds, too. In this way, the `ClearanceSpace` of a road or railway could, for instance, be modelled directly from the result of a mobile laser scanning campaign. Point clouds can either be included in a CityGML dataset or just reference an external file of some common types such as LAS or LAZ.

7.3.4. Coordinate Reference Systems

CityGML is about 3D city and landscape models. This means that nearly all geometries use 3D coordinates, where each single point and also the points defining the boundaries of surfaces and solids have three coordinate values (x,y,z) each. Coordinates always have to be given with respect to

a coordinate reference system (CRS) that relates them unambiguously with a specific position on Earth. In contrast to CAD or BIM, each 3D point is absolutely georeferenced, which makes CityGML especially suitable to represent geographically large extended structures like airports, railways, bridges, dams, where the Earth curvature has a significant effect on the object's geometry (for further explanations see [Kaden & Clemen 2017](#)).

In most CRS, the (x,y) coordinates refer to the horizontal position of a point on the Earth's surface. The z coordinate typically refers to the vertical height over (or under) the reference surface. Note that it depends on the chosen CRS whether x and y are given as angular values like latitude and longitude or as distance values in meters or feet. In general, all kinds of 3D CRS according to ISO 19111 can be used. This includes global as well as national reference systems using geocentric, geodetic, or projected coordinate systems.

7.4. CityGML Core Model: Space Concept, Levels of Detail, Special Spatial Types

7.4.1. Spaces and Space Boundaries

In CityGML 3.0, a clear semantic distinction of spatial features is introduced by mapping all city objects onto the semantic concepts of spaces and space boundaries. A Space is an entity of volumetric extent in the real world. Buildings, water bodies, trees, rooms, and traffic spaces are examples for such entities with volumetric extent. A Space Boundary is an entity with areal extent in the real world. Space Boundaries delimit and connect Spaces. Examples are the wall surfaces and roof surfaces that bound a building; the water surface as boundary between the water body and air; the road surface as boundary between the ground and the traffic space; or the digital terrain model representing the space boundary between the over- and underground space.

To obtain a more precise definition of spaces, they are further subdivided into physical spaces and logical spaces. Physical spaces are spaces that are fully or partially bounded by physical objects. Buildings and rooms, for instance, are physical spaces as they are bounded by walls and slabs. Traffic spaces of roads are physical spaces as they are bounded by road surfaces against the ground. Logical spaces, in contrast, are spaces that are not necessarily bounded by physical objects, but are defined according to thematic considerations. Depending on the application, logical spaces can also be bounded by non-physical, i.e. virtual boundaries, and they can represent aggregations of physical spaces. A building unit, for instance, is a logical space as it aggregates specific rooms to flats, the rooms being the physical spaces that are bounded by wall surfaces, whereas the aggregation as a whole is being delimited by a virtual boundary. Other examples are city districts which are bounded by virtual vertically extruded administrative boundaries; public spaces vs. Security zones in airports; or city zones with specific regulations stemming from urban planning. The definition of physical and logical spaces and of corresponding physical and virtual boundaries is in line with the discussion in [\[Smith & Varzi 2000\]](#) on the difference between bona fide and fiat boundaries to bound objects. Bona fide boundaries are physical boundaries; they correspond to the physical boundaries of physical spaces in CityGML 3.0. In contrast, fiat boundaries are man-made boundaries; they are equivalent to the virtual boundaries of logical spaces.

Physical spaces, in turn, are further classified into occupied spaces and unoccupied spaces. Occupied spaces represent physical volumetric objects that occupy space in the urban

environment. Examples for occupied spaces are buildings, bridges, trees, city furniture, and water bodies. Occupying space means that some space is blocked by these volumetric objects; for instance, the space blocked by the building in [Figure 6](#) cannot be used any more for driving through this space or placing a tree on that space. In contrast, unoccupied spaces represent physical volumetric entities that do not occupy space in the urban environment, i.e. no space is blocked by these volumetric objects. Examples for unoccupied spaces are building rooms and traffic spaces. There is a risk of misunderstanding the term OccupiedSpace. However, we decided to use the term anyway, as it is established in the field of robotics for over three decades [[Elfes 1989](#)]. The navigation of mobile robots makes use of a so-called occupancy map that marks areas that are occupied by matter and, thus, are not navigable for robots.

[OccupiedAndUnoccupiedSpaces] | [images/OccupiedAndUnoccupiedSpaces.png](#)

Figure 6. Occupied and unoccupied spaces

The new space concept offers several advantages:

- In CityGML 3.0, all geometric representations are defined in the *Core* module only. This makes (a) models of the thematic modules simpler as they no longer need to be associated directly with the geometry classes, and (b) implementation easier as all spatial concepts have only to be implemented once in the *Core* module and all thematic modules like *Building*, *Relief*, *WaterBody*, etc. are inheriting them.
- The space concept supports the expression of explicit topological, geometrical, and thematic relations between spaces and spaces, spaces and space boundaries, and space boundaries and space boundaries. Thus, implementing the checking of geometric-topological consistency will become easier, because most checks can be expressed and performed on the CityGML *Core* module and then automatically apply to all thematic modules
- For the analysis of navigable spaces (e.g. to generate IndoorGML data from CityGML) algorithms can be defined on the level of the *Core* module. These algorithms will then work with all CityGML feature classes and also ADEs as they are derived from the *Core*. The same is true for other applications of 3D city models listed in [[Biljecki et al. 2015](#)] such as visibility analyses including shadow casting or solar irradiation analyses.
- Practitioners and developers do not see much of the space concept, because the space and space boundary classes are just abstract classes. Only elements representing objects from concrete subclasses such as *Building*, *BuildingRoom*, or *TrafficSpace* will appear in CityGML data sets.

7.4.2. Modelling City Objects by the Composition of Spaces

Semantic objects in CityGML are often composed of parts, i.e. they form multi-level aggregation hierarchies. This also holds for semantic objects representing occupied and unoccupied spaces. In general, two types of compositions can be distinguished:

1. **Spatial partitioning:** Semantic objects of either the space type OccupiedSpace or UnoccupiedSpace are subdivided into different parts that are of the same space type as the parent object. Examples are Buildings that can be subdivided into BuildingParts, or Buildings that are partitioned into ConstructiveElements. Buildings as well as BuildingParts and constructiveElements represent OccupiedSpaces. Similarly, Roads can be subdivided into TrafficSpaces and AuxiliaryTrafficSpaces, all objects being UnoccupiedSpaces.

2. Nesting of alternating space types: Semantic objects of one space type contain objects that are of the opposite space type as the parent object. Examples are Buildings (OccupiedSpace) that contain BuildingRooms (UnoccupiedSpace), BuildingRooms (UnoccupiedSpace) that contain Furniture (OccupiedSpace), and Roads (UnoccupiedSpace) that contain CityFurniture (OccupiedSpace). The categorization of a semantic object into occupied or unoccupied takes place at the level of the object in relation to the parent object. A building is part of a city model; thus, in the first place it occupies urban space within a city. As long as the interior of the building is not modelled in detail, the space covered by the building needs to be considered as occupied and only viewable from the outside. To make the building accessible inside, voids need to be added to the building in the form of building rooms. The rooms add free space to the building interior, i.e. the OccupiedSpace contains now some UnoccupiedSpace. The free space inside the building can, in turn, contain objects that occupy space again, such as furniture or installations. In contrast, roads also occupy urban space in the city; however, this space is initially unoccupied as it is accessible by cars, pedestrian, or cyclists. Adding traffic signs or other city furniture objects to the free space results in specific sections of the road becoming occupied by these objects. Thus, one can also say that occupied spaces are mostly filled with matter; whereas, unoccupied spaces are mostly free of matter and, thus, realise free spaces.

7.4.3. Rules for Surface Orientations of OccupiedSpaces and UnoccupiedSpaces

The classification of feature types into OccupiedSpace and UnoccupiedSpace also defines the semantics of the geometries attached to the respective features. For OccupiedSpaces, the attached geometries describe volumes that are (mostly) physically occupied. For UnoccupiedSpaces, the attached geometries describe (or bound) volumes that are (mostly) physically unoccupied. This also has an impact on the required orientation of surface normals for attached thematic surfaces. For OccupiedSpaces, the normal vectors of thematic surfaces must point in the same direction as the surfaces of the outer shell of the volume. For UnoccupiedSpaces, the normal vectors of thematic surfaces must point in the opposite direction as the surfaces of the outer shell of the volume. This means that from the perspective of an observer of a city scene, the surface normals must always be directed towards the observer. In the case of OccupiedSpaces (e.g. Buildings, Furniture), the observer must be located outside the OccupiedSpace for the surface normals being directed towards the observer; whereas in the case of UnoccupiedSpaces (e.g. Rooms, Roads), the observer is typically inside the UnoccupiedSpace.

7.4.4. Levels of Detail (LOD)

CityGML differentiates four consecutive Levels of Detail (LOD 0-3), where objects become more detailed with increasing LOD regarding their geometry. CityGML datasets can - but do not have to - contain multiple geometries for each object in different LODs simultaneously. The LOD concept facilitates multi-scale modelling, i.e. having varying degrees of spatial abstractions that are appropriate for different applications or visualizations.

The classification of real-world objects into spaces and space boundaries is solely based on the semantics of these objects and not on their used geometry type, as CityGML 3.0 allows various geometrical representations for objects. A building, for instance, can be spatially represented by a 3D solid (e.g. in LOD1), but at the same time, the real-world geometry can also be abstracted by a single point, footprint or roofprint (LOD0), or by a 3D mesh (LOD3). The outer shell of the building

may also be semantically decomposed into wall, roof, and ground surfaces. [Figure 7](#) shows different representations of the same real-world building object in different geometric LODs (and appearances).

[BuildingLODs] | *images/BuildingLODs.png*

Figure 7. Representation of the same real-world building in the Levels of Detail 0-3.

The biggest changes between CityGML 3.0 and earlier versions are that

1. LOD4 was dropped, because now all feature types can have outdoor and indoor elements in LODs 0-3 (for those city objects where it makes sense like buildings, tunnels, or bridges). This means that the outside shell e.g. of a building could be spatially represented in LOD2 and the indoor elements like rooms, doors, hallways, stairs etc. in LOD1. CityGML can now be used to represent building floor plans, which are LOD0 representations of building interiors (cf. [Konde et al. 2018](#)). It is even possible to model the outside shell of a building in LOD1, while representing the interior structure in LOD2 or 3. [Figure 8](#) shows different indoor/outdoor representations of a building. Details on the changes to the CityGML LOD concept are provided in [\[\[Lowner2016\]\]](#).
2. Levels of Detail are no longer associated with the degree of semantic decomposition of city objects and refer to the spatial representations only. This means that, for example, buildings can have thematic surfaces (like WallSurface, GroundSurface) also in LODs 0 and 1 and windows and doors can be represented in all LODs 0-3. In CityGML 2.0 or earlier thematic surfaces were only allowed starting from LOD2, openings like doors and windows starting from LOD3, and interior rooms and furniture only in LOD4.
3. In CityGML 3.0 the geometry representations were moved from the thematic modules to the *Core* module and are now associated with the semantic concepts of *Spaces* and *Space Boundaries*. This led to a significant simplification of the models of the thematic modules. Since all feature types in the thematic modules are defined as subclasses of the space and space boundary classes, they automatically inherit the geometry classes and, thus, no longer require direct associations with them. This also led to a harmonized LOD representation over all CityGML feature types.
4. If new feature types are defined in Application Domain Extensions (ADEs) based on the abstract Space and Space Boundary classes from the Core module, they automatically inherit the spatial representations and the LOD concept.

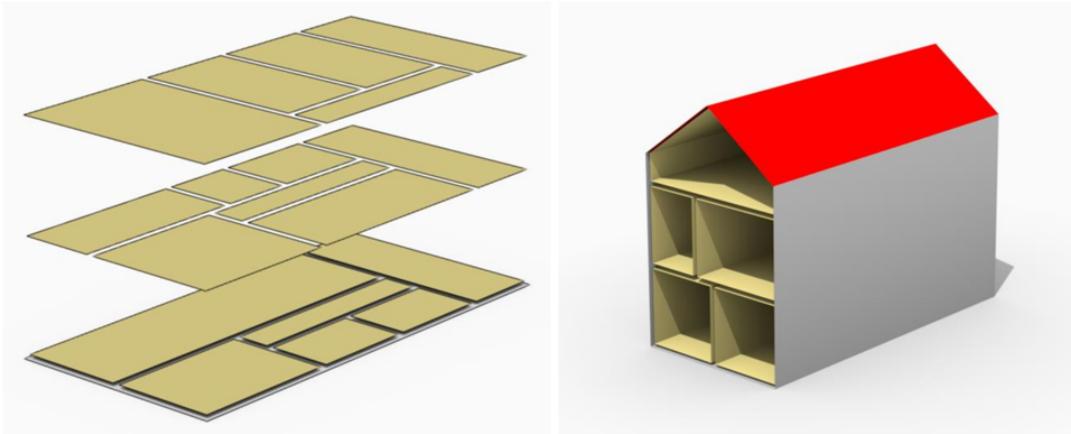


Figure 8. Floor plan representation (LOD0) of a building (left), combined LOD2 indoor and outdoor representation (right). Image adopted from [Lowner2016].

Spaces and all its subclasses like *Building*, *Room*, and *TrafficSpace* can now be spatially represented by single points in LOD0, multi-surfaces in LOD0/2/3, solids in LOD1/2/3, and multi-curves in LOD2/3. *Space Boundaries* and all its subclasses such as *WallSurface*, *LandUse*, or *Relief* can now be represented by multi-surfaces in LOD0/2/3 and as multi-curves in LOD2/3. See Section 8.2.5 for further details on the different Levels of Detail.

7.4.5. Closure Surfaces

Objects, which are spatially not represented by a volumetric geometry, must be virtually closed in order to compute their volume (e.g. pedestrian underpasses or airplane hangars). They can be sealed using a specific type of space boundary called *ClosureSurface*. These are virtual surfaces, which are taken into account, when needed to compute volumes and are neglected, when they are irrelevant or not appropriate, for example in visualisations.

The concept of *ClosureSurface* can also be employed to model the entrances of subsurface objects. Those objects like tunnels or pedestrian underpasses have to be modelled as closed solids in order to compute their volume, for example in flood simulations. The entrances to subsurface objects also have to be sealed to avoid holes in the digital terrain model (see Figure 9). However, in close-range visualisations the entrance must be treated as open. Thus, closure surfaces are an adequate way to model those entrances.

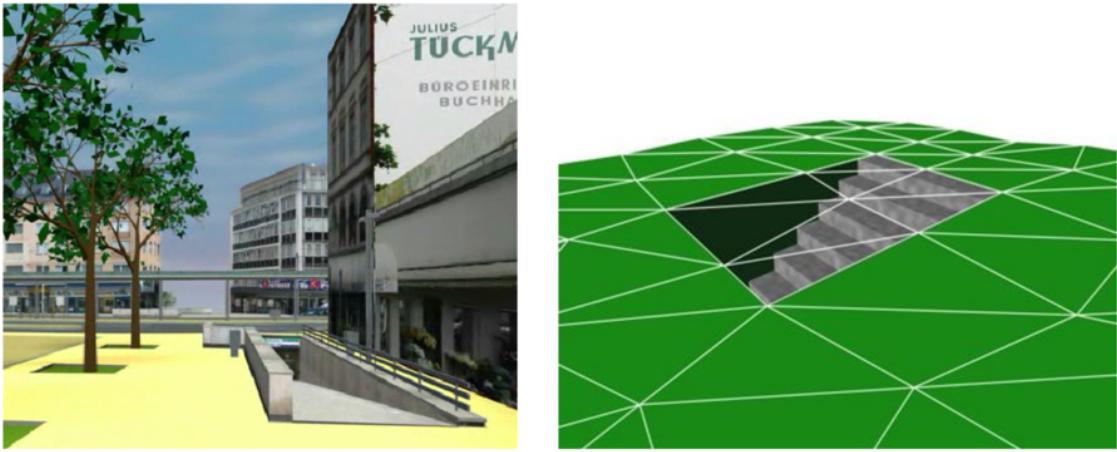


Figure 9. Closure surfaces to seal open structures. Passages are subsurface objects (left). The entrance is sealed by a virtual ClosureSurface feature, which is both part of the DTM and the subsurface object (right) (graphic: IGG Uni Bonn).

7.4.6. Terrain Intersection Curves

An important issue in city modelling is the integration of 3D objects and the terrain. Problems arise if 3D objects float over or sink into the terrain. This is particularly the case when terrains and 3D objects in different LODs are combined, when the terrain and 3D models are updated independently from each other, or when they come from different data providers [Kolbe & Gröger 2003]. To overcome this problem, the *TerrainIntersectionCurve* (TIC) of a 3D object is introduced. These curves denote the exact position, where the terrain touches the 3D object (see [Figure 10](#)). TICs can be applied to all CityGML feature types that are derived from *AbstractPhysicalSpace*, for example, buildings, bridges, tunnels, but also city furniture, vegetation, and generic city objects.

If, for example, a building has a courtyard, the TIC consists of two closed rings: one ring representing the courtyard boundary, and one which describes the building's outer boundary. This information can be used to integrate the building and a terrain by 'pulling up' or 'pulling down' the surrounding terrain to fit the *TerrainIntersectionCurve*. The digital terrain model (DTM) may be locally warped to fit the TIC. By this means, the TIC also ensures the correct positioning of textures or the matching of object textures with the DTM. Since the intersection with the terrain may differ depending on the LOD, a 3D object may have different *TerrainIntersectionCurves* for all LODs.

[*TerrainIntersectionCurves*] | *images/TerrainIntersectionCurves.png*

Figure 10. TerrainIntersectionCurve for a building (left, black) and a tunnel object (right, red). The tunnel's hollow space is sealed by a triangulated ClosureSurface (graphic: IGG Uni Bonn).

7.4.7. Coherent Semantical-Geometrical Modelling

An important design principle for CityGML is the coherent modelling of semantic objects and their spatial representations. At the semantic level, real-world entities are represented by features, such as buildings, walls, windows, or rooms. The description also includes attributes, relations and aggregation hierarchies (part-whole-relations) between features. Thus the part-of-relationship between features can be derived at the semantic level only, without considering geometry. However, at the spatial level, geometry objects are assigned to features representing their spatial location, shape, and extent. So the model consists of two hierarchies: the semantic and the geometrical in which the corresponding objects are linked by relationships (cf. [Stadler & Kolbe](#)

[2007](#)). The advantage of this approach is that it can be navigated in both hierarchies and between both hierarchies arbitrarily, for answering thematic and/or geometrical queries or performing analyses.

If both hierarchies exist for a specific object, they must be coherent (i.e. it must be ensured that they match and fit together). For example, if a building is semantically decomposed into wall surfaces, roof surfaces etc., the polygons representing these thematic surfaces (in a specific LOD) must be part of the solid geometry representing the entire building (for the same LOD).

7.5. Appearances

Information about the appearance of surfaces, i.e. observable properties of the surface, is considered an integral part of virtual 3D city and landscape models in addition to semantics and geometry. Appearance relates to any surface-based theme, e.g. infrared radiation or noise pollution, not just visual properties like RGB texture images. Consequently, data provided by appearances can be used as input for both, presentation of and analysis in virtual 3D city models.

CityGML supports feature appearances for an arbitrary number of themes per city model. Each LOD of a feature can have an individual appearance. Appearances can represent – among others – textures and georeferenced textures. CityGML’s appearance model is packaged within the Appearance module (cf. [Section 8.3](#)).

7.6. Modelling Dynamic Data

In general, city objects can have properties related to their geometry, topology, semantics, and appearance. All of these properties may change over time. For example, a construction event leads to the change in geometry of a building (i.e. addition of a new building floor or demolition of an existing door). The geometry of an object can be further classified according to its shape, location, and extent, which can also change over time. A moving car object involves changing only the location of the car object; however, a flood incident involves variations in the location and shape of water. There might be other properties, which change with respect to thematic data of city objects, e.g. hourly variations in energy or gas consumption of a building or changing the building usage from residential to commercial. Some properties involve changes in appearances over a time period, such as building textures changing over years or traffic cameras recording videos of moving traffic over definite intervals. 3D city models also represent interrelationships between objects and relations may change over time as well. Hence, it is important to consider that the representation of time-varying data is required to be associated with these different properties. A detailed discussion on the requirements of city model applications regarding the support of dynamic data is given in [[Chaturvedi & Kolbe 2019](#)].

CityGML 3.0 introduces two concepts to manage dynamic, i.e. time-dependent, properties of city models. The *Versioning* module manages changes that are slower in nature, e.g. (1) the history or evolution of cities such as construction or demolition of buildings, and (2) managing multiple versions of the city models. The *Dynamizer* module manages higher-frequent or dynamic variations of object properties, e.g. variations of (1) thematic attributes such as changes of physical quantities (energy demands, temperature, solar irradiation levels), (2) spatial properties such as change of a feature’s geometry, with respect to shape and location (moving objects), and (3) real-time sensor observations. The Dynamizer module allows establishing explicit links from city objects to sensors

and sensor data services.

7.6.1. Versioning and Histories

The bitemporal timestamps of all CityGML feature types as described in [Section 7.2.1](#) allow representing the evolution of the real city and its model over time. The new *Versioning* module extends this by the possibility to represent multiple, concurrent versions of the city model. For that purpose, the module defines two new feature types: 1) *Version*, which can be used to explicitly define named states of the 3D city model and denote all the specific versions of objects belonging to such states. 2) *VersionTransition*, which allows to explicitly link different versions of the 3D city model by describing the reason of change and the modifications applied. Details on the versioning concept are given in [[Chaturvedi et al. 2015](#)].

This approach not only facilitates the explicit representation of different city model versions, but also allows distinguishing and referring to different versions of city objects in an interoperable exchange format. All object versions could be stored and exchanged within a single dataset. Software systems could use such a dataset to visualize and work with the different versions simultaneously. The conceptual model also takes into account the management of multiple histories or multiple interpretations of the past of a city, which is required when looking at historical city developments and for archaeological applications. In addition, the Versioning module supports collaborative work, because it provides all functionalities to represent a tree of workspaces as version control systems like *git* or *SVN*. The Versioning module handles versions and version transitions as feature types, which allows the version management to be completely handled using the standard OGC Web Feature Service [[Vrenatos 2010](#)]. No extension of this standard is required to manage the versioning of city models.

7.6.2. Dynamizers: Using Time-Series Data for Object Attributes

The new Dynamizer module improves the usability of CityGML for different kinds of simulations as well as to facilitate the integration of devices from the Internet-of-Things (IoT) like sensors with 3D city models. Both, simulations and sensors provide dynamic variations of some measured or simulated properties like, for example, the electricity consumption of a building or the traffic density within a road segment. The variations of the value are typically represented using time-series data. The data sources of the time-series data could be either sensor observations (e.g. from a smart meter), pre-recorded load profiles (e.g. from an energy company), or the results of some simulation run.

[Dynamizers] | *images/Dynamizers.png*

Figure 11. Dynamizers link timeseries data coming from different sources to specific properties of individual city objects.

As shown in [Figure 11](#), Dynamizers serve three main purposes:

1. Dynamizer is a data structure to represent dynamic values in different and generic ways. Such dynamic values may be given by (1) tabulation of time/value pairs using its *AtomicTimeseries* class, (2) patterns of time/value pairs based on statistical rules using its *CompositeTimeseries* class, and (3) retrieving observations directly from external sensor/IoT services using its *SensorConnection* class. The values can be obtained from sensor services like the OGC Sensor Observation Service or OGC SensorThings API, simulation specific databases, and also external

files such as CSV or Excel sheets.

2. Dynamizer delivers a method to enhance static city models by dynamic property values. It references a specific property (e.g. spatial, thematic or appearance properties) of a specific object within a 3D city model providing dynamic values overriding the static value of the referenced object attribute.
3. Dynamizer objects establish explicit links between sensor/observation data and the respective properties of city model objects that are measured by them. By making such explicit links with city object properties, the semantics of sensor data become implicitly defined by the city model.

Dynamizers are used to inject dynamic variations of city object properties into an otherwise static representation. The advantage in following such approach is that it allows only selected properties of city models to be made dynamic. If an application does not support dynamic data, it simply does not allow/include these special types of features.

Dynamizers have already been implemented as an Application Domain Extension (ADE) for CityGML 2.0 and were employed in the OGC Future City Pilot Phase 1. More details about Dynamizers are given in [[Chaturvedi & Kolbe 2017](#)].

7.7. Extending CityGML

CityGML has been designed as a universal topographic information model that defines object types and attributes which are useful for a broad range of applications. In practical applications, the objects within specific 3D city models will most likely contain attributes which are not explicitly modelled in CityGML. Moreover, there might be 3D objects which are not covered by the thematic classes of CityGML. CityGML provides three different concepts to support the exchange of such data:

1. [Generic objects and attributes](#),
2. [Application Domain Extensions](#), and
3. [Code lists](#).

The concept of generic objects and attributes allows for the extension of CityGML applications during runtime, i.e. any city object may be augmented by additional attributes and relations, whose names, data types, and values can be provided by a running application without requiring to extend the CityGML conceptual schema and the respective encodings. Similarly, features not represented by the predefined thematic classes of the CityGML conceptual model may be modelled and exchanged using generic objects. The generic extensions of CityGML are provided by the *Generics* module (cf. [Section 8.7](#)).

Application Domain Extensions (ADE) specify additions to the CityGML conceptual model. Such additions comprise the introduction of new properties to existing CityGML feature types like e.g. the energy demand of a building or the definition of additional feature types. The difference between ADEs and generic objects and attributes is, that an ADE has to be defined in an extra conceptual schema (provided in UML) with its own namespace. Encodings have to be extended accordingly. The advantage of this approach is that the extension is formally specified. Extended CityGML datasets can be validated against the CityGML and the respective ADE schema. ADEs can be defined (and even standardised) by information communities which are interested in specific application

fields. More than one ADE can be used simultaneously in the same dataset. Examples for popular ADEs are the Utility Network ADE [Becker et al. 2011; Kutzner et al. 2018] and the Energy ADE [Nouvel et al. 2015; Agugiaro et al. 2018]. A comprehensive overview of CityGML ADEs is given in [Biljecki et al. 2018]. Further details on ADEs are given in [Chapter 10](#).

CityGML can also be extended with regard to the allowed values specified in code lists. Many attributes of CityGML types use a code list as data type such as, for instance, the attributes *class*, *usage*, and *function* of city objects. A code list defines a value domain including a code for each permissible value. In contrast to fixed enumerations, modifications and extensions to the value domain become possible with code lists. The values for all code lists in CityGML have to be defined externally, for example, by adopting classifications from global, national, or industrial standards.

Additional information about the extension features of CityGML can be found in the [CityGML 3.0 Users Guide](#).

Chapter 8. CityGML UML Model

The CityGML UML model is the normative definition of the CityGML Conceptual Model. The tables and figures in this section were software generated from the UML model. As such, this section provides a normative representation of the CityGML Conceptual Model.

An alternate representation can be found in the Data Dictionary in [Chapter 9](#).

8.1. Structural Overview of Requirements Classes

The Requirements Classes for this standard are structured as UML Packages as illustrated in [Figure 12](#). Each Requirements Class is specified in detail in their respective subsections. These subsections include a UML diagram, data dictionary, and the applicable requirements.

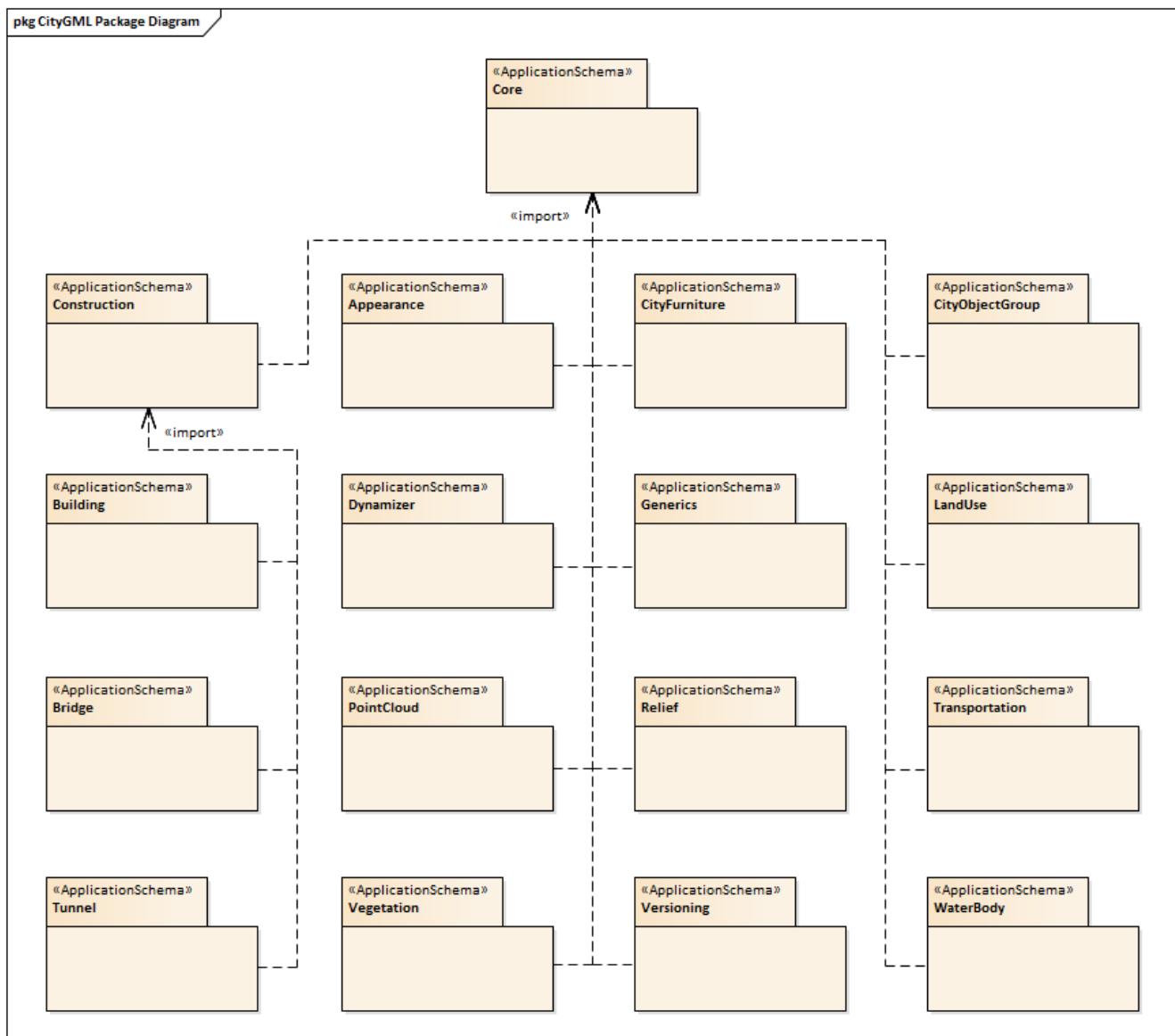


Figure 12. CityGML UML Packages

8.2. Core

Requirements Class	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-core	
Target type	Implementation Specification
Dependency	ISO 19103:2015
Dependency	ISO 19107:2003
Dependency	ISO 19109:2015
Dependency	ISO 19111:2019
Dependency	ISO 19123:2005
Dependency	OASIS xAL v3.0

The Core module defines the basic concepts and components of city models. This rather large body of work has been divided into seven sections. These sections build on each other from the fundamental principles specified by the ISO up through the full CityGML model. These sections are summarized in [Table 3](#).

Table 3. CityGML Core Sections

The Use of ISO Standards	Describes the use of the ISO 19100 series of International Standards to provide a foundation to the CityGML model.
City Models and City Objects	Defines the basic building blocks of the CityGML model.
Space Concept	Defines the concepts of space as used in the CityGML model.
Geometry and LOD	Defines the geometry and Levels Of Detail concepts.
CityGML Core Model	Presents the complete Core model.
Types, Enumerations, and Codelist	Defines the little things which make this model work.

8.2.1. Requirements

The CityGML Core defines technology-agnostic concepts. These concepts are then realized in technology-specific Implementation Specifications. The following requirements govern the creation of those Implementation Specifications.

Requirement 1	/req/core/classes
For each UML class defined or referenced in the Core Package:	

A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and mutiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

While the CityGML Conceptual Model builds on ISO Standards, there are some restrictions on the use of those standards.

Requirement 2 /req/Core/isorestrictions	
ISO classes used in the CityGML Conceptual Model are subject to the following restrictions:	
A	Classes derived from the GM_Solid class (ISO 19107) SHALL NOT include interior boundaries. (The interior association on the GM_SolidBoundary shall not be defined)

An implementing technology may not be able to support all of the concepts defined in the CityGML Conceptual Model. Alternately, some concepts from the Conceptual Model may be inappropriate for the application domain for which the Implementation Specification was developed. In those cases, elements of the Conceptual Model may be mapped to null elements in the Implementation Specification.

Permission 1 /per/Core/classes	
For each UML class defined or referenced in CityGML Conceptual Model:	
A	An Implementation Specification MAY represent that class as a null class with no attributes, associations, or definition.
B	An Implementation Specification MAY represent an association of the UML class with a null association.
C	An Implementation Specification MAY represent an attribute of the UML class with a null attribute.

D	Whenever a null element is used to represent a concept from the Conceptual Model, the Implementation Specification SHOULD document that mapping and provide an explanation for why that concept was not implemented.
---	--

Table 4 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Core module:

Table 4. Core space classes and their allowed thematic surface boundaries

Space class	Allowed space boundaries
AbstractLogicalSpace	<ul style="list-style-type: none"> • Core::AbstractSpaceBoundary and the subclasses: Core::AbstractThematicSurface, Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
AbstractOccupiedSpace	<ul style="list-style-type: none"> • Core::AbstractSpaceBoundary and the subclasses: Core::AbstractThematicSurface, Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
AbstractPhysicalSpace	<ul style="list-style-type: none"> • Core::AbstractSpaceBoundary and the subclasses: Core::AbstractThematicSurface, Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
AbstractSpace	<ul style="list-style-type: none"> • Core::AbstractSpaceBoundary and the subclasses: Core::AbstractThematicSurface, Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
AbstractUnoccupiedSpace	<ul style="list-style-type: none"> • Core::AbstractSpaceBoundary and the subclasses: Core::AbstractThematicSurface, Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs

Surface boundaries are constrained by the following requirement:

Requirement 3	/req/core/boundaries
Table 4 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Core module. An Implementation Specification SHALL NOT specify boundaries except as specified in Table 4	

The use of extension capabilities by Core elements is constrained by the following requirement:

Requirement 4	/req/Core/ade/use
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

8.2.2. ISO Dependencies

CityGML builds on the ISO 19100 family of standards. The applicable standards are identified in the diagram in Figure 13. Data dictionaries are included for all of the ISO-defined classes explicitly referenced in the CityGML UML model. These data dictionaries are provided for the convenience of the user. The ISO standards are the normative source.

[ISOandOASISstandardsinCityGML] | *figures/Core/ISOandOASISstandardsinCityGML.png*

Figure 13. Use of ISO Standards in CityGML

The ISO classes explicitly used in the CityGML UML model are introduced in Table 5. More details about these classes can be found in the Data Dictionary in Chapter 9.

Table 5. ISO Classes used in CityGML

Class Name	Description
AnyFeature	A generalization of all feature types
CV_DiscreteGridPointC overage	A coverage that returns the same feature attribute values for every direct position within any object in its domain.
Direct Position	The coordinates for a position within some coordinate reference system.
GM_Object	root class of the geometric object taxonomy.
GM_MultiCurve	An aggregate class containing only instances of GM_OrientableCurve.
GM_MultiPoint	An aggregate class containing only points.
GM_MultiSurface	An aggregate class containing only instances of GM_OrientableSurface.
GM_Point	The basic data type for a geometric object consisting of one and only one point.
GM_Solid	The basis for 3-dimensional geometry. The extent of a solid is defined by the boundary surfaces.
GM_Surface	The basis for 2-dimensional geometry.
GM_Tin	A GM_TriangulatedSurface which uses the Delaunay or similar algorithm.
GM_TriangulatedSurfac e	A GM_PolyhedralSurface that is composed only of triangles

SC_CRS	Coordinate reference system which is usually single but may be compound.
TM_Position	A union class that consists of one of the data types listed as its attributes.

8.2.3. City Models and City Objects

City models are virtual representations of real-world cities and landscapes. A city model aggregates different types of objects, which can be city objects, appearances, different versions of the city model, transitions between different versions of the city model, and feature objects. All objects defined in CityGML are features with lifespan. This allows the optional specification of the real-world and database times for the existence of each feature, as is required by the Versioning module (cf. [Section 8.13](#)). Features that define thematic concepts related to cities and landscapes, such as building, bridge, water body, or land use, are referred to as city objects. All city objects define properties that describe the objects in more detail. These static properties can be overridden with time-varying data through Dynamizers (cf. [Section 8.6](#)).

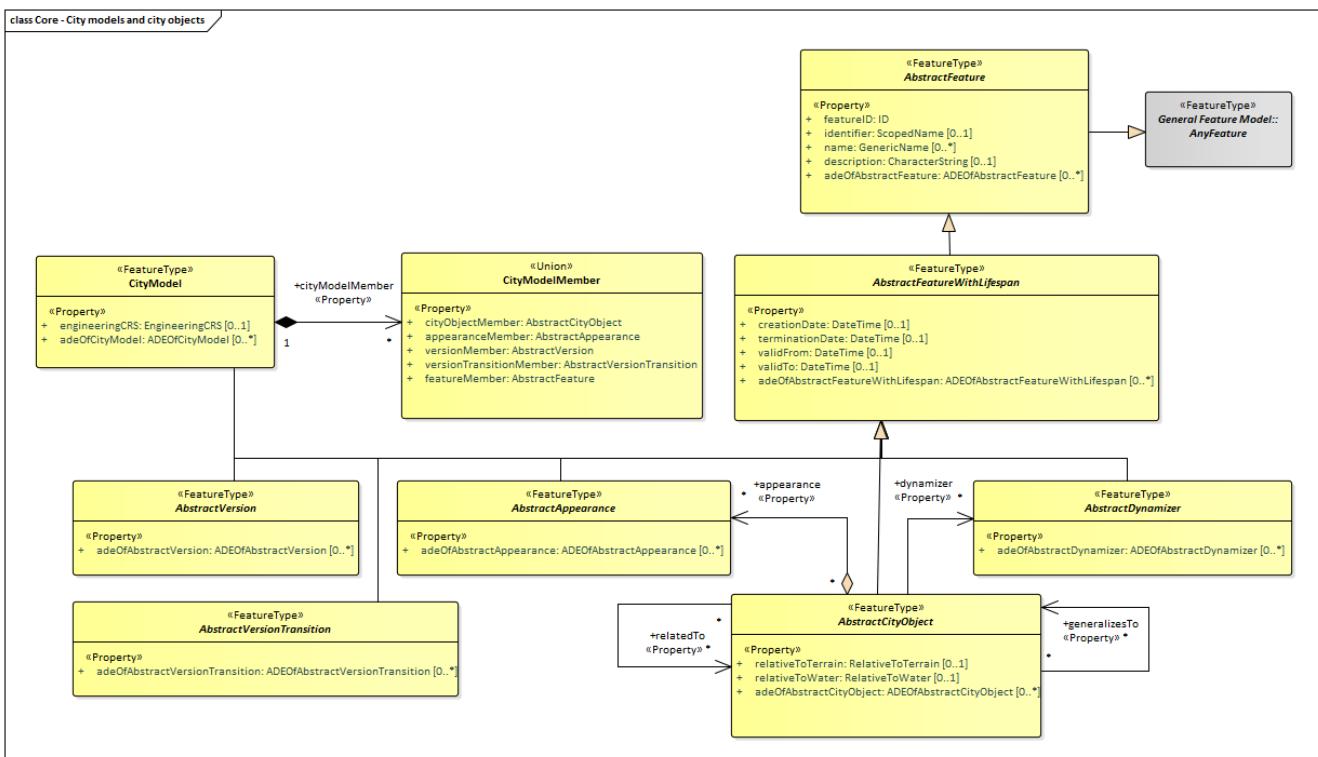


Figure 14. UML City Models and City Objects

The City Model and City Object classes defined in the CityGML UML model are introduced in [Table 6](#). More details about these classes can be found in the Data Dictionary in [Chapter 9](#).

Table 6. City Model and City Object classes used in Core

Class	Description
AbstractAppearance «FeatureType»	AbstractAppearance is the abstract superclass to represent any kind of appearance objects.
AbstractCityObject «FeatureType»	AbstractCityObject is the abstract superclass of all thematic classes within the CityGML conceptual model.

AbstractDynamizer «FeatureType»	AbstractDynamizer is the abstract superclass to represent Dynamizer objects.
AbstractFeature «FeatureType»	AbstractFeature is the abstract superclass of all feature types within the CityGML conceptual model.
AbstractFeatureWithLifespan «FeatureType»	AbstractFeatureWithLifespan is the base class for all CityGML features. It allows the optional specification of the real-world and database times for the existence of each feature.
AbstractVersion «FeatureType»	AbstractVersion is the abstract superclass to represent Version objects.
AbstractVersionTransition «FeatureType»	AbstractVersionTransition is the abstract superclass to represent VersionTransition objects.
CityModel «FeatureType»	CityModel is the container for all objects belonging to a city model.

8.2.4. Space Concept

All city objects are differentiated into spaces and space boundaries. Spaces are entities of volumetric extent in the real world. Buildings, water bodies, trees, rooms, and traffic spaces, for instance, have a volumetric extent. Spaces can be classified into physical spaces and logical spaces. Physical spaces, in turn, can be further classified into occupied spaces and unoccupied spaces.

Space boundaries, in contrast, are entities with areal extent in the real world. Space boundaries can be differentiated into different types of thematic surfaces, such as wall surfaces and roof surfaces.

A detailed introduction to the Space concept can be found in [Section 7.4](#). In particular, the classification into OccupiedSpace and UnoccupiedSpace might not always be apparent at first sight. Carports, for instance, represent an OccupiedSpace, although they are not closed and most of the space is free of matter, see [Figure 15](#). Since a carport is a roofed, immovable structure with the purpose of providing shelter to objects (i.e. cars), carports are frequently represented as buildings in cadastres. Thus, also in CityGML, a carport should be modelled as an instance of the class Building. Since Building is transitively a subclass of OccupiedSpace, a carport is an OccupiedSpace as well. However, only in LOD1, the entire volumetric region covered by the carport would be considered as physically occupied. In LOD1, the occupied space is defined by the entire carport solid (unless a room would be defined in LOD1 that would model the unoccupied part below the roof); whereas in LOD2 and LOD3, the solids represent more realistically the really physically occupied space of the carport. In addition, for all OccupiedSpaces, the normal vectors of the thematic surfaces like the RoofSurface need to point away from the solids, i.e. consistent with the solid geometry.

[carport] | *images/carport.png*

Figure 15. Representation of a carport as OccupiedSpace in different LODs. The red boxes represent solids, the green area represents a surface. In addition, the normal vectors of the roof solid (in red) and the roof surface (in green) are shown.

In contrast, a room is a physically unoccupied space. In CityGML, a room is represented by the class

BuildingRoom that is a subclass of UnoccupiedSpace. In LOD1, the entire room solid would be considered as unoccupied space, which can contain furniture and installations, though, as is shown in [Figure 16](#). In LOD2 and 3, the solid represents more realistically the really physically unoccupied space of the room (possibly somewhat generalised as indicated in the figure). For all UnoccupiedSpaces, the normal vectors of the bounding thematic surfaces like the InteriorWallSurface need to point inside the object, i.e. opposite to the solid geometry.

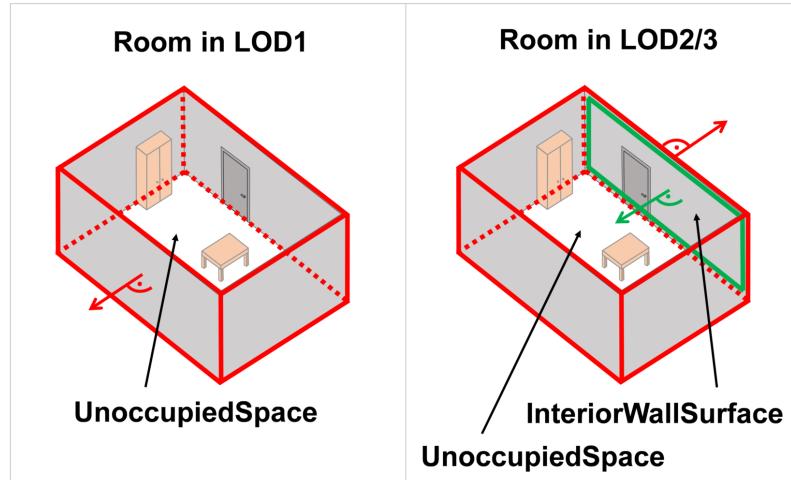


Figure 16. Representation of a room as UnoccupiedSpace in different LODs. The red boxes represent solids, the green area represents a surface. In addition, the normal vectors of the room solid (in red) and the wall surface (in green) are shown.

The UML diagram of the Space concept classes is depicted in [Figure 17](#).

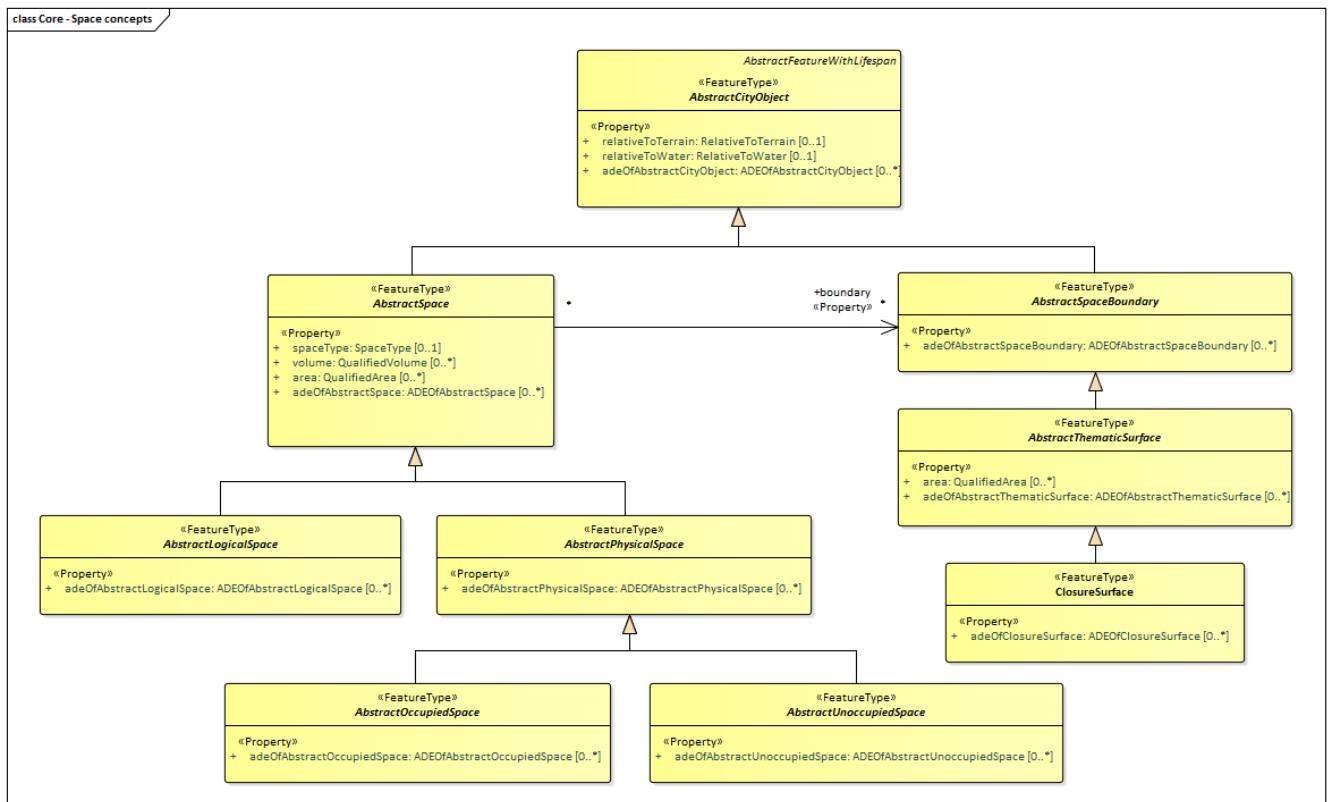


Figure 17. UML Space Concepts

The Space Concept classes defined in the CityGML UML model are introduced in [Table 7](#). More details about these classes can be found in the Data Dictionary in [Chapter 9](#).

Table 7. Space Classes used in Core

Class	Description
<code>AbstractLogicalSpace</code> «FeatureType»	AbstractLogicalSpace is the abstract superclass for all types of logical spaces. Logical space refers to spaces that are not bounded by physical surfaces but are defined according to thematic considerations.
<code>AbstractOccupiedSpace</code> «FeatureType»	AbstractOccupiedSpace is the abstract superclass for all types of physically occupied spaces. Occupied space refers to spaces that are partially or entirely filled with matter.
<code>AbstractPhysicalSpace</code> «FeatureType»	AbstractPhysicalSpace is the abstract superclass for all types of physical spaces. Physical space refers to spaces that are fully or partially bounded by physical objects.
<code>AbstractSpace</code> «FeatureType»	AbstractSpace is the abstract superclass for all types of spaces. A space is an entity of volumetric extent in the real world.
<code>AbstractSpaceBoundary</code> «FeatureType»	AbstractSpaceBoundary is the abstract superclass for all types of space boundaries. A space boundary is an entity with areal extent in the real world. Space boundaries are objects that bound a Space. They also realize the contact between adjacent spaces.
<code>AbstractThematicSurface</code> «FeatureType»	AbstractThematicSurface is the abstract superclass for all types of thematic surfaces.
<code>AbstractUnoccupiedSpace</code> «FeatureType»	AbstractUnoccupiedSpace is the abstract superclass for all types of physically unoccupied spaces. Unoccupied space refers to spaces that are entirely or mostly free of matter.
<code>ClosureSurface</code> «FeatureType»	ClosureSurface is a special type of thematic surface used to close holes in volumetric objects. Closure surfaces are virtual (non-physical) surfaces.

8.2.5. Geometry and LOD

Spaces and space boundaries can have various geometry representations depending on the Levels of Detail (LOD). Spaces can be spatially represented as single points in LOD0, multi-surfaces in LOD0/2/3, solids in LOD1/2/3, and multi-curves in LOD2/3. Space boundaries can be represented as multi-surfaces in LOD0/2/3 and as multi-curves in LOD2/3. All Levels of Detail allow for the representation of the interior of city objects.

The different Levels of Detail are defined in the following way:

- LOD 0: volumetric real-world objects (Spaces) can be spatially represented by a single point, by a set of curves, or by a set of surfaces. Areal real-world objects (Space Boundaries) can be spatially represented in LOD0 by a set of curves or a set of surfaces. LOD0 surface representations are typically the result of a projection of the shape of a volumetric object onto a plane parallel to the ground, hence, representing a footprint (e.g. a building footprint or a floor plan of the rooms inside a building). LOD0 curve representations are either the result of a projection of the shape of a vertical surface (e.g. a wall surface) onto a grounding plane or the skeleton of a volumetric shape of longitudinal extent like a road or river segment.

- LOD 1: volumetric real-world objects (Spaces) are spatially represented by a vertical extrusion solid, i.e. a solid created from a horizontal footprint by vertical extrusion. Areal real-world objects (Space Boundaries) can be spatially represented in LOD1 by a set of horizontal or vertical surfaces.
- LOD 2: volumetric real-world objects (Spaces) can be spatially represented by a set of curves, a set of surfaces, or a single solid geometry. Areal real-world objects (Space Boundaries) can be spatially represented in LOD2 by a set of surfaces. The shape of the real-world object is generalized in LOD2 and smaller details (e.g. bulges, dents, sills, but also structures like e.g. balconies or dormers of buildings) are typically neglected. LOD2 curve representations are skeletons of volumetric shapes of longitudinal extent like an antenna or a chimney.
- LOD 3: volumetric real-world objects (Spaces) can be spatially represented by a set of curves, a set of surfaces, or a single solid geometry. Areal real-world objects (Space Boundaries) can be spatially represented in LOD3 by a set of surfaces. LOD3 is the highest level of detail and respective geometries include all available shape details.

In addition, the geometry can also be represented implicitly, i.e. the shape is stored only once as a prototypical geometry, which then is re-used or referenced, wherever the corresponding feature occurs in the 3D city model.

The thematic classes, such as building, tunnel, road, land use, water body, or city furniture are defined as subclasses of the space and space boundary classes within the thematic modules. Since all city objects in the thematic modules represent subclasses of the space and space boundary classes, they automatically inherit the geometries defined in the Core module.

The UML diagram of the Geometry and LoD concept classes is depicted in Figure 18.

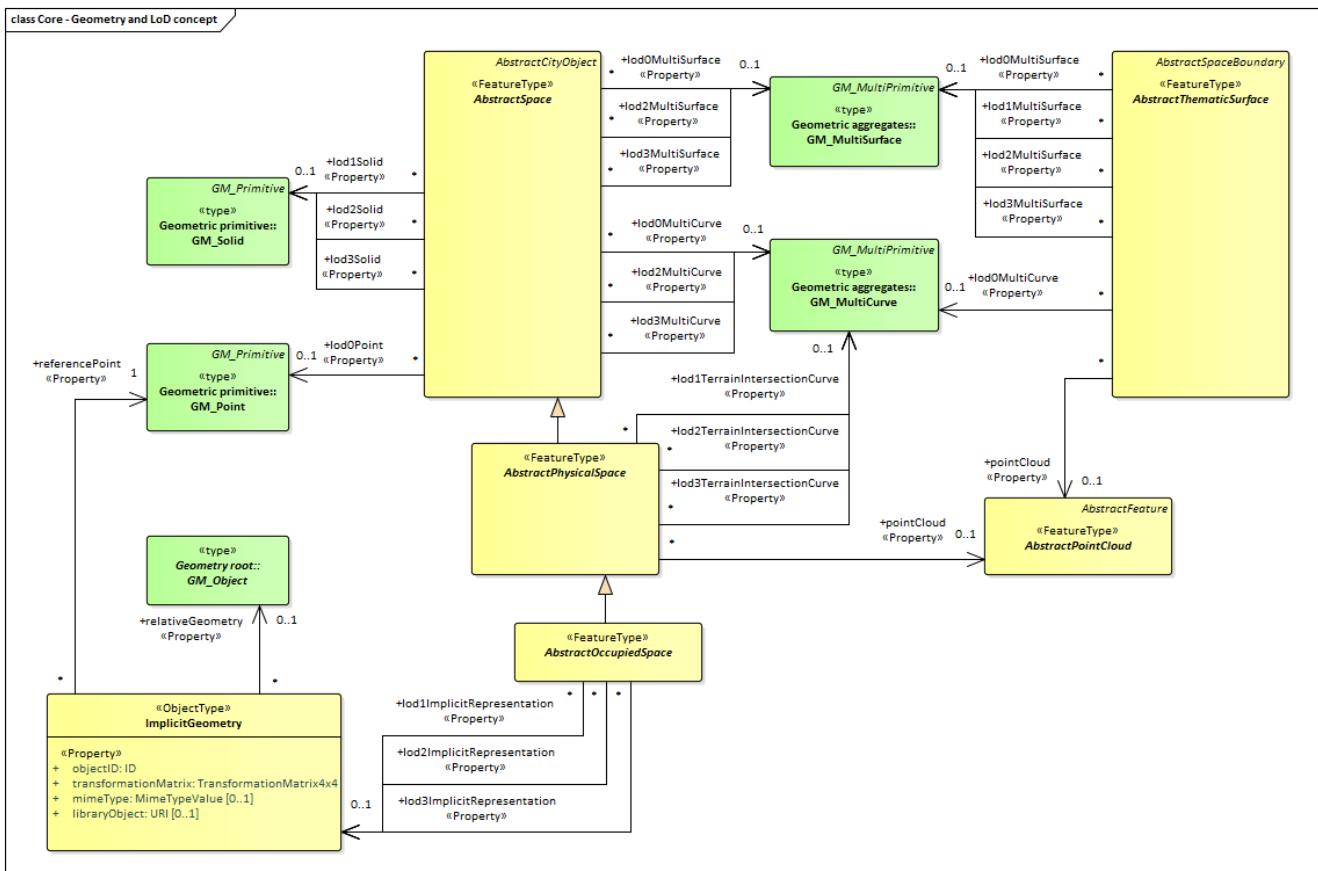


Figure 18. UML Geometry and LoD Concepts

The Geometry and LOD Concept classes defined in the CityGML UML model are introduced in [Table 8](#). More details about these classes can be found in the Data Dictionary in [Chapter 9](#).

Of particular note is the Implicit Geometry concept. Many of the objects encountered in a city landscape have the same geometry. How many types of street lamps can there be? An Implicit Geometry captures that geometry once, and re-uses that one geometry for all similar street lamp objects.

Table 8. Geometry Classes used in Core

Class	Description
AbstractOccupiedSpace «FeatureType»	AbstractOccupiedSpace is the abstract superclass for all types of physically occupied spaces. Occupied space refers to spaces that are partially or entirely filled with matter.
AbstractPhysicalSpace «FeatureType»	AbstractPhysicalSpace is the abstract superclass for all types of physical spaces. Physical space refers to spaces that are fully or partially bounded by physical objects.
AbstractPointCloud «FeatureType»	AbstractPointCloud is the abstract superclass to represent PointCloud objects.
AbstractSpace «FeatureType»	AbstractSpace is the abstract superclass for all types of spaces. A space is an entity of volumetric extent in the real world.
AbstractThematicSurface «FeatureType»	AbstractThematicSurface is the abstract superclass for all types of thematic surfaces.
ImplicitGeometry «ObjectType»	ImplicitGeometry is a geometry representation where the shape is stored only once as a prototypical geometry, for example a tree or other vegetation object, a traffic light or a traffic sign. This prototypic geometry object can be re-used or referenced many times, wherever the corresponding feature occurs in the 3D city model.

8.2.6. CityGML Core Model

The [City Model and City Object](#) classes, the [Space Concept](#) classes, and the [Geometry and LOD](#) classes define the majority of the CityGML Core module. In addition to these concepts, the Core module also specifies that city objects can have relations to other city objects and that they can have address information. All other modules defined in the CityGML model refer to the Core module.

The UML diagram of the complete Core module is depicted in [Figure 19](#).

[Core Overview] | *figures/Core/Core-Overview.png*

Figure 19. UML diagram of CityGML's core module.

[Table 6](#), [Table 7](#), and [Table 8](#) introduce already most of the classes of the CityGML Core module. The additional classes required to fill out this section are introduced in [Table 9](#). More details about these classes can be found in the Data Dictionary in [Chapter 9](#).

Table 9. Additional Classes used in Core

Class	Description
Address «FeatureType»	Address represents an address of a city object.
CityObjectRelation «ObjectType»	CityObjectRelation represents a specific relation from the city object in which it is included to another city object.

8.2.7. Data types, Enumerations, and Code lists

While FeatureTypes capture the real-world concepts on the CityGML Conceptual Model, they would be incomplete without the additional concepts from which they are made. These supporting constructs are illustrated in the following figures.

The ADE data types provided for the Core module are illustrated in the figure [Figure 20](#).

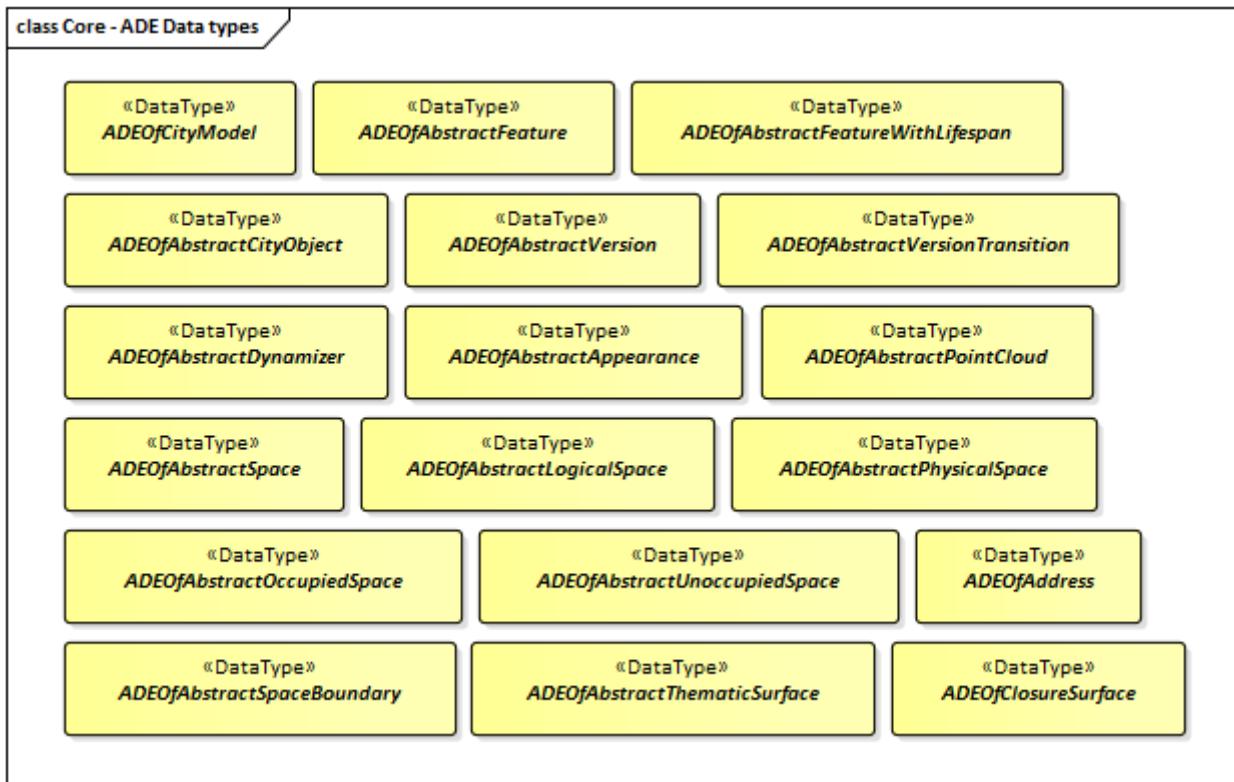


Figure 20. ADE classes of the CityGML Core module.

The Basic Types, Enumerations, and Code Lists provided for the Core module are illustrated in the figure [Figure 21](#).

[Core Basic Types Enumerations Codelists] | [figures/Core/Core-](#)

Basic_Types_Enumerations_Codelists.png

Figure 21. Basic Types, Enumerations, and Codelists from the CityGML Core module.

These supporting constructs are defined in the following tables.

Table 10. Data Types used in Core

Name	Description
AbstractGenericAttribute «DataType»	AbstractGenericAttribute is the abstract superclass for all types of generic attributes.
ADEOfAbstractAppearance «DataType»	ADEOfAbstractAppearance acts as a hook to define properties within an ADE that are to be added to AbstractAppearance.
ADEOfAbstractCityObject «DataType»	ADEOfAbstractCityObject acts as a hook to define properties within an ADE that are to be added to AbstractCityObject.
ADEOfAbstractDynamizer «DataType»	ADEOfAbstractDynamizer acts as a hook to define properties within an ADE that are to be added to AbstractDynamizer.
ADEOfAbstractFeature «DataType»	ADEOfAbstractFeature acts as a hook to define properties within an ADE that are to be added to AbstractFeature.
ADEOfAbstractFeatureWithLifespan «DataType»	ADEOfAbstractFeatureWithLifespan acts as a hook to define properties within an ADE that are to be added to AbstractFeatureWithLifespan.
ADEOfAbstractLogicalSpace «DataType»	ADEOfAbstractLogicalSpace acts as a hook to define properties within an ADE that are to be added to AbstractLogicalSpace.
ADEOfAbstractOccupiedSpace «DataType»	ADEOfAbstractOccupiedSpace acts as a hook to define properties within an ADE that are to be added to AbstractOccupiedSpace.
ADEOfAbstractPhysicalSpace «DataType»	ADEOfAbstractPhysicalSpace acts as a hook to define properties within an ADE that are to be added to AbstractPhysicalSpace.
ADEOfAbstractPointCloud «DataType»	ADEOfAbstractPointCloud acts as a hook to define properties within an ADE that are to be added to AbstractPointCloud.
ADEOfAbstractSpace «DataType»	ADEOfAbstractSpace acts as a hook to define properties within an ADE that are to be added to AbstractSpace.
ADEOfAbstractSpaceBoundary «DataType»	ADEOfAbstractSpaceBoundary acts as a hook to define properties within an ADE that are to be added to AbstractSpaceBoundary.

ADEOfAbstractThematicSurface «DataType»	ADEOfAbstractThematicSurface acts as a hook to define properties within an ADE that are to be added to AbstractThematicSurface.
ADEOfAbstractUnoccupiedSpace «DataType»	ADEOfAbstractUnoccupiedSpace acts as a hook to define properties within an ADE that are to be added to AbstractUnoccupiedSpace.
ADEOfAbstractVersion «DataType»	ADEOfAbstractVersion acts as a hook to define properties within an ADE that are to be added to AbstractVersion.
ADEOfAbstractVersionTransition «DataType»	ADEOfAbstractVersionTransition acts as a hook to define properties within an ADE that are to be added to AbstractVersionTransition.
ADEOfAddress «DataType»	ADEOfAddress acts as a hook to define properties within an ADE that are to be added to an Address.
ADEOfCityModel «DataType»	ADEOfCityModel acts as a hook to define properties within an ADE that are to be added to a CityModel.
ADEOfClosureSurface «DataType»	ADEOfClosureSurface acts as a hook to define properties within an ADE that are to be added to a ClosureSurface.
ExternalReference «DataType»	ExternalReference is a reference to a corresponding object in another information system, for example in the German cadastre (ALKIS), the German topographic information system (ATKIS), or the OS UK MasterMap®.
Occupancy «DataType»	Occupancy is an application-dependent indication of what is contained by a feature.
QualifiedArea «DataType»	QualifiedArea is an application-dependent measure of the area of a space or of a thematic surface.
QualifiedVolume «DataType»	QualifiedVolume is an application-dependent measure of the volume of a space.
XALAddress «DataType»	XALAddress represents address details according to the OASIS xAL standard.

Table 11. Primitive Data Types used in Core

Name	Description
Code «BasicType»	Code is a basic type for a String-based term, keyword, or name that can additionally have a code space.
DoubleBetween0and1 «BasicType»	DoubleBetween0and1 is a basic type for values, which are greater or equal than 0 and less or equal than 1. The type is used for color encoding, for example.
DoubleBetween0and1List «BasicType»	DoubleBetween0and1List is a basic type that represents a list of double values greater or equal than 0 and less or equal than 1. The type is used for color encoding, for example.

DoubleList «BasicType»	DoubleList is an ordered sequence of double values.
DoubleOrNilReasonList «BasicType»	DoubleOrNilReasonList is a basic type that represents a list of double values and/or nil reasons.
ID «BasicType»	ID is a basic type that represents a unique identifier.
IntegerBetween0and3 «BasicType»	IntegerBetween0and3 is a basic type for integer values, which are greater or equal than 0 and less or equal than 3. The type is used for encoding the LOD number.
MeasureOrNilReasonList «BasicType»	MeasureOrNilReasonList is a basic type that represents a list of double values and/or nil reasons together with a unit of measurement.
TransformationMatrix2x2 «BasicType»	TransformationMatrix2x2 is a 2 by 2 matrix represented as a list of four double values in row major order.
TransformationMatrix3x4 «BasicType»	TransformationMatrix3x4 is a 3 by 4 matrix represented as a list of twelve double values in row major order.
TransformationMatrix4x4 «BasicType»	TransformationMatrix4x4 is a 4 by 4 matrix represented as a list of sixteen double values in row major order.

Table 12. Union types used in Core

Name	Description
CityModelMember «Union»	CityModelMember is a union type that enumerates the different types of objects that can occur as members of a city model.
DoubleOrNilReason «Union»	DoubleOrNilReason is a union type that allows for choosing between a double value and a nil reason.
NilReason «Union»	NilReason is a union type that allows for choosing between two different types of nil reason.

Table 13. Enumerated Classes used in Core

Name	Description
RelativeToTerrain «Enumeration»	RelativeToTerrain enumerates the spatial relations of a city object relative to terrain in a qualitative way.
RelativeToWater «Enumeration»	RelativeToWater enumerates the spatial relations of a city object relative to the water surface in a qualitative way.
SpaceType «Enumeration»	SpaceType is an enumeration that characterises a space according to its closure properties.

Table 14. CodeList Classes used in Core

Name	Description
IntervalValue «CodeList»	IntervalValue is a code list used to specify a time period.
MimeTypeValue «CodeList»	MimeTypeValue is a code list used to specify the MIME type of a referenced resource.
NilReasonEnumeration «CodeList»	NilReasonEnumeration is a code list that enumerates the different nil reasons.
OccupantTypeValue «CodeList»	OccupantTypeValue is a code list used to classify occupants.
OtherRelationTypeValue «CodeList»	OtherRelationTypeValue is a code list used to classify other types of city object relations.
QualifiedAreaTypeValue «CodeList»	QualifiedAreaTypeValue is a code list used to specify area types.
QualifiedVolumeTypeValue «CodeList»	QualifiedVolumeTypeValue is a code list used to specify volume types.
RelationTypeValue «CodeList»	RelationTypeValue is a code list used to classify city object relations.
TemporalRelationTypeValue «CodeList»	TemporalRelationTypeValue is a code list used to classify temporal city object relations.
TopologicalRelationTypeValue «CodeList»	TopologicalRelationTypeValue is a code list used to classify topological city object relations.

8.2.8. Additional Information

A detailed discussion of the CityGML Core can be found in the [OGC CityGML 3.0 Users Guide](#).

8.3. Appearance

Requirements Class

<http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-appearance>

Target type	Implementation Specification
Dependency	/req/req-class-core

The Appearance module provides the representation of surface data, i.e. observable properties for surface geometry objects in the form of textures and material.

Appearances are not limited to visual data but represent arbitrary categories called themes such as

infrared radiation, noise pollution, or earthquake-induced structural stress. A single surface geometry object may have surface data for multiple themes. Similarly, surface data can be shared by multiple surface geometry objects (e.g. road paving).

Surface data that is constant across a surface is modelled as material based on the material definitions from the standards X3D and COLLADA. Surface data that depends on the exact location within the surface is modelled as texture. This can either be a parameterized texture, i.e. a texture that uses texture coordinates or a transformation matrix for parameterization, or a georeferenced texture, i.e. a texture that uses a planimetric projection.

Each surface geometry object can have both, a material and a texture per theme and side. This allows for providing a constant approximation and a complex measurement of a surface's property simultaneously.

The UML diagram of the Appearance module is illustrated in [Figure 22](#). A detailed discussion of this Requirements Class can be found in the [CityGML 3.0 Users Guide](#).

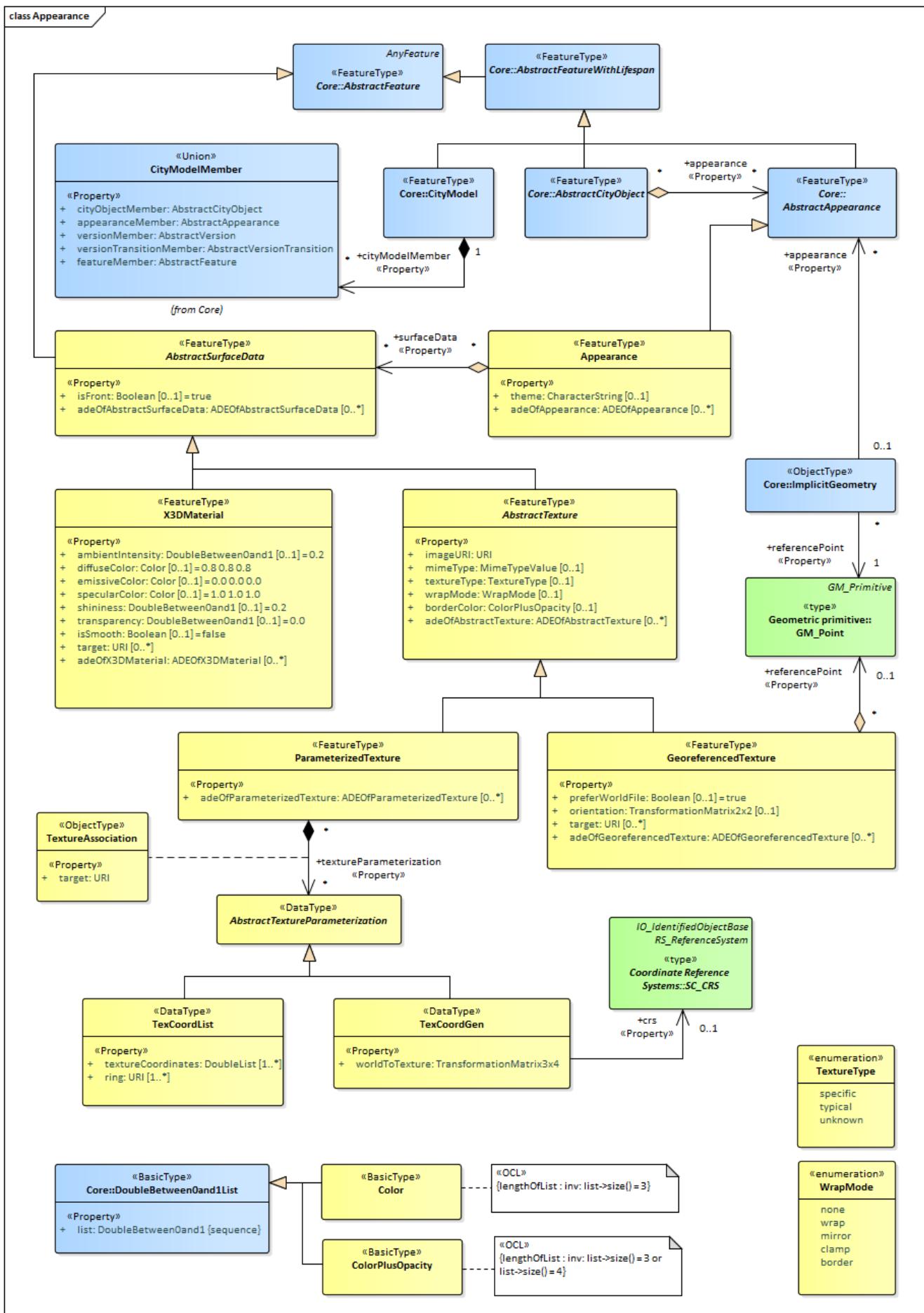


Figure 22. UML diagram of CityGML's Appearance model.

The ADE data types provided for the Appearance module are illustrated in the figure Figure 23.

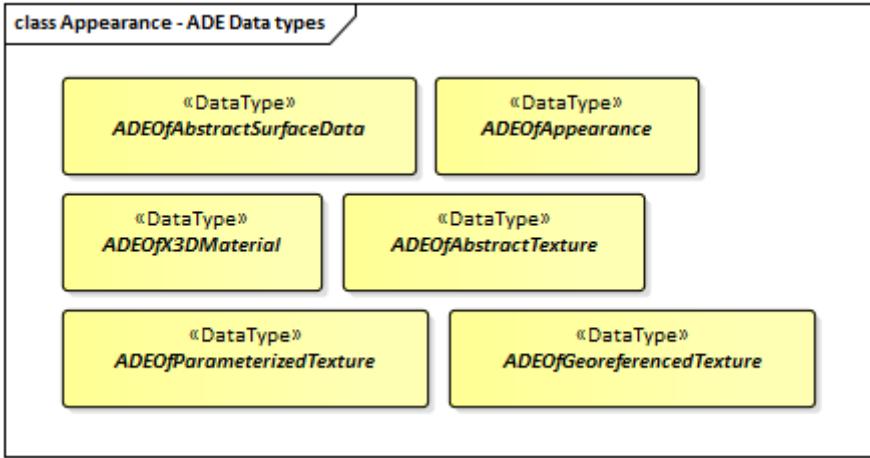


Figure 23. ADE classes of the CityGML Appearance Module.

8.3.1. Requirements

The following requirement defines the rules governing implementation of the CityGML Appearance Module as an Implementation Specification.

Requirement 5 /req/appearance/classes	
For each UML class defined or referenced in the Appearance Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and mutiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The use of extension capabilities by Appearance elements is constrained by the following requirement:

Requirement 6 /req/Appearance/ade/use	
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

8.3.2. Class Definitions

Table 15. Classes used in Appearance

Class	Description
AbstractSurfaceData «FeatureType»	AbstractSurfaceData is the abstract superclass for different kinds of textures and material.
AbstractTexture «FeatureType»	AbstractTexture is the abstract superclass to represent the common attributes of the classes ParameterizedTexture and GeoreferencedTexture.
Appearance «FeatureType»	An Appearance is a collection of surface data, i.e. observable properties for surface geometry objects in the form of textures and material.
GeoreferencedTexture «FeatureType»	A GeoreferencedTexture is a texture that uses a planimetric projection. It contains an implicit parameterization that is either stored within the image file, an accompanying world file or specified using the orientation and referencePoint elements.
ParameterizedTexture «FeatureType»	A ParameterizedTexture is a texture that uses texture coordinates or a transformation matrix for parameterization.
X3DMaterial «FeatureType»	X3DMaterial defines properties for surface geometry objects based on the material definitions from the standards X3D and COLLADA.
TextureAssociation «ObjectType»	TextureAssociation denotes the relation of a texture to a surface geometry object.

Table 16. Data Types used in Appearance

Name	Description
AbstractTextureParameterization «DataType»	AbstractTextureParameterization is the abstract superclass for different kinds of texture parameterizations.
ADEOfAbstractSurfaceData «DataType»	ADEOfAbstractSurfaceData acts as a hook to define properties within an ADE that are to be added to AbstractSurfaceData.
ADEOfAbstractTexture «DataType»	ADEOfAbstractTexture acts as a hook to define properties within an ADE that are to be added to AbstractTexture.
ADEOfAppearance «DataType»	ADEOfAppearance acts as a hook to define properties within an ADE that are to be added to an Appearance.
ADEOfGeoreferencedTexture «DataType»	ADEOfGeoreferencedTexture acts as a hook to define properties within an ADE that are to be added to a GeoreferencedTexture.
ADEOfParameterizedTexture «DataType»	ADEOfParameterizedTexture acts as a hook to define properties within an ADE that are to be added to a ParameterizedTexture.

ADEOfX3DMaterial «DataType»	ADEOfX3DMaterial acts as a hook to define properties within an ADE that are to be added to an X3DMaterial.
TexCoordGen «DataType»	TexCoordGen defines texture parameterization using a transformation matrix.
TexCoordList «DataType»	TexCoordList defines texture parameterization using texture coordinates.

Table 17. Primitive Data Types used in Appearance

Name	Description
Color «BasicType»	Color is a list of three double values between 0 and 1 defining an RGB color value.
ColorPlusOpacity «BasicType»	Color is a list of four double values between 0 and 1 defining an RGBA color value. Opacity value of 0 means transparent.

Table 18. Enumerated Classes used in Appearance

Name	Description
TextureType «Enumeration»	TextureType enumerates the different texture types.
WrapMode «Enumeration»	WrapMode enumerates the different fill modes for textures.

8.3.3. Additional Information

Additional information about the Appearance Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.4. City Furniture

Requirements Class	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-cityfurniture	
Target type	Implementation Specification
Dependency	/req/req-class-core

The CityFurniture module provides the representation of objects or pieces of equipment that are installed in the outdoor environment for various purposes, such as decoration, explanation or control. City furniture objects are relatively small, immovable objects and usually are of stereotypical form. Examples include road signs, traffic signals, bicycle racks, street lamps, fountains, flower buckets, advertising columns, and benches.

City furniture is represented in the UML model by the top-level feature type *CityFurniture*, which is also the only class of the CityFurniture module.

The UML diagram of the CityFurniture module is depicted in [Figure 24](#). A detailed discussion of this

Requirements Class can be found in the [CityGML 3.0 Users Guide](#).

[CityFurniture] | *figures/CityFurniture.png*

Figure 24. UML diagram of CityGML's City Furniture model.

The ADE data types and Code Lists provided for the CityFurniture module are illustrated in the figure [Figure 25](#).

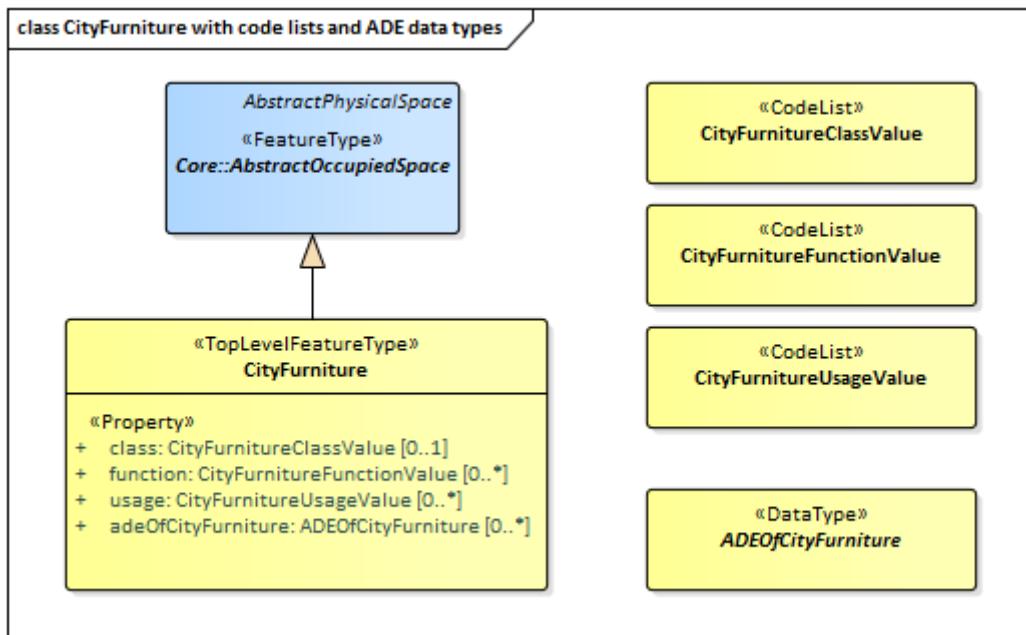


Figure 25. ADE classes and Code Lists of the CityGML CityFurniture module.

[Table 19](#) lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the CityFurniture module:

Table 19. CityFurniture space classes and their allowed thematic surface boundaries

Space class	Allowed space boundaries
CityFurniture	<ul style="list-style-type: none"> • Core::ClosureSurface • Generics::GenericThematicSurface • Possible classes from ADEs

8.4.1. Requirements

The following requirement defines the rules governing implementation of the CityGML City Furniture Module as an Implementation Specification.

Requirement 7 /req/cityfurniture/classes	
For each UML class defined or referenced in the CityFurniture Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.

B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and mutiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The implementation of this Module is further constrained by the following spatial boundary requirement:

Requirement 8 /req/cityfurniture/boundaries

Table 19 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the CityFurniture module. An Implementation Specification SHALL NOT specify boundaries except as specified in Table 19

The use of extension capabilities by City Furniture elements is constrained by the following requirement:

Requirement 9 /req/CityFurniture/ade/use

ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.

8.4.2. Class Definitions

Table 20. Classes used in CityFurniture

Class	Description
CityFurniture «TopLevelFeatureType»	CityFurniture is an object or piece of equipment installed in the outdoor environment for various purposes. Examples include street signs, traffic signals, street lamps, benches, fountains.

Table 21. Data Types used in CityFurniture

Name	Description
ADEOfCityFurniture «DataType»	ADEOfCityFurniture acts as a hook to define properties within an ADE that are to be added to a CityFurniture.

Table 22. CodeList Classes used in CityFurniture

Name	Description
CityFurnitureClassValue «CodeList»	CityFurnitureClassValue is a code list used to further classify a CityFurniture.
CityFurnitureFunctionValue «CodeList»	CityFurnitureFunctionValue is a code list that enumerates the different purposes of a CityFurniture.
CityFurnitureUsageValue «CodeList»	CityFurnitureUsageValue is a code list that enumerates the different uses of a CityFurniture.

8.4.3. Additional Information

Additional information about the City Furniture Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.5. City Object Group

Requirements Class	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-cityobjectgroup	
Target type	Implementation Specification
Dependency	/req/req-class-core

The CityObjectGroup module provides the application-specific aggregation of city objects according to some user-defined criteria. Examples for groups are the buildings in a specific region, the result of a query, or objects put together for visualization purposes. Each member of a group may be qualified by a role name, reflecting the role each city object plays in the context of the group. City object groups are represented in the UML model by the top-level feature type *CityObjectGroup*, which is the main class of the CityObjectGroup module.

City object groups can be linked to other city objects, the so-called parent objects, which allows for modelling a generic hierarchical grouping concept. In addition, as city object groups represent city objects themselves, a group may become a member of another group realizing recursive aggregation in this way.

The UML diagram of the CityObjectGroup module is depicted in [Figure 26](#). A detailed discussion of this Requirements Class can be found in the [CityGML 3.0 Users Guide](#).

[CityObjectGroup] | *figures/CityObjectGroup.png*

Figure 26. UML diagram of the City Object Group Model.

The ADE data types provided for the CityObjectGroup module are illustrated in the figure [Figure 27](#).

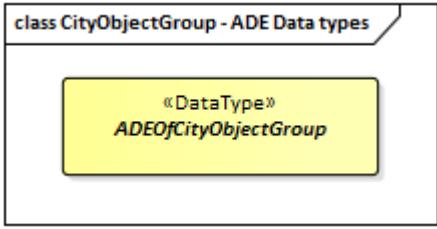


Figure 27. ADE classes of the CityGML CityObjectGroup module.

The Code Lists provided for the CityObjectGroup module are illustrated in the figure [Figure 28](#).

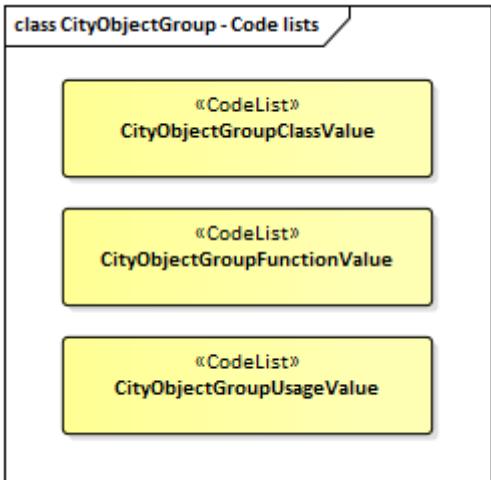


Figure 28. Codelists from the CityGML CityObjectGroup module.

[Table 23](#) lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the CityObjectGroup module:

Table 23. CityObjectGroup space classes and their allowed thematic surface boundaries

Space class	Allowed space boundaries
CityObjectGroup	<ul style="list-style-type: none"> • Core::ClosureSurface • Generics::GenericThematicSurface • Possible classes from ADEs

8.5.1. Requirements

The following requirement defines the rules governing implementation of the CityGML City Object Group Module as an Implementation Specification.

Requirement 10 /req/cityobjectgroup/classes	
For each UML class defined or referenced in the CityObjectGroup Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.

B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and mutiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The implementation of this Module is further constrained by the following spatial boundary requirement:

Requirement 11	/req/cityobjectgroup/boundaries
Table 23 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the CityObjectGroup module. An Implementation Specification SHALL NOT specify boundaries except as specified in Table 23	

The use of extension capabilities by City Object Group elements is constrained by the following requirement:

Requirement 12	/req/CityObjectGroup/ade/use
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

8.5.2. Class Definitions

Table 24. Classes used in CityObjectGroup

Class	Description
CityObjectGroup «TopLevelFeatureType»	A CityObjectGroup represents an application-specific aggregation of city objects according to some user-defined criteria. Examples for groups are the buildings in a specific region, the result of a query, or objects put together for visualization purposes. Each member of a group may be qualified by a role name, reflecting the role each city object plays in the context of the group.
Role «ObjectType»	Role qualifies the function of a city object within the CityObjectGroup.

Table 25. Data Types used in CityObjectGroup

Name	Description
ADEOfCityObjectGroup «DataType»	ADEOfCityObjectGroup acts as a hook to define properties within an ADE that are to be added to a CityObjectGroup.

Table 26. CodeList Classes used in CityObjectGroup

Name	Description
CityObjectGroupClassValue «CodeList»	CityObjectGroupClassValue is a code list used to further classify a CityObjectGroup.
CityObjectGroupFunctionValue «CodeList»	CityObjectGroupFunctionValue is a code list that enumerates the different purposes of a CityObjectGroup.
CityObjectGroupUsageValue «CodeList»	CityObjectGroupUsageValue is a code list that enumerates the different uses of a CityObjectGroup.

8.5.3. Additional Information

Additional information about the City Object Group Module can be found in the [OGC CityGML 3.0 Users Guide](#).

8.6. Dynamizer

Requirements Class	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-dynamizer	
Target type	Implementation Specification
Dependency	/req/req-class-core

The Dynamizer module provides the concepts that allow for representing time-varying data for city object properties as well as for integrating sensors with 3D city models. Dynamizers are objects that inject timeseries data for an individual attribute of the city object in which it is included. The timeseries data overrides the static value of the referenced city object attribute in order to represent dynamic, i.e. time-dependent, variations of its value.

The dynamic values may be given by retrieving observation results directly from external sensor/IoT services using a sensor connection (e.g. OGC SensorThings API, Sensor Observation Service, or other sensor data platforms including MQTT). Alternatively, the dynamic values may be provided as atomic timeseries that represent time-varying data of a specific data type for a single contiguous time interval. The data can be provided in external tabulated files, such as CSV or Excel sheets, in external files that format timeseries data according to the OGC TimeseriesML or OGC Observations & Measurements standards, or inline as embedded time-value-pairs. Furthermore, timeseries data can also be aggregated to form composite timeseries with non-overlapping time intervals.

By using the Dynamizer module, fast changes over a short or longer time period with respect to cities and city models can be represented. This includes variations of spatial properties, i.e. change of a feature's geometry, both in respect to shape and to location (e.g. moving objects), variations of thematic attributes, i.e. changes of physical quantities like energy demands, temperatures, solar irradiation, traffic density, pollution concentration, or overpressure on building walls, and variations with respect to sensor or real-time data resulting from simulations or measurements.

The UML diagram of the Dynamizer module is depicted in Figure 29. A detailed discussion of this Requirements Class can be found in the [CityGML 3.0 Users Guide](#).

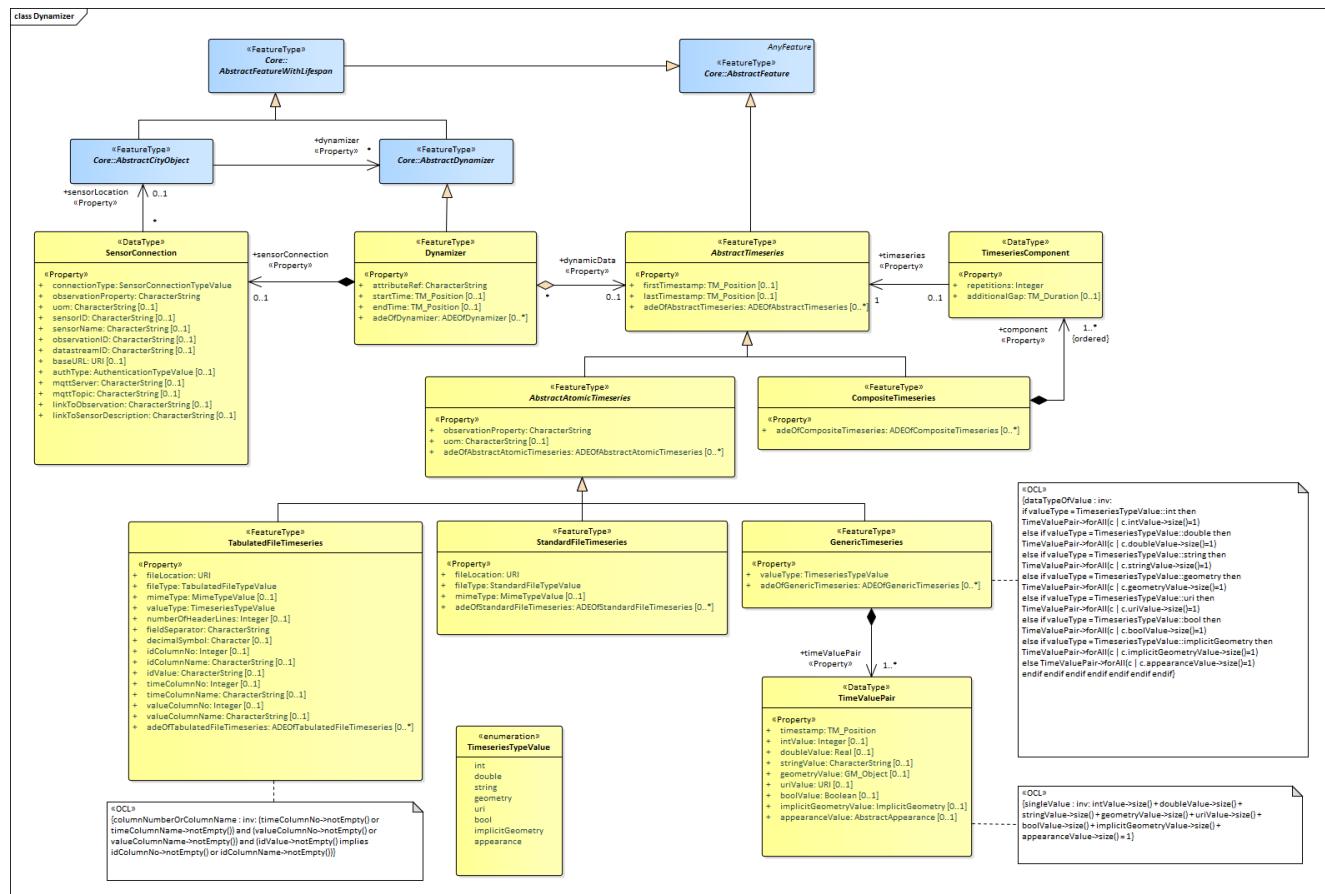


Figure 29. UML diagram of the Dynamizer Model.

The ADE data types provided for the Dynamizer module are illustrated in the figure Figure 30.

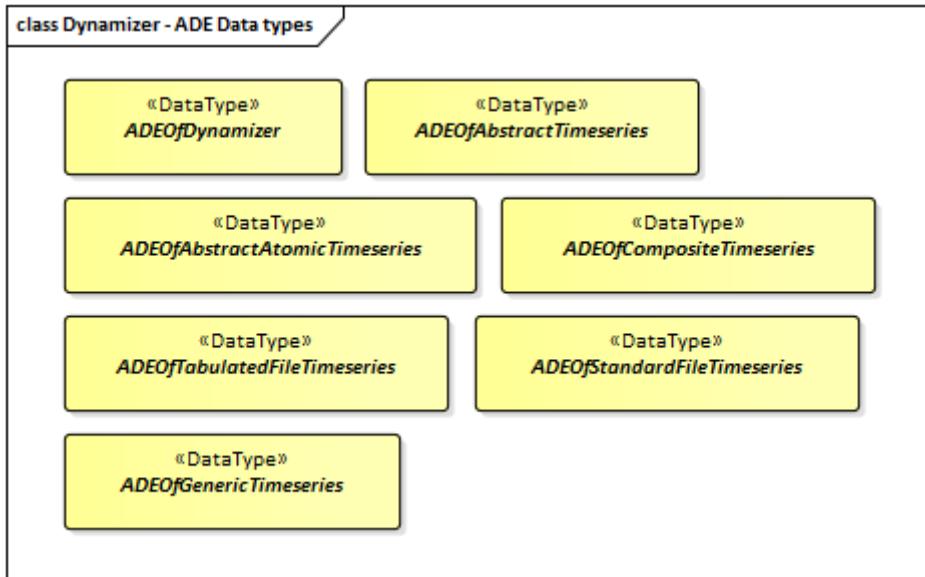


Figure 30. ADE classes of the CityGML Dynamizer module.

The Code Lists provided for the Dynamizer module are illustrated in the figure [Figure 31](#).

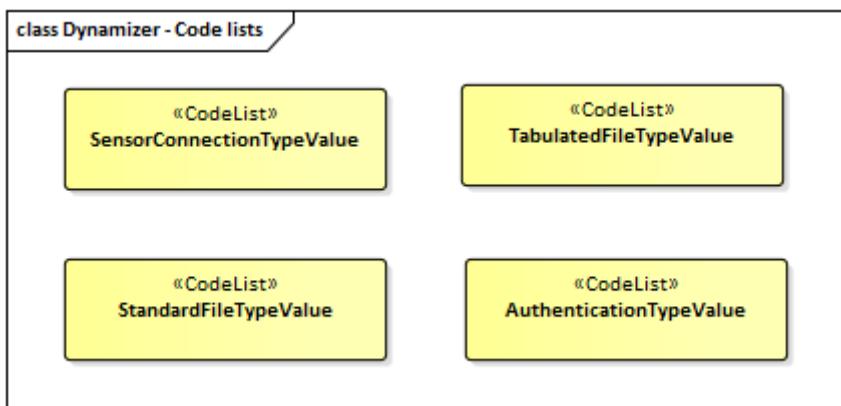


Figure 31. Codelists from the CityGML Dynamizer module.

8.6.1. Requirements

The following requirement defines the rules governing implementation of the CityGML Dynamizer Module as an Implementation Specification.

Requirement 13 /req/dynamizer/classes	
For each UML class defined or referenced in the Dynamizer Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.

D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and multiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The use of extension capabilities by Dynamizer elements is constrained by the following requirement:

Requirement 14	/req/Dynamizer/ade/use
ADE element and property extensions	SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.

8.6.2. Class Definitions

Table 27. Classes used in Dynamizer

Class	Description
AbstractAtomicTimeseries «FeatureType»	AbstractAtomicTimeseries represents the attributes and relationships that are common to all kinds of atomic timeseries (GenericTimeseries, TabulatedFileTimeseries, StandardFileTimeseries). An atomic timeseries represents time-varying data of a specific data type for a single contiguous time interval.
AbstractTimeseries «FeatureType»	AbstractTimeseries is the abstract superclass representing any type of timeseries data.
CompositeTimeseries «FeatureType»	A CompositeTimeseries is a (possibly recursive) aggregation of atomic and composite timeseries. The components of a composite timeseries must have non-overlapping time intervals.
Dynamizer «FeatureType»	A Dynamizer is an object that injects timeseries data for an individual attribute of the city object in which it is included. The timeseries data overrides the static value of the referenced city object attribute in order to represent dynamic, i.e. time-dependent, variations of its value.
GenericTimeseries «FeatureType»	A GenericTimeseries represents time-varying data in the form of embedded time-value-pairs of a specific data type for a single contiguous time interval.

StandardFileTimeseries «FeatureType»	A StandardFileTimeseries represents time-varying data for a single contiguous time interval. The data is provided in an external file referenced in the StandardFileTimeseries. The data within the external file shall be encoded according to a dedicated format for the representation of timeseries data, for example, the OGC TimeseriesML or OGC Observations & Measurements standard. The data type of the data has to be specified within the external file.
TabulatedFileTimeseries «FeatureType»	A TabulatedFileTimeseries represents time-varying data of a specific data type for a single contiguous time interval. The data is provided in an external file referenced in the TabulatedFileTimeseries. The file shall contain table structured data using an appropriate file format like comma-separated values (CSV), Microsoft Excel (XLSX) or Google Spreadsheet. The timestamps and the values are given in specific columns of the table. Each row represents a single time-value-pair. A subset of rows can be selected using the idColumn and idValue attributes.

Table 28. Data Types used in Dynamizer

Name	Description
ADEOfAbstractAtomicTimeseries «DataType»	ADEOfAbstractAtomicTimeseries acts as a hook to define properties within an ADE that are to be added to AbstractAtomicTimeseries.
ADEOfAbstractTimeseries «DataType»	ADEOfAbstractTimeseries acts as a hook to define properties within an ADE that are to be added to AbstractTimeseries.
ADEOfCompositeTimeseries «DataType»	ADEOfCompositeTimeseries acts as a hook to define properties within an ADE that are to be added to a CompositeTimeseries.
ADEOfDynamizer «DataType»	ADEOfDynamizer acts as a hook to define properties within an ADE that are to be added to a Dynamizer.
ADEOfGenericTimeseries «DataType»	ADEOfGenericTimeseries acts as a hook to define properties within an ADE that are to be added to a GenericTimeseries.
ADEOfStandardFileTimeseries «DataType»	ADEOfStandardFileTimeseries acts as a hook to define properties within an ADE that are to be added to a StandardFileTimeseries.
ADEOfTabulatedFileTimeseries «DataType»	ADEOfTabulatedFileTimeseries acts as a hook to define properties within an ADE that are to be added to a TabulatedFileTimeseries.

SensorConnection «DataType»	A SensorConnection provides all details that are required to retrieve a specific datastream from an external sensor web service. It comprises the service type (e.g. OGC SensorThings API, OGC Sensor Observation Services, MQTT, proprietary platforms), the URL of the sensor service, the identifier for the sensor or thing, and its observed property as well as information about the required authentication method.
TimeseriesComponent «DataType»	TimeseriesComponent represents an element of a CompositeTimeseries.
TimeValuePair «DataType»	A TimeValuePair represents a value that is valid for a given timepoint. For each TimeValuePair, only one of the value properties can be used mutually exclusive. Which value property has to be provided depends on the selected value type in the GenericTimeSeries feature, in which the TimeValuePair is included.

Table 29. Enumerated Classes used in Dynamizer

Name	Description
TimeseriesTypeValue «Enumeration»	TimeseriesTypeValue enumerates the possible value types for GenericTimeseries and TimeValuePair.

Table 30. CodeList Classes used in Dynamizer

Name	Description
AuthenticationTypeValue «CodeList»	AuthenticationTypeValue is a code list used to specify the authentication method to be used to access the referenced sensor service. Each value shall provide enough information such that a software application could determine the required access credentials.
SensorConnectionTypeValue «CodeList»	SensorConnectionTypeValue is a code list used to specify the type of the referenced sensor service. Each value shall provide enough information such that a software application would be able to identify the API type and version.
StandardFileTypeValue «CodeList»	StandardFileTypeValue is a code list used to specify the type of the referenced external timeseries data file. Each value shall provide information about the standard and version.
TabulatedFileTypeValue «CodeList»	TabulatedFileTypeValue is a code list used to specify the data format of the referenced external tabulated data file.

8.6.3. Additional Information

Additional information about the Dynamizer Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.7. Generics

Requirements Class

<http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-generics>

Target type	Implementation Specification
Dependency	/req/req-class-core

The Generics module provides the representation of generic city objects, i.e. city objects that are not covered by any explicitly modelled thematic class within CityGML, and of generic attributes, i.e. attributes that are not explicitly represented in CityGML. In order to avoid problems concerning semantic interoperability, generic city objects and generic attributes shall only be used if appropriate thematic classes and attributes are not provided by any other CityGML module.

In accordance with the CityGML Space concept defined in the Core module ([cf. Section Core](#)) generic city objects can be represented as generic logical spaces, generic occupied spaces, generic unoccupied spaces, and generic thematic surfaces. In this way, spaces and surfaces can be defined that are not represented by any explicitly modelled class within CityGML that is a subclass of the classes `AbstractLogicalSpace`, `AbstractOccupiedSpace`, `AbstractUnoccupiedSpace` or `AbstractThematicSurface`, respectively. Generic city objects are represented in the UML model by the top-level feature types `GenericLogicalSpace`, `GenericOccupiedSpace`, `GenericUnoccupiedSpace` and `GenericThematicSurface`.

Generic attributes are defined as name-value pairs and are always associated with a city object. Generic attributes can be of type String, Integer, Double, Date, URI, Measure, and Code. In addition, generic attributes can be grouped under a common name as generic attribute sets.

The UML diagram of the Generics module is depicted in [Figure 32](#). A detailed discussion of this Requirements Class can be found in the [CityGML 3.0 Users Guide](#).

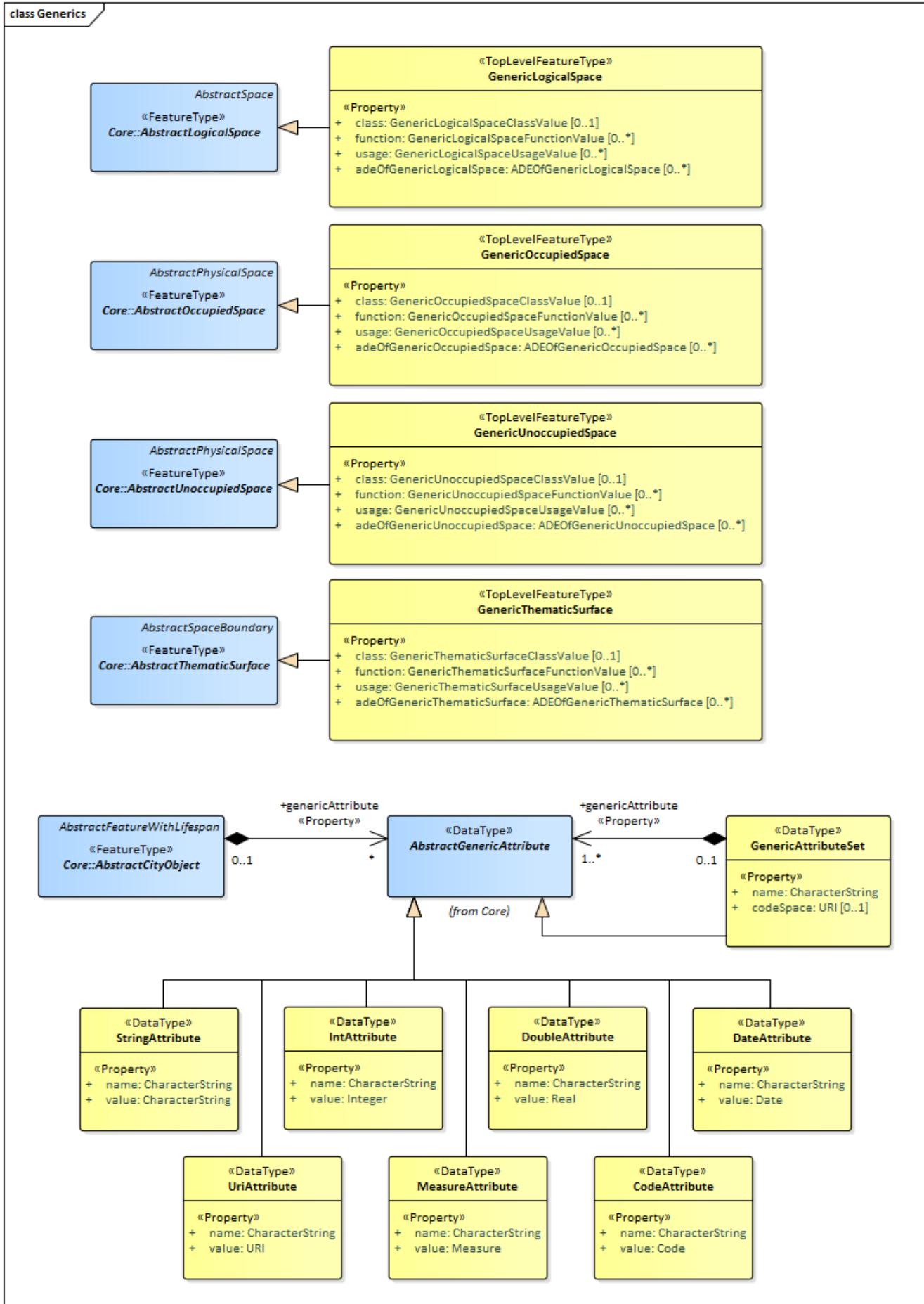


Figure 32. UML diagram of the Generics Model.

The ADE data types provided for the Generics module are illustrated in the figure Figure 33.

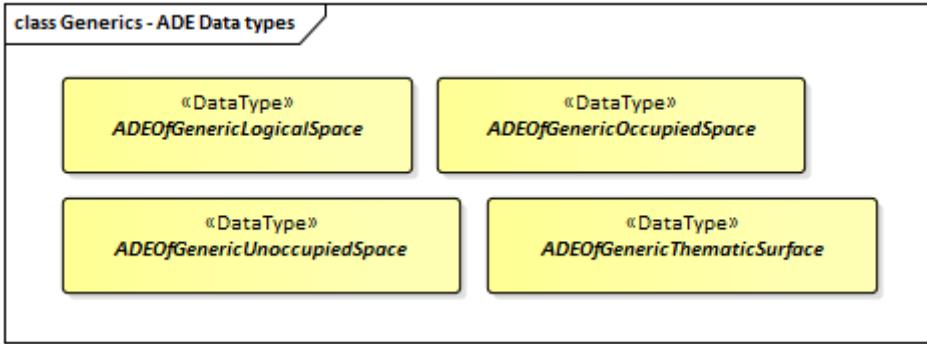


Figure 33. ADE classes of the CityGML Generics module.

The Code Lists provided for the Generics module are illustrated in the figure [Figure 34](#).

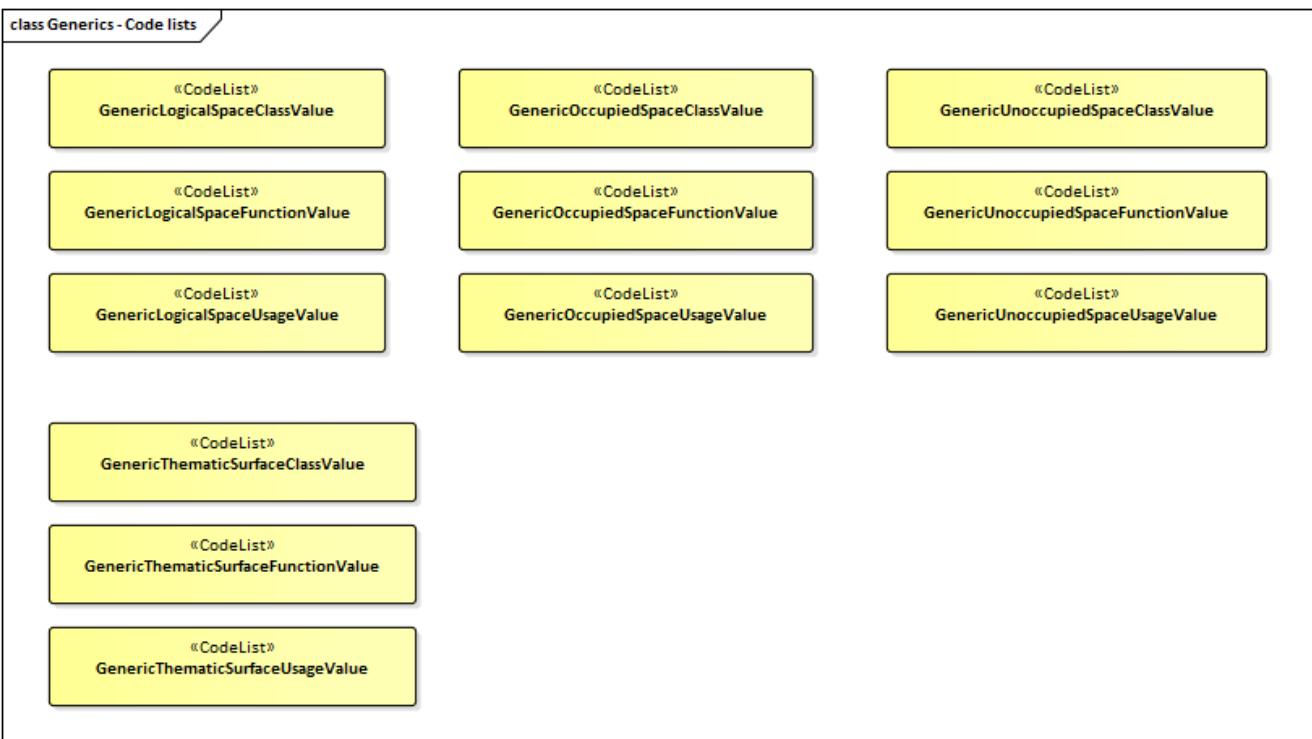


Figure 34. Codelists from the CityGML Generics module.

[Table 31](#) lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Generics module:

Table 31. Generics space classes and their allowed thematic surface boundaries

Space class	Allowed space boundaries
GenericLogicalSpace	<ul style="list-style-type: none"> • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
GenericOccupiedSpace	<ul style="list-style-type: none"> • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs

GenericUnoccupiedSpace	<ul style="list-style-type: none"> • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
------------------------	--

8.7.1. Requirements

The following requirement defines the rules governing implementation of the CityGML Generics Module as an Implementation Specification.

Requirement 15 /req/generics/classes	
For each UML class defined or referenced in the Generics Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and mutiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The implementation of this Module is further constrained by the following spatial boundary requirement:

Requirement 16 /req/generics/boundaries	
Table 31 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Generics module. An Implementation Specification SHALL NOT specify boundaries except as specified in Table 31	

The decision of whether or not to use Generics is constrained by the following requirement:

Requirement 17 /req/generics/use	
Generic objects and attributes SHALL only be used if a more specific feature class or attribute is not available from the CityGML conceptual model.	

The use of extension capabilities by Generics elements is constrained by the following requirement:

Requirement 18	/req/Generics/ade/use
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

8.7.2. Class Definitions

Table 32. Classes used in Generics

Class	Description
GenericLogicalSpace «TopLevelFeatureType»	A GenericLogicalSpace is a space that is not represented by any explicitly modelled AbstractLogicalSpace subclass within CityGML.
GenericOccupiedSpace «TopLevelFeatureType»	A GenericOccupiedSpace is a space that is not represented by any explicitly modelled AbstractOccupiedSpace subclass within CityGML.
GenericThematicSurface «TopLevelFeatureType»	A GenericThematicSurface is a surface that is not represented by any explicitly modelled AbstractThematicSurface subclass within CityGML.
GenericUnoccupiedSpace «TopLevelFeatureType»	A GenericUnoccupiedSpace is a space that is not represented by any explicitly modelled AbstractUnoccupiedSpace subclass within CityGML.

Table 33. Data Types used in Generics

Name	Description
ADEOfGenericLogicalSpace «DataType»	ADEOfGenericLogicalSpace acts as a hook to define properties within an ADE that are to be added to a GenericLogicalSpace.
ADEOfGenericOccupiedSpace «DataType»	ADEOfGenericOccupiedSpace acts as a hook to define properties within an ADE that are to be added to a GenericOccupiedSpace.
ADEOfGenericThematicSurface «DataType»	ADEOfGenericThematicSurface acts as a hook to define properties within an ADE that are to be added to a GenericThematicSurface.
ADEOfGenericUnoccupiedSpace «DataType»	ADEOfGenericUnoccupiedSpace acts as a hook to define properties within an ADE that are to be added to a GenericUnoccupiedSpace.
CodeAttribute «DataType»	CodeAttribute is a data type used to define generic attributes of type "Code".
DateAttribute «DataType»	DateAttribute is a data type used to define generic attributes of type "Date".
DoubleAttribute «DataType»	DoubleAttribute is a data type used to define generic attributes of type "Double".

GenericAttributeSet «DataType»	A GenericAttributeSet is a named collection of generic attributes.
IntAttribute «DataType»	IntAttribute is a data type used to define generic attributes of type "Integer".
MeasureAttribute «DataType»	MeasureAttribute is a data type used to define generic attributes of type "Measure".
StringAttribute «DataType»	StringAttribute is a data type used to define generic attributes of type "String".
UriAttribute «DataType»	UriAttribute is a data type used to define generic attributes of type "URI".

Table 34. *CodeList Classes used in Generics*

Name	Description
GenericLogicalSpaceClassValue «CodeList»	GenericLogicalSpaceClassValue is a code list used to further classify a GenericLogicalSpace.
GenericLogicalSpaceFunctionValue «CodeList»	GenericLogicalSpaceFunctionValue is a code list that enumerates the different purposes of a GenericLogicalSpace.
GenericLogicalSpaceUsageValue «CodeList»	GenericLogicalSpaceUsageValue is a code list that enumerates the different uses of a GenericLogicalSpace.
GenericOccupiedSpaceClassValue «CodeList»	GenericOccupiedSpaceClassValue is a code list used to further classify a GenericOccupiedSpace.
GenericOccupiedSpaceFunctionValue «CodeList»	GenericOccupiedSpaceFunctionValue is a code list that enumerates the different purposes of a GenericOccupiedSpace.
GenericOccupiedSpaceUsageValue «CodeList»	GenericOccupiedSpaceUsageValue is a code list that enumerates the different uses of a GenericOccupiedSpace.
GenericThematicSurfaceClassValue «CodeList»	GenericThematicSurfaceClassValue is a code list used to further classify a GenericThematicSurface.
GenericThematicSurfaceFunctionValue «CodeList»	GenericThematicSurfaceFunctionValue is a code list that enumerates the different purposes of a GenericThematicSurface.
GenericThematicSurfaceUsageValue «CodeList»	GenericThematicSurfaceUsageValue is a code list that enumerates the different uses of a GenericThematicSurface.

<code>GenericUnoccupiedSpaceClassValue</code> «CodeList»	GenericUnoccupiedSpaceClassValue is a code list used to further classify a GenericUnoccupiedSpace.
<code>GenericUnoccupiedSpaceFunctionValue</code> «CodeList»	GenericUnoccupiedSpaceFunctionValue is a code list that enumerates the different purposes of a GenericUnoccupiedSpace.
<code>GenericUnoccupiedSpaceUsageValue</code> «CodeList»	GenericUnoccupiedSpaceUsageValue is a code list that enumerates the different uses of a GenericUnoccupiedSpace.

8.7.3. Additional Information

Additional information about the Genericsn Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.8. Land Use

Requirements Class

<http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-landuse>

Target type	Implementation Specification
Dependency	/req/req-class-core

The LandUse module defines objects that can be used to describe areas of the earth's surface dedicated to a specific land use or having a specific land cover with or without vegetation, such as sand, rock, mud flats, forest, grasslands, or wetlands (i.e. the physical appearance). Land use and land cover are different concepts; the first describes human activities on the earth's surface, the second describes its physical and biological cover. However, the two concepts are interlinked and often mixed in practice. Land use objects in CityGML support both concepts: They can be employed to represent parcels, spatial planning objects, recreational objects, and objects describing the physical characteristics of an area in 3D. Land use objects are represented in the UML model by the top-level feature type *LandUse*, which is also the only class of the LandUse module.

The UML diagram of the LandUse module is depicted in [Figure 35](#). A detailed discussion of this Requirements Class can be found in the [CityGML User Guide](#).

[LandUse] | *figures/LandUse.png*

Figure 35. UML diagram of the Land Use Model.

The ADE data types provided for the Land Use module are illustrated in the figure [Figure 36](#).

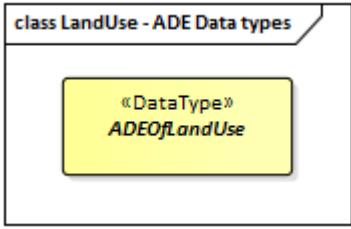


Figure 36. ADE classes of the CityGML Land Use module.

The Code Lists provided for the Land Use module are illustrated in the figure [Figure 37](#).

[LandUse Codelists] | *figures/LandUse-Codelists.png*

Figure 37. Codelists from the CityGML Land Use module.

8.8.1. Requirements

The following requirement defines the rules governing implementation of the CityGML Land Use Module as an Implementation Specification.

Requirement 19	/req/landuse/classes
For each UML class defined or referenced in the LandUse Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and mutiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The use of extension capabilities by Land Use elements is constrained by the following requirement:

Requirement 20	/req/LandUse/ade/use
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

8.8.2. Class Definitions

Table 35. Classes used in LandUse

Class	Description
LandUse «TopLevelFeatureType»	A LandUse object is an area of the earth's surface dedicated to a specific land use or having a specific land cover with or without vegetation, such as sand, rock, mud flats, forest, grasslands, or wetlands.

Table 36. Data Types used in LandUse

Name	Description
ADEOfLandUse «DataType»	ADEOfLandUse acts as a hook to define properties within an ADE that are to be added to a LandUse.

Table 37. CodeList Classes used in LandUse

Name	Description
LandUseClassValue «CodeList»	LandUseClassValue is a code list used to further classify a LandUse.
LandUseFunctionValue «CodeList»	LandUseFunctionValue is a code list that enumerates the different purposes of a LandUse.
LandUseUsageValue «CodeList»	LandUseUsageValue is a code list that enumerates the different uses of a LandUse.

8.8.3. Additional Information

Additional information about the Land Use Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.9. Point Cloud

Requirements Class	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-pointcloud	
Target type	Implementation Specification
Dependency	/req/req-class-core

The PointCloud module offers the possibility to provide the geometry of physical spaces and of thematic surfaces by 3D point clouds. In this way, the building hull, a room within a building or a single wall surface can be spatially represented by a point cloud only. The same applies to all other thematic feature types including transportation objects, vegetation, city furniture, etc. Point clouds can either be provided inline within a CityGML file or as reference to external point cloud files of common file types such as LAS or LAZ. Point clouds are represented in the UML model by the feature type *PointCloud*, which is also the only class of the PointCloud module.

The UML diagram of the PointCloud module is depicted in [Figure 38](#). A detailed discussion of this

Requirements Class can be found in the [CityGML User Guide](#).

[PointCloud] | *figures/PointCloud.png*

Figure 38. UML diagram of the Point Cloud Model.

The ADE data types provided for the Point Cloud module are illustrated in the figure [Figure 39](#).

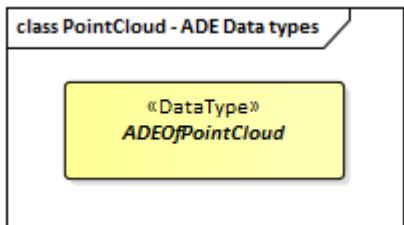


Figure 39. ADE classes of the CityGML Point Cloud module.

8.9.1. Requirements

The following requirement defines the rules governing implementation of the CityGML Point Cloud Module as an Implementation Specification.

Requirement 21 /req/pointcloud/classes	
For each UML class defined or referenced in the PointCloud Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and mutiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The use of extension capabilities by Point Cloud elements is constrained by the following requirement:

Requirement 22 /req/PointCloud/ade/use	

ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.

8.9.2. Class Definitions

Table 38. Classes used in PointCloud

Class	Description
PointCloud «FeatureType»	A PointCloud is an unordered collection of points that is a sampling of the geometry of a space or space boundary.
ADEOfPointCloud «DataType»	ADEOfPointCloud acts as a hook to define properties within an ADE that are to be added to a PointCloud.

8.9.3. Additional Information

Additional information about the Point Cloud Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.10. Relief

Requirements Class	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-relief	
Target type	Implementation Specification
Dependency	/req/req-class-core

The Relief module provides the representation of terrain which is an essential part of city models. In CityGML, the terrain is modelled by relief features. They are represented in the UML model by the top-level feature type *ReliefFeature*, which is the main class of the Relief module. The relief features, in turn, are collections of relief components that describe the Earth's surface, also known as the Digital Terrain Model. The relief components can have different terrain representations which can coexist. Each relief component may be specified as a regular raster or grid, as a TIN (Triangulated Irregular Network), by break lines, or by mass points. In addition, the validity of the relief components may be restricted to certain areas.

The UML diagram of the Relief module is depicted in [Figure 40](#). A detailed discussion of this Requirements Class can be found in the [CityGML User Guide](#).

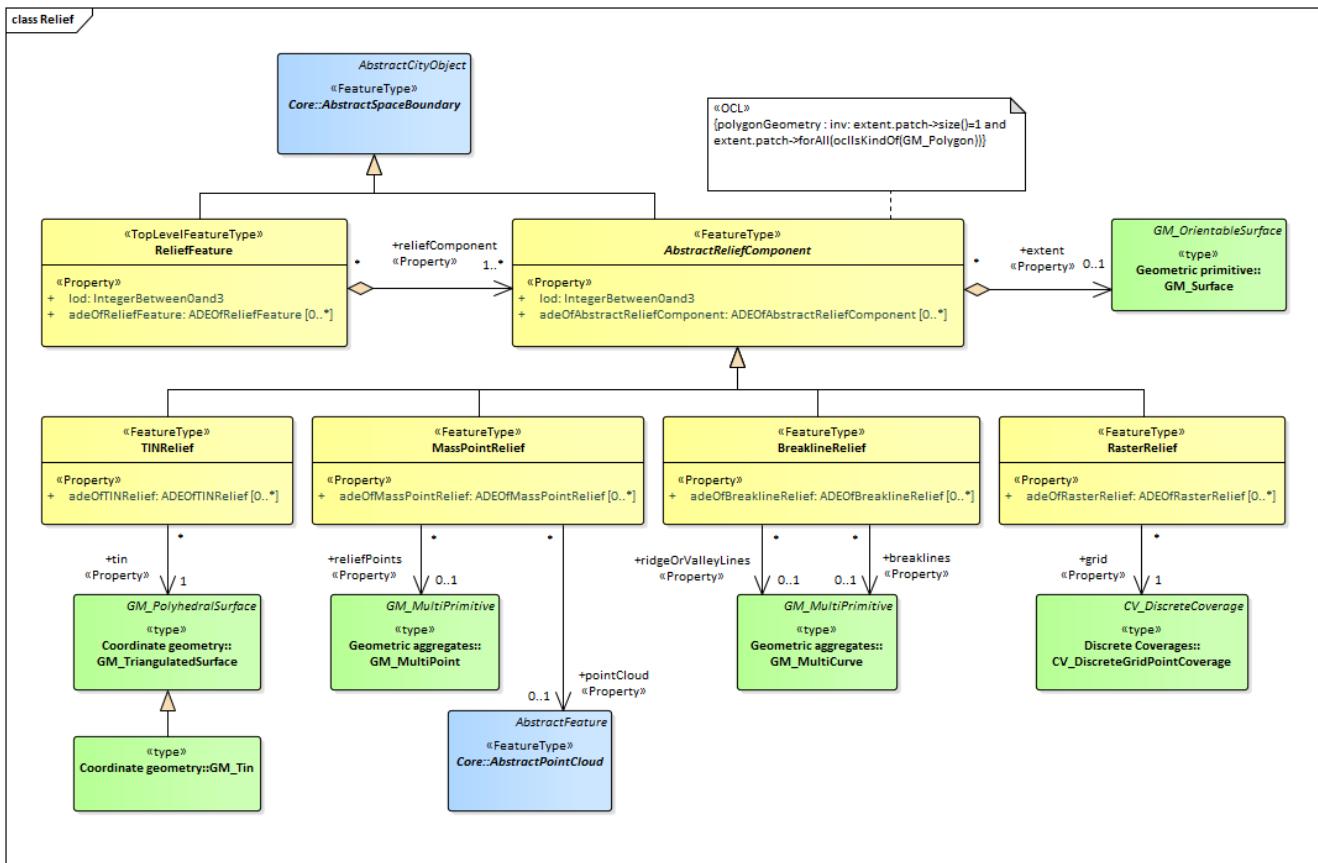


Figure 40. UML diagram of Relief module.

The ADE data types provided for the Relief module are illustrated in the figure [Figure 41](#).

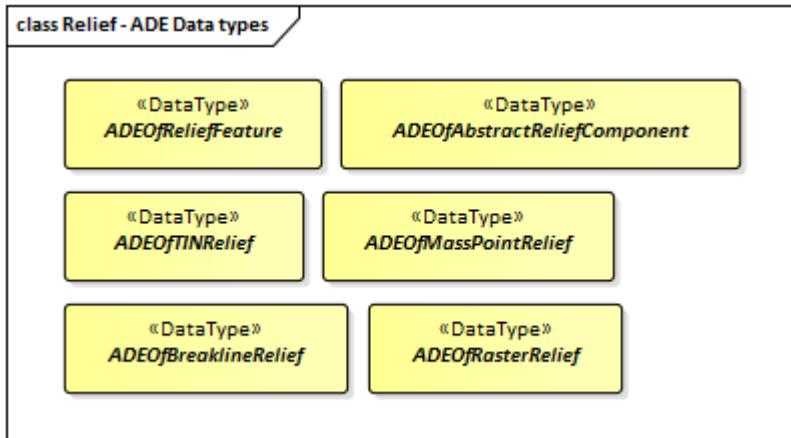


Figure 41. ADE classes of the CityGML Relief module.

8.10.1. Requirements

The following requirement defines the rules governing implementation of the CityGML Relief Module as an Implementation Specification.

Requirement 23	/req/relief/classes
-----------------------	---------------------

For each UML class defined or referenced in the Relief Package:

A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and mutiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The use of extension capabilities by Relief elements is constrained by the following requirement:

Requirement 24	/req/Relief/ade/use
	ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.

8.10.2. Class Definitions

Table 39. Classes used in Relief

Class	Description
ReliefFeature «TopLevelFeatureType»	A ReliefFeature is a collection of terrain components representing the Earth's surface, also known as the Digital Terrain Model.
AbstractReliefComponent «FeatureType»	An AbstractReliefComponent represents an element of the terrain surface - either a TIN, a raster or grid, mass points or break lines.
BreaklineRelief «FeatureType»	A BreaklineRelief represents a terrain component with 3D lines. These lines denote break lines or ridge/valley lines.
MassPointRelief «FeatureType»	A MassPointRelief represents a terrain component as a collection of 3D points.
RasterRelief «FeatureType»	A RasterRelief represents a terrain component as a regular raster or grid.
TINRelief «FeatureType»	A TINRelief represents a terrain component as a triangulated irregular network.

Table 40. Data Types used in Relief

Name	Description
ADEOfAbstractReliefComponent «DataType»	ADEOfAbstractReliefComponent acts as a hook to define properties within an ADE that are to be added to AbstractReliefComponent.
ADEOfBreaklineRelief «DataType»	ADEOfBreaklineRelief acts as a hook to define properties within an ADE that are to be added to a BreaklineRelief.
ADEOfMassPointRelief «DataType»	ADEOfMassPointRelief acts as a hook to define properties within an ADE that are to be added to a MassPointRelief.
ADEOfRasterRelief «DataType»	ADEOfRasterRelief acts as a hook to define properties within an ADE that are to be added to a RasterRelief.
ADEOfReliefFeature «DataType»	ADEOfReliefFeature acts as a hook to define properties within an ADE that are to be added to a ReliefFeature.
ADEOfTINRelief «DataType»	ADEOfTINRelief acts as a hook to define properties within an ADE that are to be added to a TINRelief.

8.10.3. Additional Information

Additional information about the Relief Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.11. Transportation

Requirements Class	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-transportation	
Target type	Implementation Specification
Dependency	/req/req-class-core

The Transportation module defines central elements of the traffic infrastructure. This includes the transportation objects road, track, and square for the movement of vehicles, bicycles, and pedestrians, the transportation object railway for the movement of wheeled vehicles on rails, as well as the transportation object waterway for the movement of vessels upon or within water bodies. The transportation objects are represented in the UML model by the top-level feature types *Road*, *Track*, *Square*, *Railway*, and *Waterway*, which are the main classes of the Transportation module. Transportation objects can be subdivided into sections, which can be regular road, track or railway legs, into intersection areas, and into roundabouts.

For each transportation object, traffic spaces and auxiliary traffic spaces can be provided, which are bounded at the bottom by traffic areas and auxiliary traffic areas, respectively. Traffic areas are elements that are important in terms of traffic usage, such as driving lanes, sidewalks, and cycle lanes, whereas auxiliary traffic areas describe further elements, such as kerbstones, middle lanes, and green areas. The corresponding spaces define the free space above the areas. In addition, each traffic space can have an optional clearance space. The transportation objects can be represented in different levels of granularity, either as a single area, split up into individual lanes or even decomposed into individual (carriage)ways. Furthermore, holes in the surfaces of roads, tracks or squares, such as road damages, manholes or drains, can be represented including their

corresponding boundary surfaces. In addition, markings for the structuring or restriction of traffic can be added to the transportation areas. Examples are road markings and markings related to railway or waterway traffic.

The UML diagram of the Transportation module is depicted in [Figure 42](#). A detailed discussion of this Requirements Class can be found in the [CityGML 3.0 Users Guide](#).

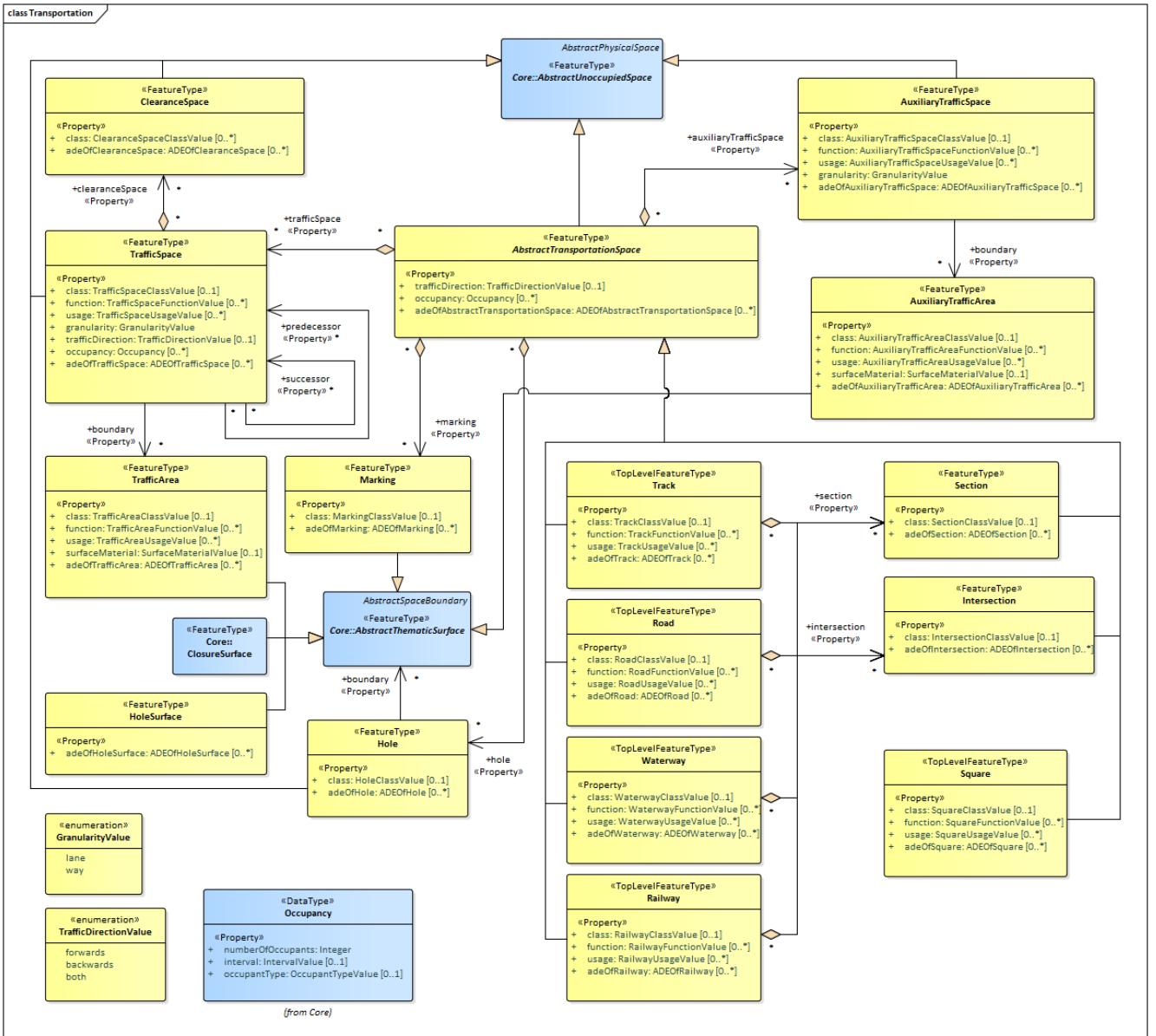


Figure 42. UML diagram of the Transportation Model.

The ADE data types provided for the Transportation module are illustrated in the figure [Figure 43](#).

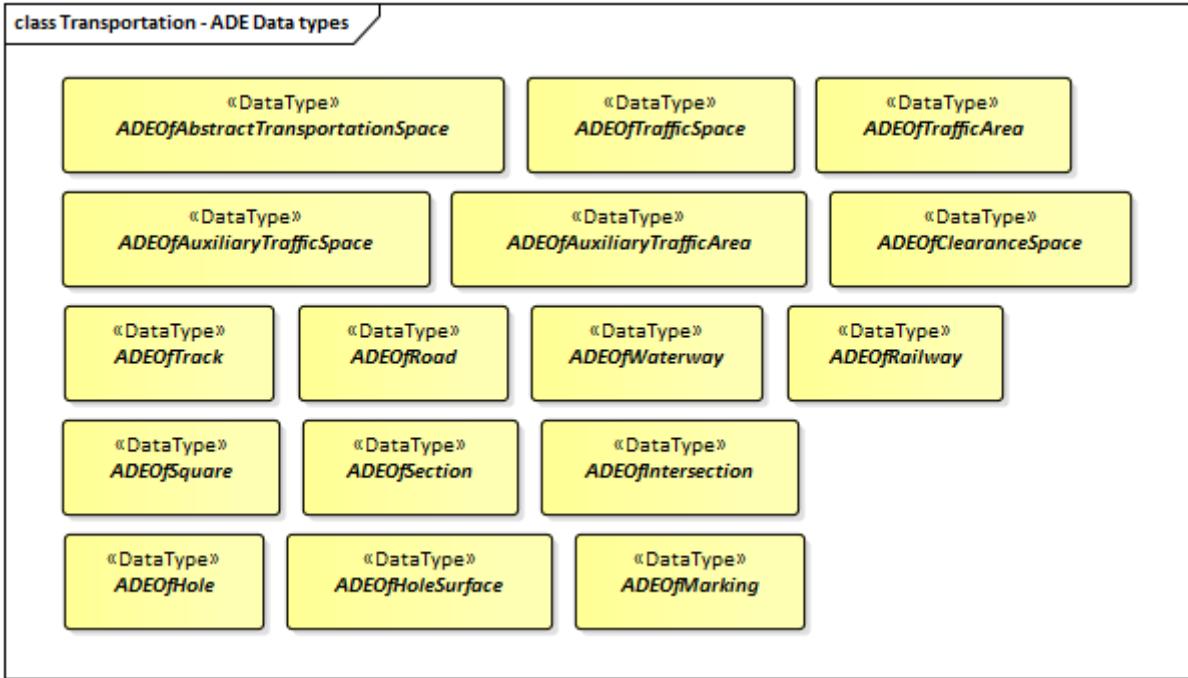


Figure 43. ADE classes of the CityGML Transportation module.

The Code Lists provided for the Transportation module are illustrated in the figure Figure 44.

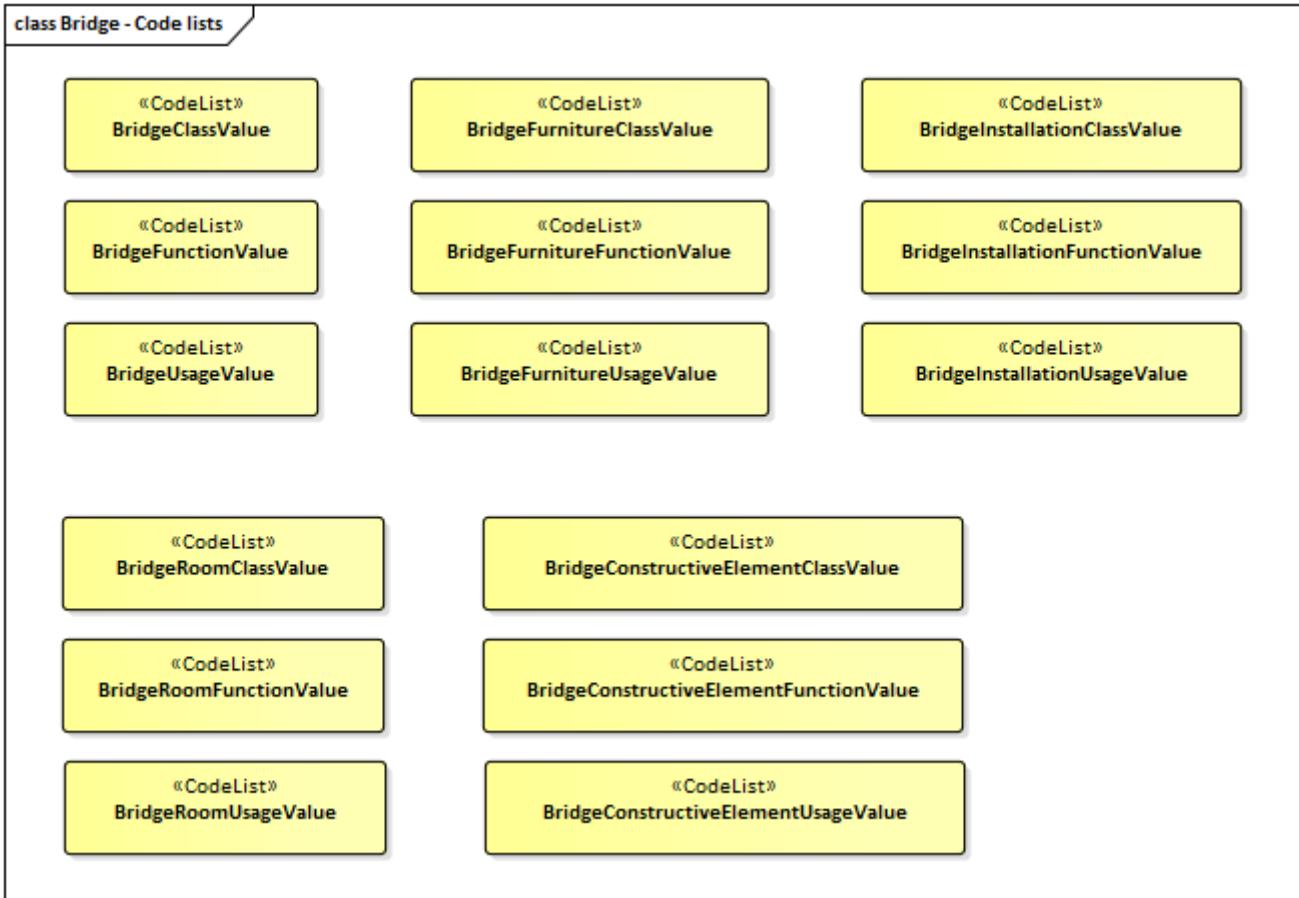


Figure 44. Codelists from the CityGML Transportation module.

Table 41 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Transportation module:

Table 41. Transportation space classes and their allowed thematic surface boundaries

Space class	Allowed space boundaries
AbstractTransportation Space	<ul style="list-style-type: none"> • Transportation::Marking • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
AuxiliaryTrafficSpace	<ul style="list-style-type: none"> • Transportation::AuxiliaryTrafficArea • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
ClearanceSpace	<ul style="list-style-type: none"> • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
Hole	<ul style="list-style-type: none"> • Transportation::HoleSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
Intersection	<ul style="list-style-type: none"> • Transportation::Marking • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
Railway	<ul style="list-style-type: none"> • Transportation::Marking • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
Road	<ul style="list-style-type: none"> • Transportation::Marking • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs

Section	<ul style="list-style-type: none"> • Transportation::Marking • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
Square	<ul style="list-style-type: none"> • Transportation::Marking • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
Track	<ul style="list-style-type: none"> • Transportation::Marking • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
TrafficSpace	<ul style="list-style-type: none"> • Transportation::TrafficArea • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
Waterway	<ul style="list-style-type: none"> • Transportation::Marking • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs

8.11.1. Requirements

The following requirement defines the rules governing implementation of the CityGML Transportation Module as an Implementation Specification.

Requirement 25 /req/transportation/classes	
For each UML class defined or referenced in the Transportation Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.

C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and mutiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The implementation of this Module is further constrained by the following spatial boundary requirement:

Requirement 26	/req/transportation/boundaries
Table 41 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Transportation module. An Implementation Specification SHALL NOT specify boundaries except as specified in Table 41	

The use of extension capabilities by Transportation elements is constrained by the following requirement:

Requirement 27	/req/Transportation/ade/use
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

8.11.2. Class Definitions

Table 42. Classes used in Transportation

Class	Description
Railway «TopLevelFeatureType»	A Railway is a transportation space used by wheeled vehicles on rails.
Road «TopLevelFeatureType»	A Road is a transportation space used by vehicles, bicycles and/or pedestrians.
Square «TopLevelFeatureType»	A Square is a transportation space for unrestricted movement for vehicles, bicycles and/or pedestrians. This includes plazas as well as large sealed surfaces such as parking lots.
Track «TopLevelFeatureType»	A Track is a small path mainly used by pedestrians. Tracks can be segmented into Sections and Intersections.
Waterway «TopLevelFeatureType»	A Waterway is a transportation space used for the movement of vessels upon or within a water body.

AbstractTransportationSpace «FeatureType»	AbstractTransportationSpace is the abstract superclass of transportation objects such as Roads, Tracks, Railways, Waterways or Squares.
AuxiliaryTrafficArea «FeatureType»	An AuxiliaryTrafficArea is the ground surface of an AuxiliaryTrafficSpace.
AuxiliaryTrafficSpace «FeatureType»	An AuxiliaryTrafficSpace is a space within the transportation space not intended for traffic purposes.
ClearanceSpace «FeatureType»	A ClearanceSpace represents the actual free space above a TrafficArea within which a mobile object can move without contacting an obstruction.
Hole «FeatureType»	A Hole is an opening in the surface of a Road, Track or Square such as road damages, manholes or drains. Holes can span multiple transportation objects.
HoleSurface «FeatureType»	A HoleSurface is a representation of the ground surface of a hole.
Intersection «FeatureType»	An Intersection is a transportation space that is a shared segment of multiple Road, Track, Railway, or Waterway objects (e.g. a crossing of two roads or a level crossing of a road and a railway).
Marking «FeatureType»	A Marking is a visible pattern on a transportation area relevant to the structuring or restriction of traffic. Examples are road markings and markings related to railway or waterway traffic.
Section «FeatureType»	A Section is a transportation space that is a segment of a Road, Railway, Track, or Waterway.
TrafficArea «FeatureType»	A TrafficArea is the ground surface of a TrafficSpace. Traffic areas are the surfaces upon which traffic actually takes place.
TrafficSpace «FeatureType»	A TrafficSpace is a space in which traffic takes place. Traffic includes the movement of entities such as trains, vehicles, pedestrians, ships, or other transportation types.

Table 43. Data Types used in Transportation

Name	Description
ADEOfAbstractTransportationSpace «DataType»	ADEOfAbstractTransportationSpace acts as a hook to define properties within an ADE that are to be added to AbstractTransportationSpace.
ADEOfAuxiliaryTrafficArea «DataType»	ADEOfAuxiliaryTrafficArea acts as a hook to define properties within an ADE that are to be added to an AuxiliaryTrafficArea.
ADEOfAuxiliaryTrafficSpace «DataType»	ADEOfAuxiliaryTrafficSpace acts as a hook to define properties within an ADE that are to be added to an AuxiliaryTrafficSpace.

ADEOfClearanceSpace «DataType»	ADEOfClearanceSpace acts as a hook to define properties within an ADE that are to be added to a ClearanceSpace.
ADEOfHole «DataType»	ADEOfHole acts as a hook to define properties within an ADE that are to be added to a Hole.
ADEOfHoleSurface «DataType»	ADEOfHoleSurface acts as a hook to define properties within an ADE that are to be added to a HoleSurface.
ADEOfIntersection «DataType»	ADEOfIntersection acts as a hook to define properties within an ADE that are to be added to an Intersection.
ADEOfMarking «DataType»	ADEOfMarking acts as a hook to define properties within an ADE that are to be added to a Marking.
ADEOfRailway «DataType»	ADEOfRailway acts as a hook to define properties within an ADE that are to be added to a Railway.
ADEOfRoad «DataType»	ADEOfRoad acts as a hook to define properties within an ADE that are to be added to a Road.
ADEOfSection «DataType»	ADEOfSection acts as a hook to define properties within an ADE that are to be added to a Section.
ADEOfSquare «DataType»	ADEOfSquare acts as a hook to define properties within an ADE that are to be added to a Square.
ADEOfTrack «DataType»	ADEOfTrack acts as a hook to define properties within an ADE that are to be added to a Track.
ADEOfTrafficArea «DataType»	ADEOfTrafficArea acts as a hook to define properties within an ADE that are to be added to a TrafficArea.
ADEOfTrafficSpace «DataType»	ADEOfTrafficSpace acts as a hook to define properties within an ADE that are to be added to a TrafficSpace.
ADEOfWaterway «DataType»	ADEOfWaterway acts as a hook to define properties within an ADE that are to be added to a Waterway.

Table 44. Enumerated Classes used in Transportation

Name	Description
GranularityValue «Enumeration»	GranularityValue enumerates the different levels of granularity in which transportation objects are represented.
TrafficDirectionValue «Enumeration»	TrafficDirectionValue enumerates the allowed directions of travel of a mobile object.

Table 45. CodeList Classes used in Transportation

Name	Description
AuxiliaryTrafficAreaClassValue «CodeList»	AuxiliaryTrafficAreaClassValue is a code list used to further classify an AuxiliaryTrafficArea.

AuxiliaryTrafficAreaFunctionValue «CodeList»	AuxiliaryTrafficAreaFunctionValue is a code list that enumerates the different purposes of an AuxiliaryTrafficArea.
AuxiliaryTrafficAreaUsageValue «CodeList»	AuxiliaryTrafficAreaUsageValue is a code list that enumerates the different uses of an AuxiliaryTrafficArea.
AuxiliaryTrafficSpaceClassValue «CodeList»	AuxiliaryTrafficSpaceClassValue is a code list used to further classify an AuxiliaryTrafficSpace.
AuxiliaryTrafficSpaceFunctionValue «CodeList»	AuxiliaryTrafficSpaceFunctionValue is a code list that enumerates the different purposes of an AuxiliaryTrafficSpace.
AuxiliaryTrafficSpaceUsageValue «CodeList»	AuxiliaryTrafficSpaceUsageValue is a code list that enumerates the different uses of an AuxiliaryTrafficSpace.
ClearanceSpaceClassValue «CodeList»	ClearanceSpaceClassValue is a code list used to further classify a ClearanceSpace.
HoleClassValue «CodeList»	HoleClassValue is a code list used to further classify a Hole.
IntersectionClassValue «CodeList»	IntersectionClassValue is a code list used to further classify an Intersection.
MarkingClassValue «CodeList»	MarkingClassValue is a code list used to further classify a Marking.
RailwayClassValue «CodeList»	RailwayClassValue is a code list used to further classify a Railway.
RailwayFunctionValue «CodeList»	RailwayFunctionValue is a code list that enumerates the different purposes of a Railway.
RailwayUsageValue «CodeList»	RailwayUsageValue is a code list that enumerates the different uses of a Railway.
RoadClassValue «CodeList»	RoadClassValue is a code list used to further classify a Road.
RoadFunctionValue «CodeList»	RoadFunctionValue is a code list that enumerates the different purposes of a Road.
RoadUsageValue «CodeList»	RoadUsageValue is a code list that enumerates the different uses of a Road.
SectionClassValue «CodeList»	SectionClassValue is a code list used to further classify a Section.
SquareClassValue «CodeList»	SquareClassValue is a code list used to further classify a Square.

SquareFunctionValue «CodeList»	SquareFunctionValue is a code list that enumerates the different purposes of a Square.
SquareUsageValue «CodeList»	SquareUsageValue is a code list that enumerates the different uses of a Square.
SurfaceMaterialValue «CodeList»	SurfaceMaterialValue is a code list that enumerates the different surface materials.
TrackClassValue «CodeList»	TrackClassValue is a code list used to further classify a Track.
TrackFunctionValue «CodeList»	TrackFunctionValue is a code list that enumerates the different purposes of a Track.
TrackUsageValue «CodeList»	TrackUsageValue is a code list that enumerates the different uses of a Track.
TrafficAreaClassValue «CodeList»	TrafficAreaClassValue is a code list used to further classify a TrafficArea.
TrafficAreaFunctionValue «CodeList»	TrafficAreaFunctionValue is a code list that enumerates the different purposes of a TrafficArea.
TrafficAreaUsageValue «CodeList»	TrafficAreaUsageValue is a code list that enumerates the different uses of a TrafficArea.
TrafficSpaceClassValue «CodeList»	TrafficSpaceClassValue is a code list used to further classify a TrafficSpace.
TrafficSpaceFunctionValue «CodeList»	TrafficSpaceFunctionValue is a code list that enumerates the different purposes of a TrafficSpace.
TrafficSpaceUsageValue «CodeList»	TrafficSpaceUsageValue is a code list that enumerates the different uses of a TrafficSpace.
WaterwayClassValue «CodeList»	WaterwayClassValue is a code list used to further classify a Waterway.
WaterwayFunctionValue «CodeList»	WaterwayFunctionValue is a code list that enumerates the different purposes of a Waterway.
WaterwayUsageValue «CodeList»	WaterwayUsageValue is a code list that enumerates the different uses of a Waterway.

8.11.3. Additional Information

Additional information about the Transportation Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.12. Vegetation

Requirements Class

<http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-vegetation>

Target type	Implementation Specification
Dependency	/req/req-class-core

The Vegetation module defines the concepts to represent vegetation within city models. Vegetation can be represented either as solitary vegetation objects, such as trees, bushes and ferns, or as vegetation areas that are covered by plants of a given species or a typical mixture of plant species, such as forests, steppes and wet meadows. Vegetation is represented in the UML model by the top-level feature types *SolitaryVegetationObject* and *PlantCover*, which are also the only classes of the Vegetation module.

The UML diagram of the Vegetation module is depicted in [Figure 45](#). A detailed discussion of this Requirements Class can be found in the [CityGML User Guide](#).

[Vegetation] | *figures/Vegetation.png*

Figure 45. UML diagram of the Vegetation Model.

The ADE data types provided for the Vegetation module are illustrated in the figure [Figure 46](#).

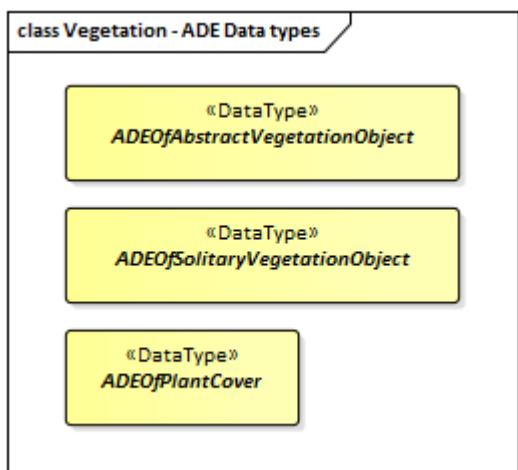


Figure 46. ADE classes of the CityGML Vegetation module.

The Code Lists provided for the Vegetation module are illustrated in the figure [Figure 47](#).

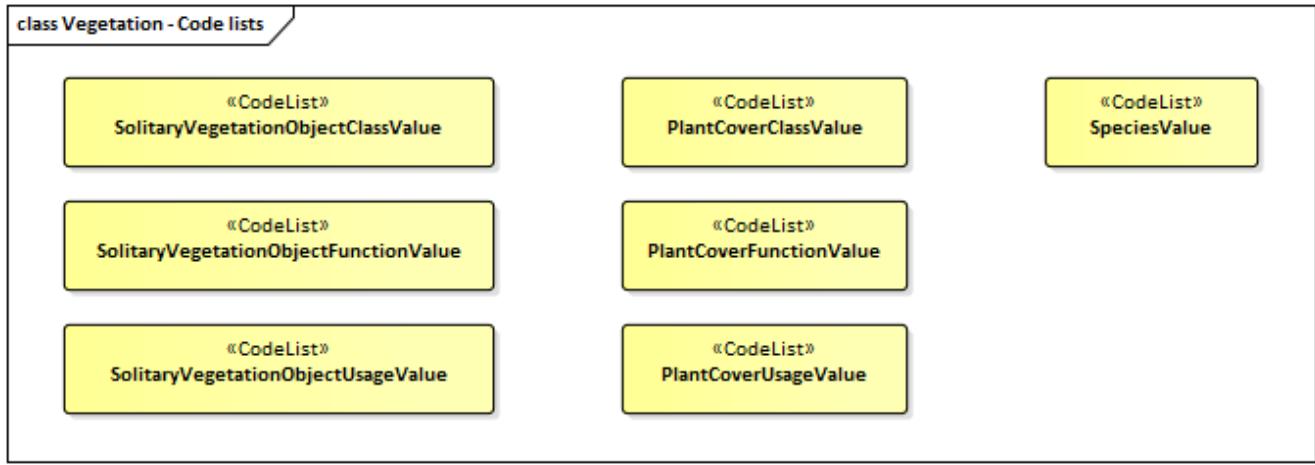


Figure 47. Codelists from the CityGML Vegetation module.

Table 46 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Vegetation module.

Table 46. Vegetation space classes and their allowed thematic surface boundaries

Space class	Allowed space boundaries
AbstractVegetationObject	No boundaries allowed
PlantCover	No boundaries allowed
SolitaryVegetationObject	No boundaries allowed

8.12.1. Requirements

The following requirement defines the rules governing implementation of the CityGML Vegetation Module as an Implementation Specification.

Requirement 28	/req/vegetation/classes
For each UML class defined or referenced in the Vegetation Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and mutiplicity.

E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The implementation of this Module is further constrained by the following spatial boundary requirement:

Requirement 29	/req/vegetation/boundaries
Table 46 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Vegetation module. An Implementation Specification SHALL NOT specify boundaries except as specified in Table 46	

The use of extension capabilities by Vegetation elements is constrained by the following requirement:

Requirement 30	/req/Vegetation/ade/use
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

8.12.2. Class Definitions

Table 47. Classes used in Vegetation

Class	Description
<code>PlantCover</code> «TopLevelFeatureType»	A PlantCover represents a space covered by vegetation.
<code>SolitaryVegetationObject</code> «TopLevelFeatureType»	A SolitaryVegetationObject represents individual vegetation objects, e.g. trees or bushes.
<code>AbstractVegetationObject</code> «FeatureType»	AbstractVegetationObject is the abstract superclass for all kinds of vegetation objects.

Table 48. Data Types used in Vegetation

Name	Description
<code>ADEOfAbstractVegetationObject</code> «DataType»	ADEOfAbstractVegetationObject acts as a hook to define properties within an ADE that are to be added to AbstractVegetationObject.
<code>ADEOfPlantCover</code> «DataType»	ADEOfPlantCover acts as a hook to define properties within an ADE that are to be added to a PlantCover.

ADEOfSolitaryVegetationObject «DataType»	ADEOfSolitaryVegetationObject acts as a hook to define properties within an ADE that are to be added to a SolitaryVegetationObject.
---	---

Table 49. *CodeList Classes used in Vegetation*

Name	Description
PlantCoverClassValue «CodeList»	PlantCoverClassValue is a code list used to further classify a PlantCover.
PlantCoverFunctionValue «CodeList»	PlantCoverFunctionValue is a code list that enumerates the different purposes of a PlantCover.
PlantCoverUsageValue «CodeList»	PlantCoverUsageValue is a code list that enumerates the different uses of a PlantCover.
SolitaryVegetationObjectClassValue «CodeList»	SolitaryVegetationObjectClassValue is a code list used to further classify a SolitaryVegetationObject.
SolitaryVegetationObjectFunctionValue «CodeList»	SolitaryVegetationObjectFunctionValue is a code list that enumerates the different purposes of a SolitaryVegetationObject.
SolitaryVegetationObjectUsageValue «CodeList»	SolitaryVegetationObjectUsageValue is a code list that enumerates the different uses of a SolitaryVegetationObject.
SpeciesValue «CodeList»	A SpeciesValue is a code list that enumerates the species of a SolitaryVegetationObject.

8.12.3. Additional Information

Additional information about the Transportation Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.13. Versioning

Requirements Class	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-versioning	
Target type	Implementation Specification
Dependency	/req/req-class-core

The Versioning module provides the concepts that allow for representing multiple versions of a city model. A specific version represents a defined state of a city model consisting of the dedicated versions of all city object instances that belong to the respective city model version. Each version can be complemented by version transitions that describe the change of the state of a city model from one version to another and that give the reason for the change and the modifications applied. In addition, the Versioning module introduces bitemporal timestamps for all objects. This allows for

providing all objects with information on 1) the time period a specific version of an object is an integral part of the 3D city model and 2) the lifespan a specific version of an object exists in the real world.

By using the Versioning module, slow changes over a long time period with respect to cities and city models can be represented. This includes the creation and termination of objects (e.g. construction or demolition of sites, planting of trees, construction of new roads), structural changes of objects (e.g. raising of buildings), and changes in the status of an object (e.g. change of building owner, change of the traffic direction of a road to a one-way street). In this way, the history or evolution of cities and city models can be modelled, parallel or alternative versions of cities and city models can be managed, and changes of geometries and thematic properties of individual city objects over time can be tracked.

The UML diagram of the Versioning module is depicted in [Figure 48](#). A detailed discussion of this Requirements Class can be found in the [CityGML User Guide](#).

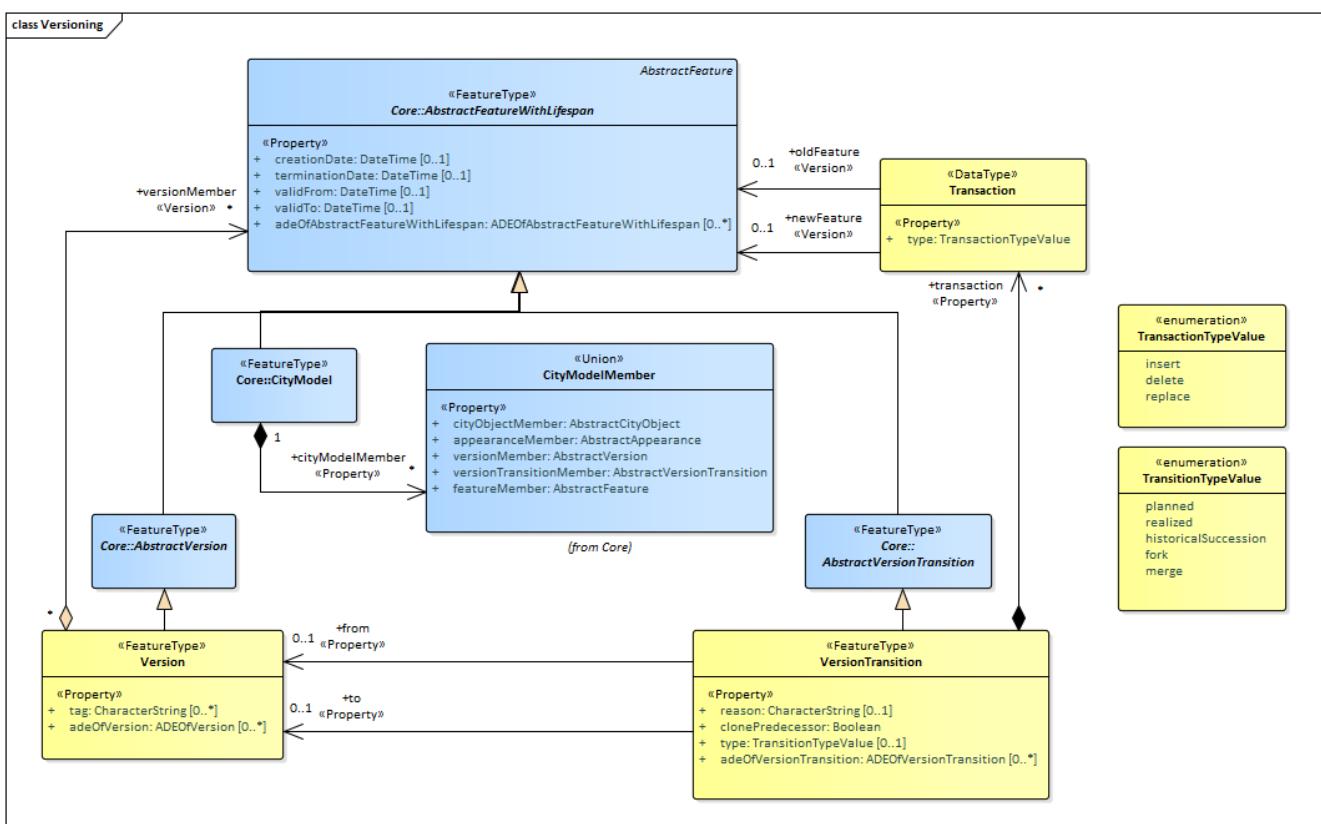


Figure 48. UML diagram of the Versioning Model.

The ADE data types provided for the Versioning module are illustrated in the figure [Figure 49](#).

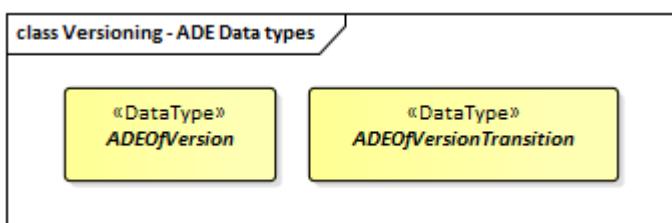


Figure 49. ADE classes of the CityGML Versioning module.

8.13.1. Requirements

The following requirement defines the rules governing implementation of the CityGML Versioning Module as an Implementation Specification.

Requirement 31 /req/versioning/classes	
For each UML class defined or referenced in the Versioning Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and mutiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The use of extension capabilities by Versioning elements is constrained by the following requirement:

Requirement 32 /req/Versioning/ade/use	
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

8.13.2. Class Definitions

Table 50. Classes used in Versioning

Class	Description
Version «FeatureType»	Version represents a defined state of a city model consisting of the dedicated versions of all city object instances that belong to the respective city model version. Versions can have names, a description and can be labeled with an arbitrary number of user defined tags.
VersionTransition «FeatureType»	VersionTransition describes the change of the state of a city model from one version to another. Version transitions can have names, a description and can be further qualified by a type and a reason.

Table 51. Data Types used in Versioning

Name	Description
ADEOfVersion «DataType»	ADEOfVersion acts as a hook to define properties within an ADE that are to be added to a Version.
ADEOfVersionTransitio n «DataType»	ADEOfVersionTransition acts as a hook to define properties within an ADE that are to be added to a VersionTransition.
Transaction «DataType»	Transaction represents a modification of the city model by the creation, termination, or replacement of a specific city object. While the creation of a city object also marks its first object version, the termination marks the end of existence of a real world object and, hence, also terminates the final version of a city object. The replacement of a city object means that a specific version of it is replaced by a new version.

Table 52. Enumerated Classes used in Versioning

Name	Description
TransactionTypeValue «Enumeration»	TransactionTypeValue enumerates the three possible types of transactions: insert, delete, or replace.
TransitionTypeValue «Enumeration»	TransitionTypeValue enumerates the different kinds of version transitions. “planned” and “fork” should be used in cases when from one city model version multiple successor versions are being created. “realized” and “merge” should be used when different city model versions are converging into a common successor version.

8.13.3. Additional Information

Additional information about the Transportation Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.14. Water Body

Requirements Class	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-waterbody	
Target type	Implementation Specification
Dependency	/req/req-class-core

The WaterBody module provides the representation of significant and permanent or semi-permanent accumulations of surface water, usually covering a part of the Earth. Examples of such water bodies that can be modelled with CityGML are rivers, canals, lakes, and basins. Water bodies are represented in the UML model by the top-level feature type *WaterBody*, which is the main class of the WaterBody module.

Water bodies can be bounded by water surfaces, which represent the upper exterior interface between the water body and the atmosphere, and by water ground surfaces, which represent the

exterior boundary surfaces of the submerged bottom of a water body (e.g. DTM or floor of a 3D basin object). Water surfaces are dynamic surfaces, thus, the visible water surface can regularly as well as irregularly change in height and covered area due to natural forces such as tides and floods.

The UML diagram of the WaterBody module is depicted in [Figure 50](#). A detailed discussion of this Requirements Class can be found in the [CityGML User Guide](#).

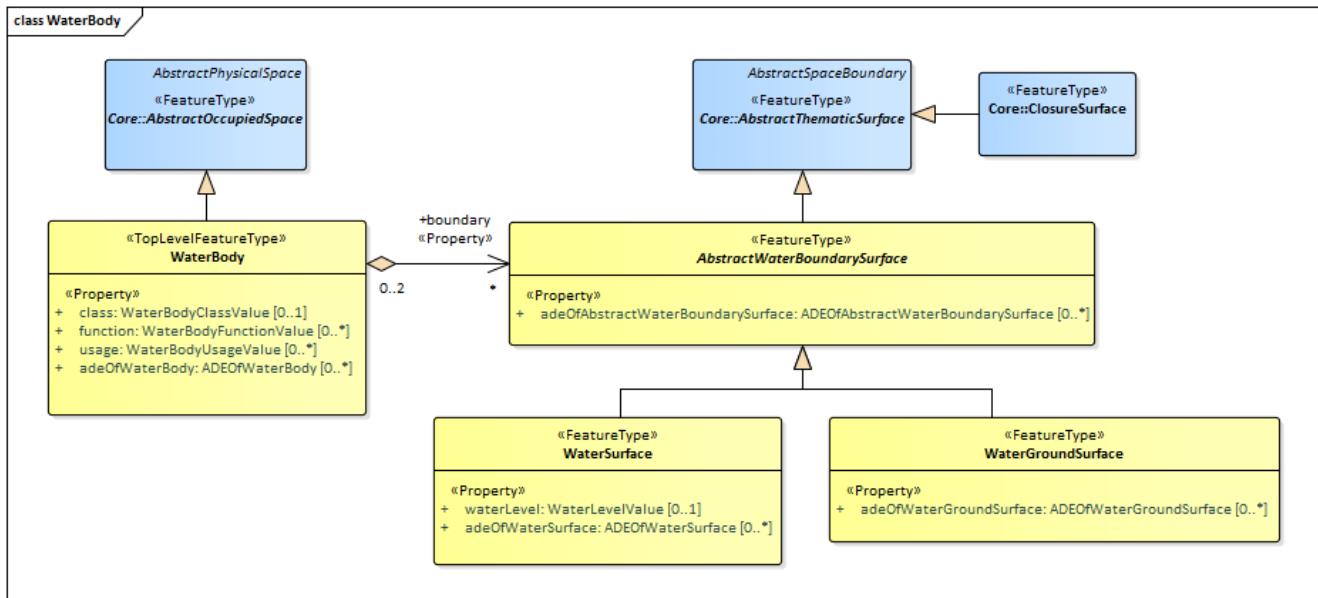


Figure 50. UML diagram of the Water Body Model.

The ADE data types provided for the Water Body module are illustrated in the figure [Figure 51](#).

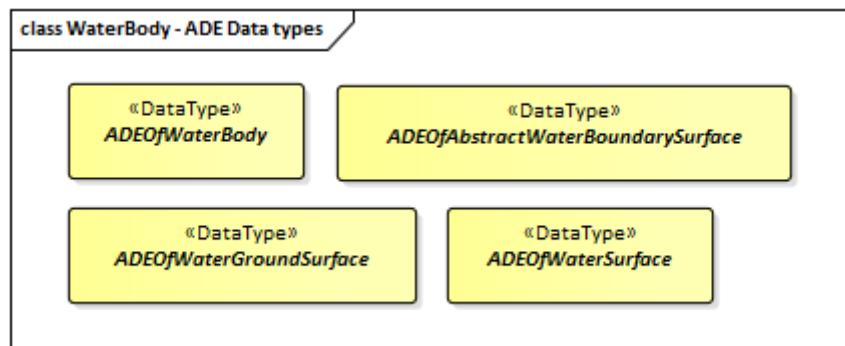


Figure 51. ADE classes of the CityGML Water Body module.

The Code Lists provided for the Water Body module are illustrated in the figure [Figure 52](#).

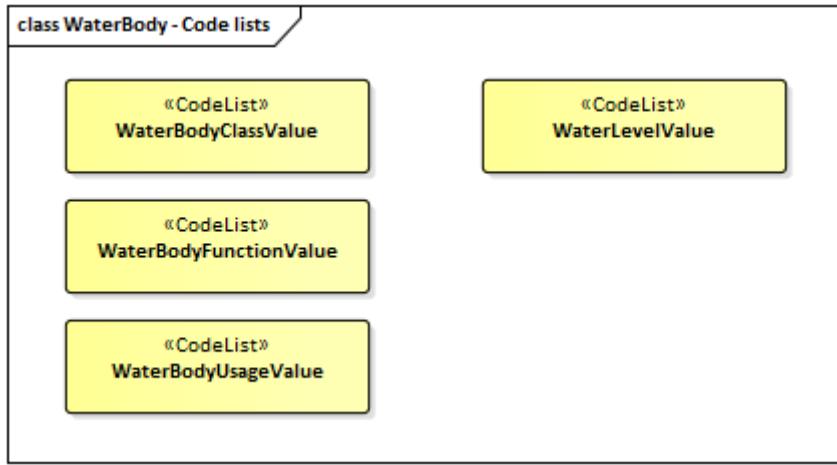


Figure 52. Codelists from the CityGML Water Body module.

[Table 53](#) lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the WaterBody module:

Table 53. WaterBody space classes and their allowed thematic surface boundaries

Space class	Allowed space boundaries
WaterBody	<ul style="list-style-type: none"> • WaterBody::AbstractWaterBoundarySurface and all subclasses, i.e. WaterBody::WaterGroundSurface, WaterBody::WaterSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs

8.14.1. Requirements

The following requirement defines the rules governing implementation of the CityGML WaterBody Module as an Implementation Specification.

Requirement 33 /req/waterbody/classes	
For each UML class defined or referenced in the Waterbody Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.

D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and multiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The implementation of this Module is further constrained by the following spatial boundary requirement:

Requirement 34	/req/waterbody/boundaries
Table 53 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Waterbody module. An Implementation Specification SHALL NOT specify boundaries except as specified in Table 53	

The use of extension capabilities by Waterbody elements is constrained by the following requirement:

Requirement 35	/req/Waterbody/ade/use
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

8.14.2. Class Definitions

Table 54. Classes used in WaterBody

Class	Description
WaterBody «TopLevelFeatureType»	A WaterBody represents significant and permanent or semi-permanent accumulations of surface water, usually covering a part of the Earth.
AbstractWaterBoundarySurface «FeatureType»	AbstractWaterBoundarySurface is the abstract superclass for all kinds of thematic surfaces bounding a water body.
WaterGroundSurface «FeatureType»	A WaterGroundSurface represents the exterior boundary surface of the submerged bottom of a water body.
WaterSurface «FeatureType»	A WaterSurface represents the upper exterior interface between a water body and the atmosphere.

Table 55. Data Types used in WaterBody

Name	Description
ADEOfAbstractWaterBoundarySurface «DataType»	ADEOfAbstractWaterBoundarySurface acts as a hook to define properties within an ADE that are to be added to AbstractWaterBoundarySurface.

ADEOfWaterBody «DataType»	ADEOfWaterBody acts as a hook to define properties within an ADE that are to be added to a WaterBody.
ADEOfWaterGroundSurface «DataType»	ADEOfWaterGroundSurface acts as a hook to define properties within an ADE that are to be added to a WaterGroundSurface.
ADEOfWaterSurface «DataType»	ADEOfWaterSurface acts as a hook to define properties within an ADE that are to be added to a WaterSurface.

Table 56. *CodeList Classes used in WaterBody*

Name	Description
WaterBodyClassValue «CodeList»	WaterBodyClassValue is a code list used to further classify a WaterBody.
WaterBodyFunctionValue «CodeList»	WaterBodyFunctionValue is a code list that enumerates the different purposes of a WaterBody.
WaterBodyUsageValue «CodeList»	WaterBodyUsageValue is a code list that enumerates the different uses of a WaterBody.
WaterLevelValue «CodeList»	WaterLevelValue is a code list that enumerates the different levels of a water surface.

8.14.3. Additional Information

Additional information about the WaterBody Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.15. Construction

Requirements Class	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-construction	
Target type	Implementation Specification
Dependency	/req/req-class-core
Dependency	/req/req-class-generics

The Construction module defines concepts that are common to all kinds of constructions. Constructions are objects that are manufactured by humans from construction materials, are connected to earth, and are intended to be permanent. The Construction module focuses on as-built representations of constructions and integrates all concepts that are similar over different types of constructions, in particular buildings, bridges, and tunnels. In addition, for representing man-made structures that are neither buildings, nor bridges, nor tunnels so-called other constructions (e.g. large chimneys or city walls) can be defined.

Furniture, installations, and constructive elements are further concepts that are defined in the Construction module. Installations are permanent parts of a construction that strongly affect the

outer or inner appearance of the construction and that cannot be moved (e.g. balconies, chimneys, or stairs), whereas furniture represent moveable objects of a construction (e.g. tables and chairs). Constructive elements allow for decomposing a construction into volumetric components, such as walls, beams, and slabs. Constructions and constructive elements can be bounded by different types of surfaces. In this way, the outer structure of constructions and constructive elements can be differentiated semantically into wall surfaces, roof surfaces, ground surfaces, outer floor surfaces, and outer ceiling surfaces, whereas the visible surface of interior spaces can be structured into interior wall surfaces, floor surfaces, and ceiling surfaces. Furthermore, the openings of constructions, i.e. windows and doors, can be represented as so-called filling elements including their corresponding filling surfaces.

The UML diagram of the Construction module is depicted in [Figure 53](#). The Contruction module defines concepts that are inherited and, where necessary, are specialized by the modules Building, Bridge, and Tunnel (cf. [Section 8.17](#), [Section 8.16](#), and [Section 8.18](#)). A detailed discussion of the Requirements Class Construction can be found in the [CityGML 3.0 Users Guide](#).

[Construction] | *figures/Construction.png*

Figure 53. UML diagram of the Construction Model.

The ADE data types provided for the Construction module are illustrated in the figure [Figure 54](#).

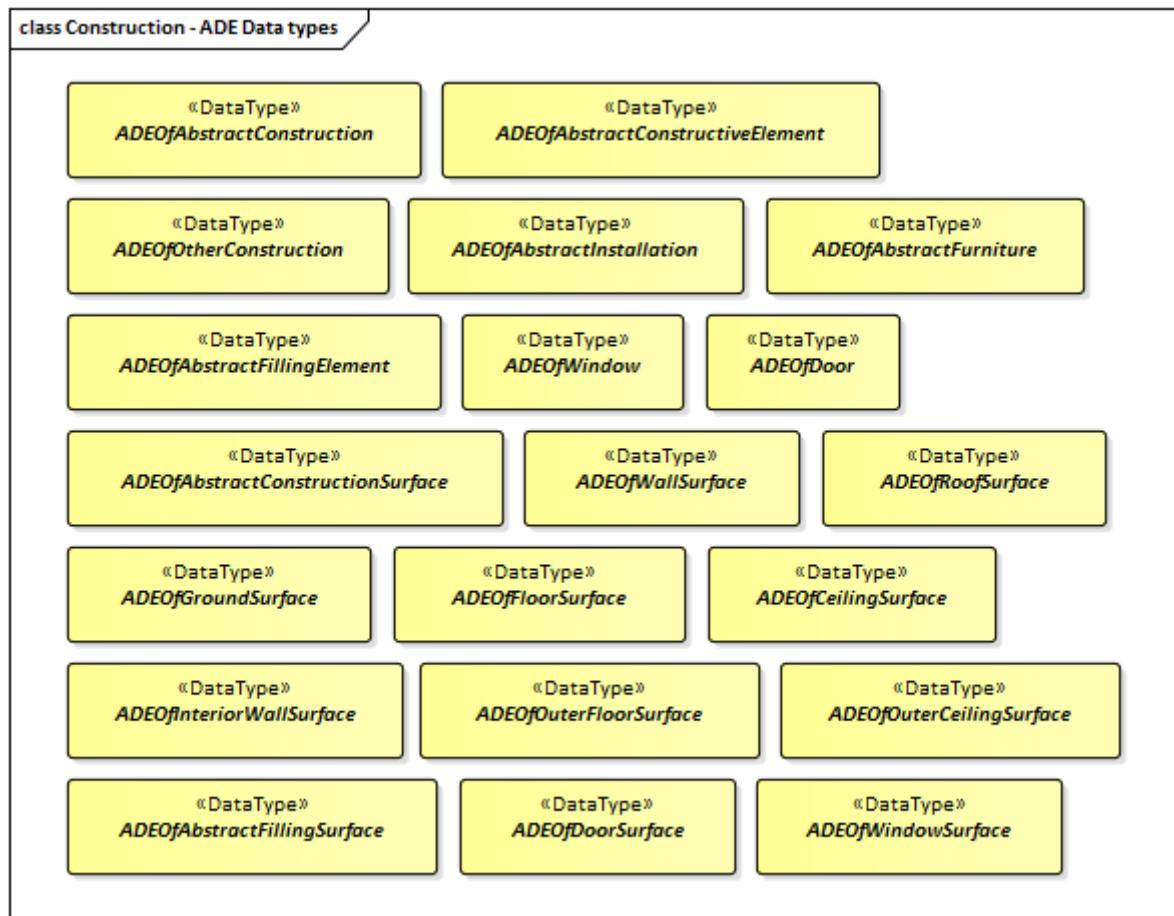


Figure 54. ADE classes of the CityGML Bridge module.

The Code Lists provided for the Construction module are illustrated in the figure [Figure 55](#).

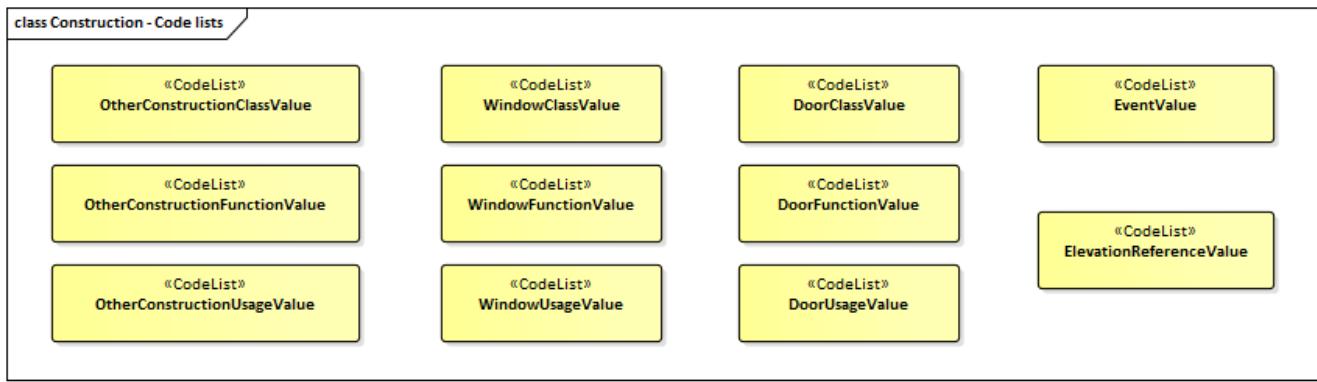


Figure 55. Codelists from the CityGML Bridge module.

[Table 57](#) lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Construction module:

Table 57. Construction space classes and their allowed thematic surface boundaries

Space class	Allowed space boundaries
AbstractConstruction	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
AbstractConstructiveElement	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
AbstractFillingElement	No boundaries allowed

AbstractFurniture	No boundaries allowed
AbstractInstallation	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
Door	<ul style="list-style-type: none"> • Construction::DoorSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
OtherConstruction	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
Window	<ul style="list-style-type: none"> • Construction::WindowSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs

8.15.1. Requirements

The following requirement defines the rules governing implementation of the CityGML

Construction Module as an Implementation Specification.

Requirement 36 /req/construction/classes	
For each UML class defined or referenced in the Construction Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and mutiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The implementation of this Module is further constrained by the following spatial boundary requirement:

Requirement 37 /req/construction/boundaries	
Table 57 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Construction module. An Implementation Specification SHALL NOT specify boundaries except as specified in Table 57	

The use of extension capabilities by Construction elements is constrained by the following requirement:

Requirement 38 /req/Construction/ade/use	
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

8.15.2. Class Definitions

Table 58. Classes used in Construction

Class	Description
OtherConstruction «TopLevelFeatureType»	An OtherConstruction is a construction that is not covered by any of the other subclasses of AbstractConstruction.

AbstractConstruction «FeatureType»	AbstractConstruction is the abstract superclass for objects that are manufactured by humans from construction materials, are connected to earth, and are intended to be permanent. A connection with the ground also exists when the construction rests by its own weight on the ground or is moveable limited on stationary rails or if the construction is intended to be used mainly stationary.
AbstractConstructionSurface «FeatureType»	AbstractConstructionSurface is the abstract superclass for different kinds of surfaces that bound a construction.
AbstractConstructiveElement «FeatureType»	AbstractConstructiveElement is the abstract superclass for the representation of volumetric elements of a construction. Examples are walls, beams, slabs.
AbstractFillingElement «FeatureType»	AbstractFillingElement is the abstract superclass for different kinds of elements that fill the openings of a construction.
AbstractFillingSurface «FeatureType»	AbstractFillingSurface is the abstract superclass for different kinds of surfaces that seal openings filled by filling elements.
AbstractFurniture «FeatureType»	AbstractFurniture is the abstract superclass for the representation of furniture objects of a construction.
AbstractInstallation «FeatureType»	AbstractInstallation is the abstract superclass for the representation of installation objects of a construction.
CeilingSurface «FeatureType»	A CeilingSurface is a surface that represents the interior ceiling of a construction. An example is the ceiling of a room.
Door «FeatureType»	A Door is a construction for closing an opening intended primarily for access or egress or both. [cf. ISO 6707-1]
DoorSurface «FeatureType»	A DoorSurface is either a boundary surface of a Door feature or a surface that seals an opening filled by a door.
FloorSurface «FeatureType»	A FloorSurface is surface that represents the interior floor of a construction. An example is the floor of a room.
GroundSurface «FeatureType»	A GroundSurface is a surface that represents the ground plate of a construction. The polygon defining the ground plate is congruent with the footprint of the construction.
InteriorWallSurface «FeatureType»	An InteriorWallSurface is a surface that is visible from inside a construction. An example is the wall of a room.
OuterCeilingSurface «FeatureType»	An OuterCeilingSurface is a surface that belongs to the outer building shell with the orientation pointing downwards. An example is the ceiling of a loggia.
OuterFloorSurface «FeatureType»	An OuterFloorSurface is a surface that belongs to the outer construction shell with the orientation pointing upwards. An example is the floor of a loggia.
RoofSurface «FeatureType»	A RoofSurface is a surface that delimits major roof parts of a construction.

WallSurface «FeatureType»	A WallSurface is a surface that is part of the building facade visible from the outside.
Window «FeatureType»	A Window is a construction for closing an opening in a wall or roof, primarily intended to admit light and/or provide ventilation. [cf. ISO 6707-1]
WindowSurface «FeatureType»	A WindowSurface is either a boundary surface of a Window feature or a surface that seals an opening filled by a window.

Table 59. Data Types used in Construction

Name	Description
ADEOfAbstractConstruction «DataType»	ADEOfAbstractConstruction acts as a hook to define properties within an ADE that are to be added to AbstractConstruction.
ADEOfAbstractConstructionSurface «DataType»	ADEOfAbstractConstructionSurface acts as a hook to define properties within an ADE that are to be added to AbstractConstructionSurface.
ADEOfAbstractConstructiveElement «DataType»	ADEOfAbstractConstructiveElement acts as a hook to define properties within an ADE that are to be added to AbstractConstructiveElement.
ADEOfAbstractFillingElement «DataType»	ADEOfAbstractFillingElement acts as a hook to define properties within an ADE that are to be added to AbstractFillingElement.
ADEOfAbstractFillingSurface «DataType»	ADEOfAbstractFillingSurface acts as a hook to define properties within an ADE that are to be added to AbstractFillingSurface.
ADEOfAbstractFurniture «DataType»	ADEOfAbstractFurniture acts as a hook to define properties within an ADE that are to be added to AbstractFurniture.
ADEOfAbstractInstallation «DataType»	ADEOfAbstractInstallation acts as a hook to define properties within an ADE that are to be added to AbstractInstallation.
ADEOfCeilingSurface «DataType»	ADEOfCeilingSurface acts as a hook to define properties within an ADE that are to be added to a CeilingSurface.
ADEOfDoor «DataType»	ADEOfDoor acts as a hook to define properties within an ADE that are to be added to a Door.
ADEOfDoorSurface «DataType»	ADEOfDoorSurface acts as a hook to define properties within an ADE that are to be added to a DoorSurface.
ADEOfFloorSurface «DataType»	ADEOfFloorSurface acts as a hook to define properties within an ADE that are to be added to a FloorSurface.
ADEOfGroundSurface «DataType»	ADEOfGroundSurface acts as a hook to define properties within an ADE that are to be added to a GroundSurface.

ADEOfInteriorWallSurface «DataType»	ADEOfInteriorWallSurface acts as a hook to define properties within an ADE that are to be added to an InteriorWallSurface.
ADEOfOtherConstruction «DataType»	ADEOfOtherConstruction acts as a hook to define properties within an ADE that are to be added to an OtherConstruction.
ADEOfOuterCeilingSurface «DataType»	ADEOfOuterCeilingSurface acts as a hook to define properties within an ADE that are to be added to an OuterCeilingSurface.
ADEOfOuterFloorSurface «DataType»	ADEOfOuterFloorSurface acts as a hook to define properties within an ADE that are to be added to an OuterFloorSurface.
ADEOfRoofSurface «DataType»	ADEOfRoofSurface acts as a hook to define properties within an ADE that are to be added to a RoofSurface.
ADEOfWallSurface «DataType»	ADEOfWallSurface acts as a hook to define properties within an ADE that are to be added to a WallSurface.
ADEOfWindow «DataType»	ADEOfWindow acts as a hook to define properties within an ADE that are to be added to a Window.
ADEOfWindowSurface «DataType»	ADEOfWindowSurface acts as a hook to define properties within an ADE that are to be added to a WindowSurface.
ConstructionEvent «DataType»	A ConstructionEvent is a data type used to describe a specific event that is associated with a construction. Examples are the issuing of a building permit or the renovation of a building.
Elevation «DataType»	Elevation is a data type that includes the elevation value itself and information on how this elevation was measured. [cf. INSPIRE]
Height «DataType»	Height represents a vertical distance (measured or estimated) between a low reference and a high reference. [cf. INSPIRE]

Table 60. Enumerated Classes used in Construction

Name	Description
ConditionOfConstructionValue «Enumeration»	ConditionOfConstructionValue enumerates different conditions of a construction. [cf. INSPIRE]
HeightStatusValue «Enumeration»	HeightStatusValue enumerates the different methods used to capture a height. [cf. INSPIRE]
RelationToConstruction «Enumeration»	RelationToConstruction is an enumeration used to describe whether an installation is positioned inside and/or outside of a construction.

Table 61. CodeList Classes used in Construction

Name	Description

DoorClassValue «CodeList»	DoorClassValue is a code list used to further classify a Door.
DoorFunctionValue «CodeList»	DoorFunctionValue is a code list that enumerates the different purposes of a Door.
DoorUsageValue «CodeList»	DoorUsageValue is a code list that enumerates the different uses of a Door.
ElevationReferenceValue «CodeList»	ElevationReferenceValue is a code list that enumerates the different elevation reference levels used to measure construction heights.
EventValue «CodeList»	EventValue is a code list that enumerates the different events of a construction.
OtherConstructionClassName «CodeList»	OtherConstructionClassName is a code list used to further classify an OtherConstruction.
OtherConstructionFunctionValue «CodeList»	OtherConstructionFunctionValue is a code list that enumerates the different purposes of an OtherConstruction.
OtherConstructionUsageValue «CodeList»	OtherConstructionUsageValue is a code list that enumerates the different uses of an OtherConstruction.
WindowClassValue «CodeList»	WindowClassValue is a code list used to further classify a Window.
WindowFunctionValue «CodeList»	WindowFunctionValue is a code list that enumerates the different purposes of a Window.
WindowUsageValue «CodeList»	WindowUsageValue is a code list that enumerates the different uses of a Window.

8.15.3. Additional Information

Additional information about the Construction Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.16. Bridge

Requirements Class	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-bridge	
Target type	Implementation Specification
Dependency	/req/req-class-core
Dependency	/req/req-class-construction

The Bridge module provides the representation of thematic and spatial aspects of bridges. Bridges are movable or unmovable structures that span intervening natural or built elements. In this way,

bridges allow the passage of pedestrians, animals, vehicles, and service(s) above obstacles or between two points at a height above ground. Bridges are represented in the UML model by the top-level feature type *Bridge*, which is the main class of the Bridge module. Bridges can physically or functionally be subdivided into bridge parts. In addition, bridges can be decomposed into structural elements, such as pylons, anchorages, cables, slabs, and beams.

The free space inside bridges is represented by rooms, which allows a virtual accessibility of bridges. Bridges can contain installations and furniture. Installations are permanent parts of a bridge that strongly affect the outer or inner appearance of the bridge and that cannot be moved. Examples are stairways, signals, railings, and lamps. Furniture, in contrast, represent moveable objects of a bridge, like signs, art works, and benches. Bridges can be bounded by different types of surfaces. In this way, the outer structure of bridges can be differentiated semantically into wall surfaces, roof surfaces, ground surfaces, outer floor surfaces, and outer ceiling surfaces, whereas the visible surface of rooms can be structured into interior wall surfaces, floor surfaces, and ceiling surfaces. Furthermore, the openings of bridges, i.e. windows and doors, can be represented including their corresponding surfaces.

The UML diagram of the Bridge module is depicted in [Figure 56](#). The Bridge module inherits concepts from the Construction module (cf. [Section 8.15](#)). The Construction module defines objects that are common to all types of construction, such as the different surface types and the openings. A detailed discussion of Requirements Class *Bridge* can be found in the [CityGML 3.0 Users Guide](#).

[Bridge] | *figures/Bridge.png*

Figure 56. UML diagram of the Bridge Model.

The ADE data types provided for the Bridge module are illustrated in the figure [Figure 57](#).

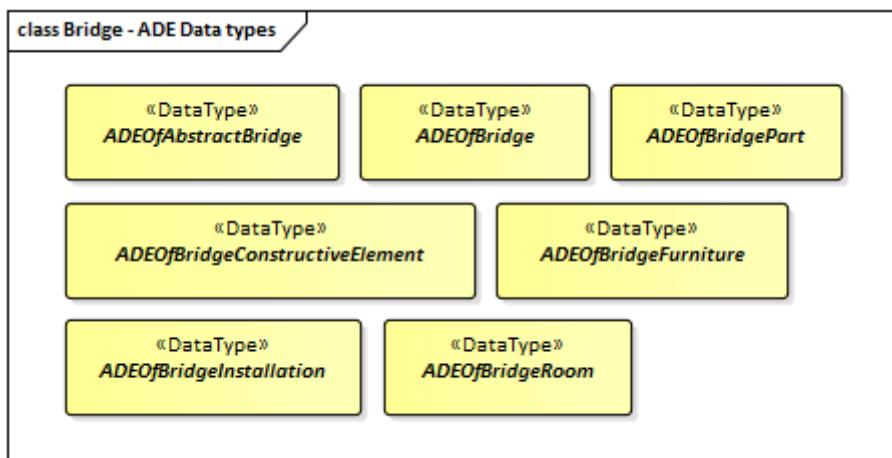


Figure 57. ADE classes of the CityGML Bridge module.

The Code Lists provided for the Bridge module are illustrated in the figure [Figure 58](#).

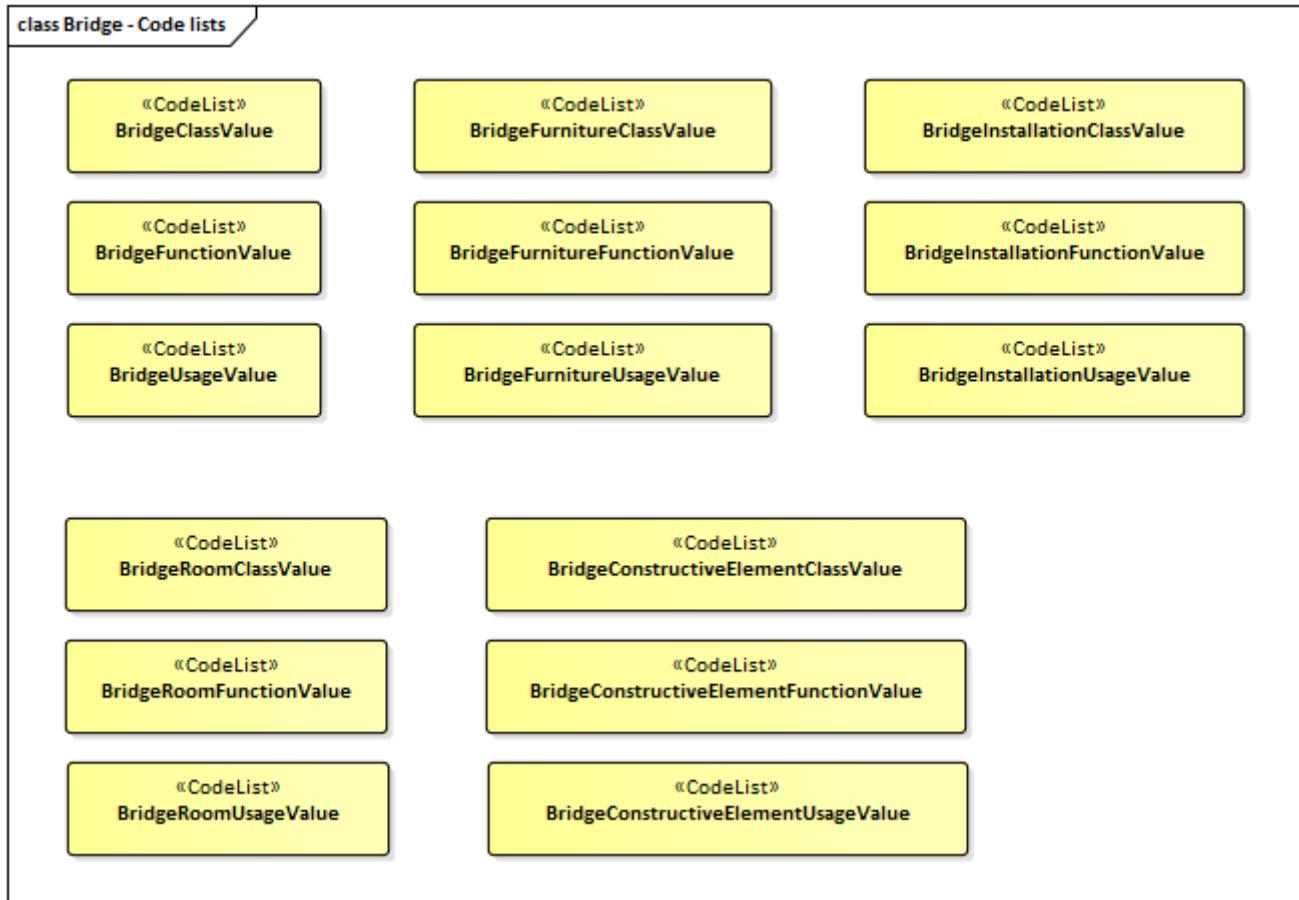


Figure 58. Codelists from the CityGML Bridge module.

[Table 62](#) lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Bridge module:

Table 62. Bridge space classes and their allowed thematic surface boundaries

Space class	Allowed space boundaries
AbstractBridge	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs

Bridge	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
BridgeConstructiveElement	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
BridgeFurniture	No boundaries allowed
BridgeInstallation	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs

BridgePart	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
BridgeRoom	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs

8.16.1. Requirements

The following requirement defines the rules governing implementation of the CityGML Bridge Module as an Implementation Specification.

Requirement 39	/req/bridge/classes
For each UML class defined or referenced in the Bridge Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.

D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and multiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The implementation of this Module is further constrained by the following spatial boundary requirement:

Requirement 40	/req/bridge/boundaries
Table 62 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Bridge module. An Implementation Specification SHALL NOT specify boundaries except as specified in Table 62	

The use of extension capabilities by Bridge elements is constrained by the following requirement:

Requirement 41	/req/Bridge/ade/use
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

8.16.2. Class Definitions

Table 63. Classes used in Bridge

Class	Description
Bridge «TopLevelFeatureType»	A Bridge represents a structure that affords the passage of pedestrians, animals, vehicles, and service(s) above obstacles or between two points at a height above ground. [cf. ISO 6707-1]
AbstractBridge «FeatureType»	AbstractBridge is an abstract superclass representing the common attributes and associations of the classes Bridge and BridgePart.
BridgeConstructiveElement «FeatureType»	A BridgeConstructiveElement is an element of a bridge which is essential from a structural point of view. Examples are pylons, anchorages, slabs, beams.
BridgeFurniture «FeatureType»	A BridgeFurniture is an equipment for occupant use, usually not fixed to the bridge. [cf. ISO 6707-1]
BridgeInstallation «FeatureType»	A BridgeInstallation is a permanent part of a Bridge (inside and/or outside) which does not have the significance of a BridgePart. In contrast to BridgeConstructiveElements, a BridgeInstallation is not essential from a structural point of view. Examples are stairs, antennas or railways.

BridgePart «FeatureType»	A BridgePart is a physical or functional subdivision of a Bridge. It would be considered a Bridge, if it were not part of a collection of other BridgeParts.
BridgeRoom «FeatureType»	A BridgeRoom is a space within a Bridge or BridgePart intended for human occupancy (e.g. a place of work or recreation) and/or containment (storage) of animals or things. A BridgeRoom is bounded physically and/or virtually (e.g. by ClosureSurfaces or GenericSurfaces).

Table 64. Data Types used in Bridge

Name	Description
ADEOfAbstractBridge «DataType»	ADEOfAbstractBridge acts as a hook to define properties within an ADE that are to be added to AbstractBridge.
ADEOfBridge «DataType»	ADEOfBridge acts as a hook to define properties within an ADE that are to be added to a Bridge.
ADEOfBridgeConstructiveElement «DataType»	ADEOfBridgeConstructiveElement acts as a hook to define properties within an ADE that are to be added to a BridgeConstructiveElement.
ADEOfBridgeFurniture «DataType»	ADEOfBridgeFurniture acts as a hook to define properties within an ADE that are to be added to a BridgeFurniture.
ADEOfBridgeInstallation «DataType»	ADEOfBridgeInstallation acts as a hook to define properties within an ADE that are to be added to a BridgeInstallation.
ADEOfBridgePart «DataType»	ADEOfBridgePart acts as a hook to define properties within an ADE that are to be added to a BridgePart.
ADEOfBridgeRoom «DataType»	ADEOfBridgeRoom acts as a hook to define properties within an ADE that are to be added to a BridgeRoom.

Table 65. CodeList Classes used in Bridge

Name	Description
BridgeClassValue «CodeList»	BridgeClassValue is a code list used to further classify a Bridge.
BridgeConstructiveElementClassValue «CodeList»	BridgeConstructiveElementClassValue is a code list used to further classify a BridgeConstructiveElement.
BridgeConstructiveElementFunctionValue «CodeList»	BridgeConstructiveElementFunctionValue is a code list that enumerates the different purposes of a BridgeConstructiveElement.
BridgeConstructiveElementUsageValue «CodeList»	BridgeConstructiveElementUsageValue is a code list that enumerates the different uses of a BridgeConstructiveElement.
BridgeFunctionValue «CodeList»	BridgeFunctionValue is a code list that enumerates the different purposes of a Bridge.

BridgeFurnitureClassValue «CodeList»	BridgeFurnitureClassValue is a code list used to further classify a BridgeFurniture.
BridgeFurnitureFunctionValue «CodeList»	BridgeFurnitureFunctionValue is a code list that enumerates the different purposes of a BridgeFurniture.
BridgeFurnitureUsageValue «CodeList»	BridgeFurnitureUsageValue is a code list that enumerates the different uses of a BridgeFurniture.
BridgeInstallationClassValue «CodeList»	BridgeInstallationClassValue is a code list used to further classify a BridgeInstallation.
BridgeInstallationFunctionValue «CodeList»	BridgeInstallationFunctionValue is a code list that enumerates the different purposes of a BridgeInstallation.
BridgeInstallationUsageValue «CodeList»	BridgeInstallationUsageValue is a code list that enumerates the different uses of a BridgeInstallation.
BridgeRoomClassValue «CodeList»	BridgeRoomClassValue is a code list used to further classify a BridgeRoom.
BridgeRoomFunctionValue «CodeList»	BridgeRoomFunctionValue is a code list that enumerates the different purposes of a BridgeRoom.
BridgeRoomUsageValue «CodeList»	BridgeRoomUsageValue is a code list that enumerates the different uses of a BridgeRoom.
BridgeUsageValue «CodeList»	BridgeUsageValue is a code list that enumerates the different uses of a Bridge.

8.16.3. Additional Information

Additional information about the Bridge Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.17. Building

Requirements Class

<http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-building>

Target type	Implementation Specification
Dependency	/req/req-class-core
Dependency	/req/req-class-construction

The Building module provides the representation of thematic and spatial aspects of buildings. Buildings are free-standing, self-supporting constructions that are roofed and usually walled, and

that can be entered by humans and are normally designed to stand permanently in one place. Buildings are intended for human occupancy (e.g. a place of work or recreation), habitation and/or shelter of humans, animals or things. Buildings are represented in the UML model by the top-level feature type *Building*, which is the main class of the Building module. Buildings can physically or functionally be subdivided into building parts, and logically into storeys and building units (e.g. apartments). In addition, buildings can be decomposed into structural elements, such as walls, slabs, staircases, and beams.

The interior of buildings is represented by rooms. This allows a virtual accessibility of buildings, e.g. for visitor information in a museum (“Location Based Services”), the examination of accommodation standards or the presentation of daylight illumination of a building. Buildings can contain installations and furniture. Installations are permanent parts of a building that strongly affect the outer or inner appearance of the building and that cannot be moved. Examples are balconies, chimneys, dormers or stairs. Furniture, in contrast, represent moveable objects inside a building, like tables and chairs. Buildings can be bounded by different types of surfaces. In this way, the outer façade of buildings can be differentiated semantically into wall surfaces, roof surfaces, ground surfaces, outer floor surfaces, and outer ceiling surfaces, whereas the visible surface of rooms can be structured into interior wall surfaces, floor surfaces, and ceiling surfaces. Furthermore, the openings of buildings, i.e. windows and doors, can be represented including their corresponding surfaces.

The UML diagram of the building module is depicted in [Figure 59](#). The Building module inherits concepts from the Construction module (cf. [Section 8.15](#)). The Construction module defines objects that are common to all types of construction, such as the different surface types and the openings. A detailed discussion of the Requirements Class *Building* can be found in the [CityGML 3.0 Users Guide](#).

[Building] | *figures/Building.png*

Figure 59. UML diagram of CityGML’s building model.

The ADE data types provided for the Building module are illustrated in the figure [Figure 60](#).

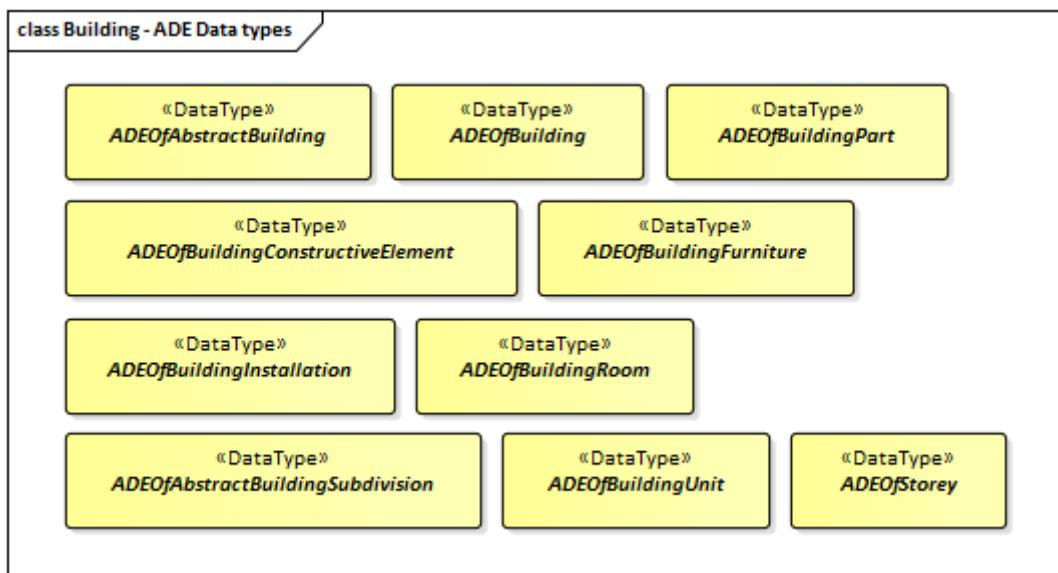


Figure 60. ADE classes of the CityGML Building module.

The Code Lists provided for the Building module are illustrated in the figure [Figure 61](#).

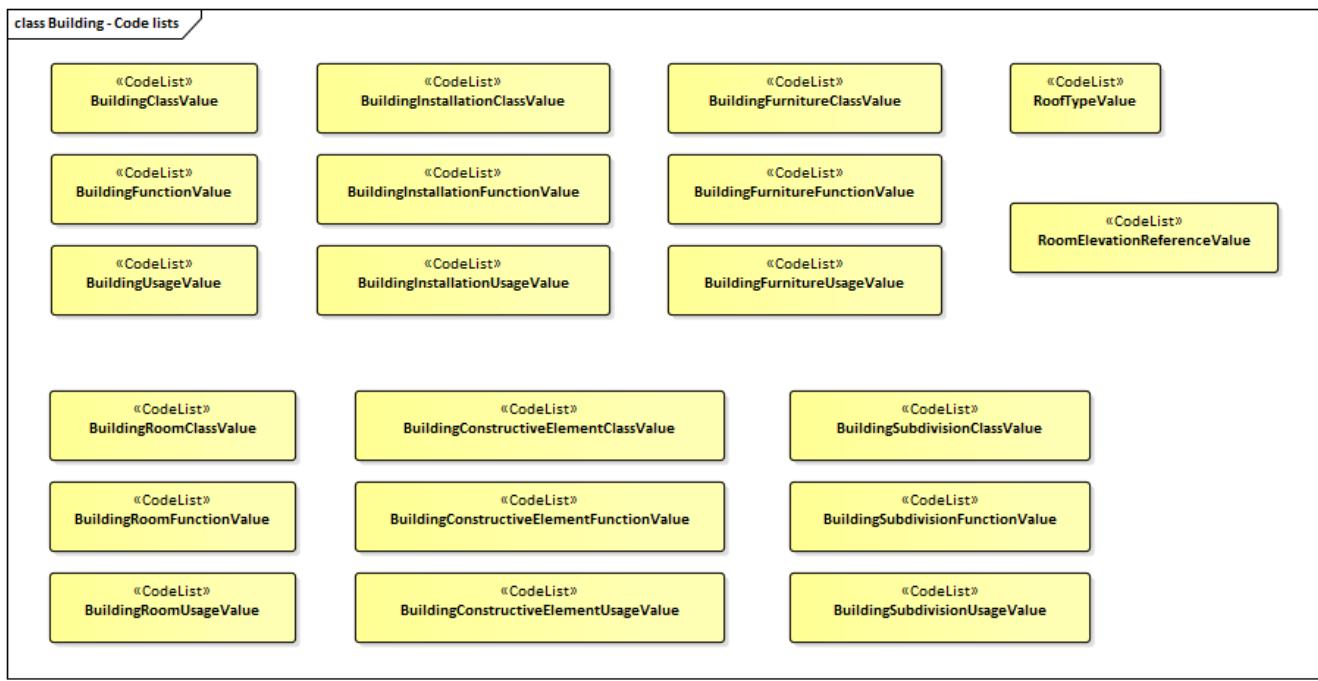


Figure 61. Codelists from the CityGML Building module.

[Table 66](#) lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Building module:

Table 66. Building space classes and their allowed thematic surface boundaries

Space class	Allowed space boundaries
AbstractBuilding	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
AbstractBuildingSubdivision	No boundaries allowed

Building	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
BuildingConstructiveElement	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
BuildingFurniture	No boundaries allowed
BuildingInstallation	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs

BuildingPart	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
BuildingRoom	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
BuildingUnit	<ul style="list-style-type: none"> • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
Storey	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs

8.17.1. Requirements

The following requirement defines the rules governing implementation of the CityGML Building Module as an Implementation Specification.

Requirement 42 /req/building/classes	
For each UML class defined or referenced in the Building Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and mutiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The implementation of this Module is further constrained by the following spatial boundary requirement:

Requirement 43 /req/building/boundaries	
<p>Table 66 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Building module. An Implementation Specification SHALL NOT specify boundaries except as specified in Table 66</p>	

The use of extension capabilities by Building elements is constrained by the following requirement:

Requirement 44 /req/Building/ade/use	
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

8.17.2. Class Definitions

Table 67. Classes used in Building

Class	Description

Building «TopLevelFeatureType»	A Building is a free-standing, self-supporting construction that is roofed, usually walled, and can be entered by humans and is normally designed to stand permanently in one place. It is intended for human occupancy (e.g. a place of work or recreation), habitation and/or shelter of humans, animals or things.
AbstractBuilding «FeatureType»	AbstractBuilding is an abstract superclass representing the common attributes and associations of the classes Building and BuildingPart.
AbstractBuildingSubdivision «FeatureType»	AbstractBuildingSubdivision is the abstract superclass for different kinds of logical building subdivisions.
BuildingConstructiveElement «FeatureType»	A BuildingConstructiveElement is an element of a Building which is essential from a structural point of view. Examples are walls, slabs, staircases, beams.
BuildingFurniture «FeatureType»	A BuildingFurniture is an equipment for occupant use, usually not fixed to the building. [cf. ISO 6707-1]
BuildingInstallation «FeatureType»	A BuildingInstallation is a permanent part of a Building (inside and/or outside) which has not the significance of a BuildingPart. Examples are stairs, antennas, balconies or small roofs.
BuildingPart «FeatureType»	A BuildingPart is a physical or functional subdivision of a Building. It would be considered a Building, if it were not part of a collection of other BuildingParts.
BuildingRoom «FeatureType»	A BuildingRoom is a space within a Building or BuildingPart intended for human occupancy (e.g. a place of work or recreation) and/or containment of animals or things. A BuildingRoom is bounded physically and/or virtually (e.g. by ClosureSurfaces or GenericSurfaces).
BuildingUnit «FeatureType»	A BuildingUnit is a logical subdivision of a Building. BuildingUnits are formed according to some homogeneous property like function, ownership, management, or accessibility. They may be separately sold, rented out, inherited, managed, etc.
Storey «FeatureType»	A Storey is typically a horizontal section of a Building. Storeys are not always defined according to the building structure, but can also be defined according to logical considerations.

Table 68. Data Types used in Building

Name	Description
ADEOfAbstractBuilding «DataType»	ADEOfAbstractBuilding acts as a hook to define properties within an ADE that are to be added to AbstractBuilding.
ADEOfAbstractBuildingSubdivision «DataType»	ADEOfAbstractBuildingSubdivision acts as a hook to define properties within an ADE that are to be added to AbstractBuildingSubdivision.
ADEOfBuilding «DataType»	ADEOfBuilding acts as a hook to define properties within an ADE that are to be added to a Building.

ADEOfBuildingConstructiveElement «DataType»	ADEOfBuildingConstructiveElement acts as a hook to define properties within an ADE that are to be added to a BuildingConstructiveElement.
ADEOfBuildingFurniture «DataType»	ADEOfBuildingFurniture acts as a hook to define properties within an ADE that are to be added to a BuildingFurniture.
ADEOfBuildingInstallation «DataType»	ADEOfBuildingInstallation acts as a hook to define properties within an ADE that are to be added to a BuildingInstallation.
ADEOfBuildingPart «DataType»	ADEOfBuildingPart acts as a hook to define properties within an ADE that are to be added to a BuildingPart.
ADEOfBuildingRoom «DataType»	ADEOfBuildingRoom acts as a hook to define properties within an ADE that are to be added to a BuildingRoom.
ADEOfBuildingUnit «DataType»	ADEOfBuildingUnit acts as a hook to define properties within an ADE that are to be added to a BuildingUnit.
ADEOfStorey «DataType»	ADEOfStorey acts as a hook to define properties within an ADE that are to be added to a Storey.
RoomHeight «DataType»	The RoomHeight represents a vertical distance (measured or estimated) between a low reference and a high reference. [cf. INSPIRE]

Table 69. *CodeList Classes used in Building*

Name	Description
BuildingClassValue «CodeList»	BuildingClassValue is a code list used to further classify a Building.
BuildingConstructiveElementClassValue «CodeList»	BuildingConstructiveElementClassValue is a code list used to further classify a BuildingConstructiveElement.
BuildingConstructiveElementFunctionValue «CodeList»	BuildingConstructiveElementFunctionValue is a code list that enumerates the different purposes of a BuildingConstructiveElement.
BuildingConstructiveElementUsageValue «CodeList»	BuildingConstructiveElementUsageValue is a code list that enumerates the different uses of a BuildingConstructiveElement.
BuildingFunctionValue «CodeList»	BuildingFunctionValue is a code list that enumerates the different purposes of a Building.
BuildingFurnitureClassValue «CodeList»	BuildingFurnitureClassValue is a code list used to further classify a BuildingFurniture.
BuildingFurnitureFunctionValue «CodeList»	BuildingFurnitureFunctionValue is a code list that enumerates the different purposes of a BuildingFurniture.

BuildingFurnitureUsageValue «CodeList»	BuildingFurnitureUsageValue is a code list that enumerates the different uses of a BuildingFurniture.
BuildingInstallationClassValue «CodeList»	BuildingInstallationClassValue is a code list used to further classify a BuildingInstallation.
BuildingInstallationFunctionValue «CodeList»	BuildingInstallationFunctionValue is a code list that enumerates the different purposes of a BuildingInstallation.
BuildingInstallationUsageValue «CodeList»	BuildingInstallationUsageValue is a code list that enumerates the different uses of a BuildingInstallation.
BuildingRoomClassValue «CodeList»	BuildingRoomClassValue is a code list used to further classify a BuildingRoom.
BuildingRoomFunctionValue «CodeList»	BuildingRoomFunctionValue is a code list that enumerates the different purposes of a BuildingRoom.
BuildingRoomUsageValue «CodeList»	BuildingRoomUsageValue is a code list that enumerates the different uses of a BuildingRoom.
BuildingSubdivisionClassValue «CodeList»	BuildingSubdivisionClassValue is a code list used to further classify a BuildingSubdivision.
BuildingSubdivisionFunctionValue «CodeList»	BuildingSubdivisionFunctionValue is a code list that enumerates the different purposes of a BuildingSubdivision.
BuildingSubdivisionUsageValue «CodeList»	BuildingSubdivisionUsageValue is a code list that enumerates the different uses of a BuildingSubdivision.
BuildingUsageValue «CodeList»	BuildingUsageValue is a code list that enumerates the different uses of a Building.
RoofTypeValue «CodeList»	RoofTypeValue is a code list that enumerates different roof types.
RoomElevationReferenceValue «CodeList»	RoomElevationReferenceValue is a code list that enumerates the different elevation reference levels used to measure room heights.

8.17.3. Additional Information

Additional information about the Building Module can be found in the [OGC CityGML 3.0 Users Guide](#)

8.18. Tunnel

Requirements Class

<http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-tunnel>

Target type	Implementation Specification
Dependency	/req/req-class-core
Dependency	/req/req-class-construction

The Tunnel module provides the representation of thematic and spatial aspects of tunnels. Tunnels are horizontal or sloping enclosed passage ways of a certain length, mainly underground or underwater. Tunnels are intended for passing obstacles such as mountains, waterways or other traffic routes by humans, animals or things. Tunnels are represented in the UML model by the top-level feature type *Tunnel*, which is the main class of the Tunnel module. Tunnels can physically or functionally be subdivided into tunnel parts. In addition, tunnels can be decomposed into structural elements, such as walls, slabs, staircases, and beams.

The interior of tunnels is represented by hollow spaces. This allows a virtual accessibility of tunnels, e.g. for driving through a tunnel, for simulating disaster management, or for presenting the light illumination within a tunnel. Tunnels can contain installations and furniture. Installations are permanent parts of a tunnel that strongly affect the outer or inner appearance of the tunnel and that cannot be moved. Examples are stairs, railings, radiators or pipes. Furniture, in contrast, represent moveable objects inside a tunnel, like movable equipment in control areas. Tunnels can be bounded by different types of surfaces. In this way, the outer structure of tunnels can be differentiated semantically into wall surfaces, roof surfaces, ground surfaces, outer floor surfaces, and outer ceiling surfaces, whereas the visible surface of hollow spaces can be structured into interior wall surfaces, floor surfaces, and ceiling surfaces. Furthermore, the openings of tunnels, i.e. windows and doors, can be represented including their corresponding surfaces.

The UML diagram of the Tunnel module is depicted in [Figure 62](#). The Tunnel module inherits concepts from the Construction module (cf. [Section 8.15](#)). The Construction module defines objects that are common to all types of construction, such as the different surface types and the openings. A detailed discussion of the Requirements Class Tunnel can be found in the [CityGML 3.0 Users Guide](#).

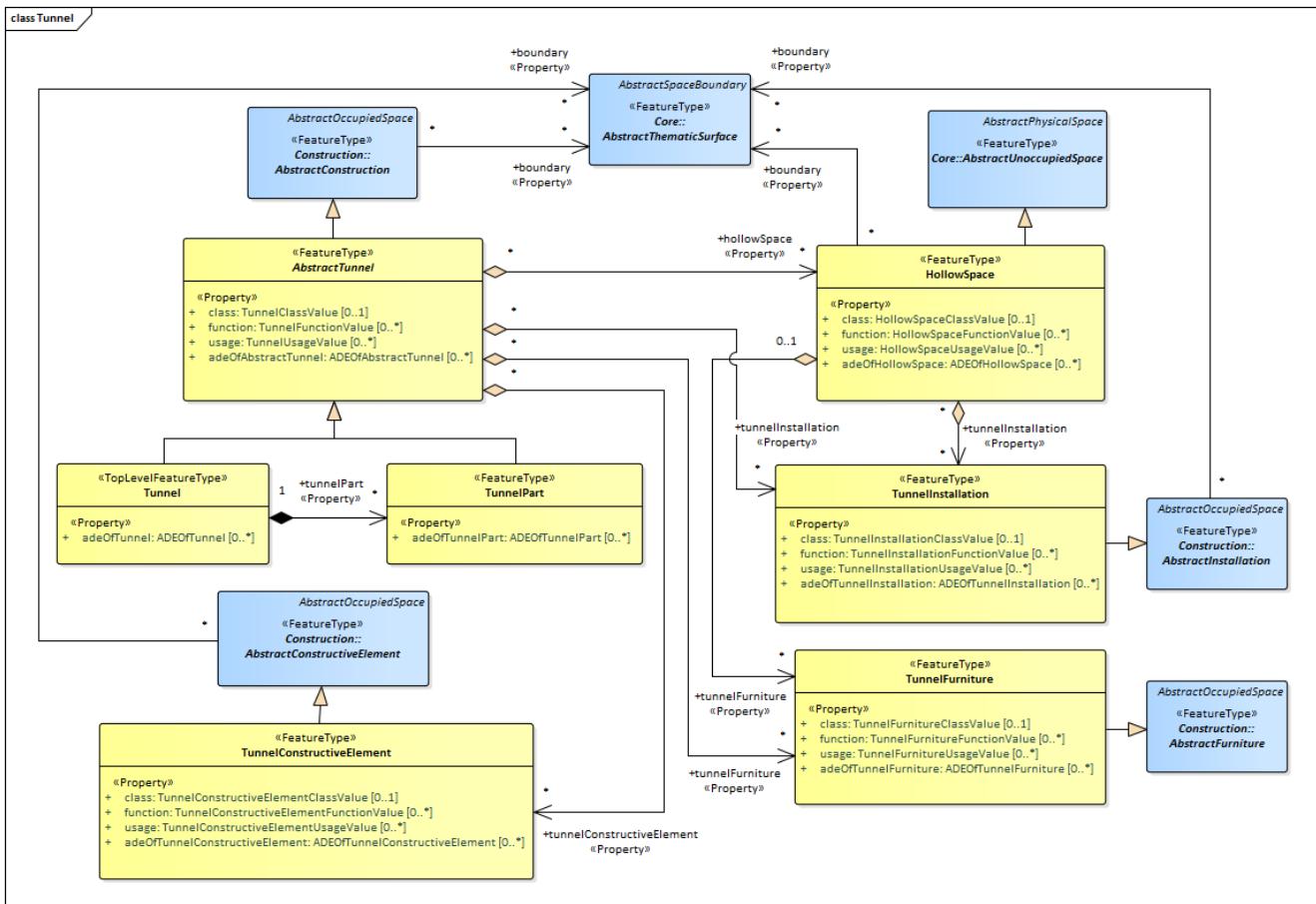


Figure 62. UML diagram of the Tunnel Model.

The ADE data types provided for the Tunnel module are illustrated in the figure Figure 63.

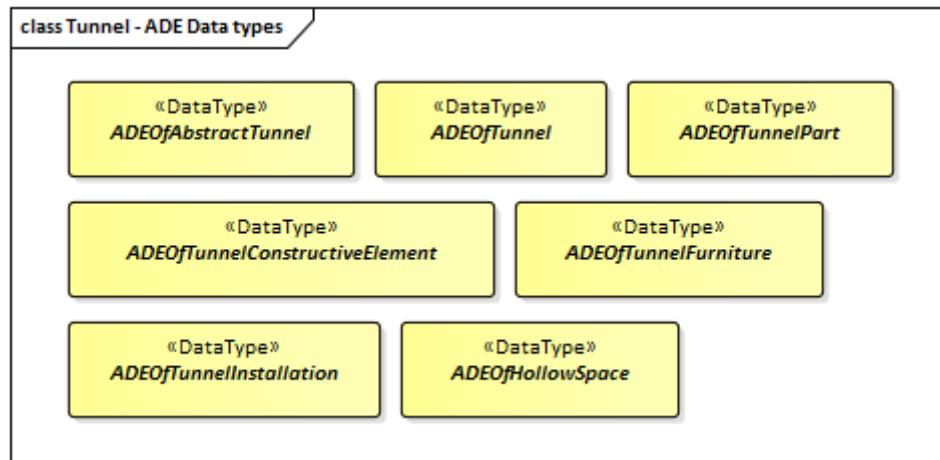


Figure 63. ADE classes of the CityGML Tunnel module.

The Code Lists provided for the Tunnel module are illustrated in the figure Figure 64.

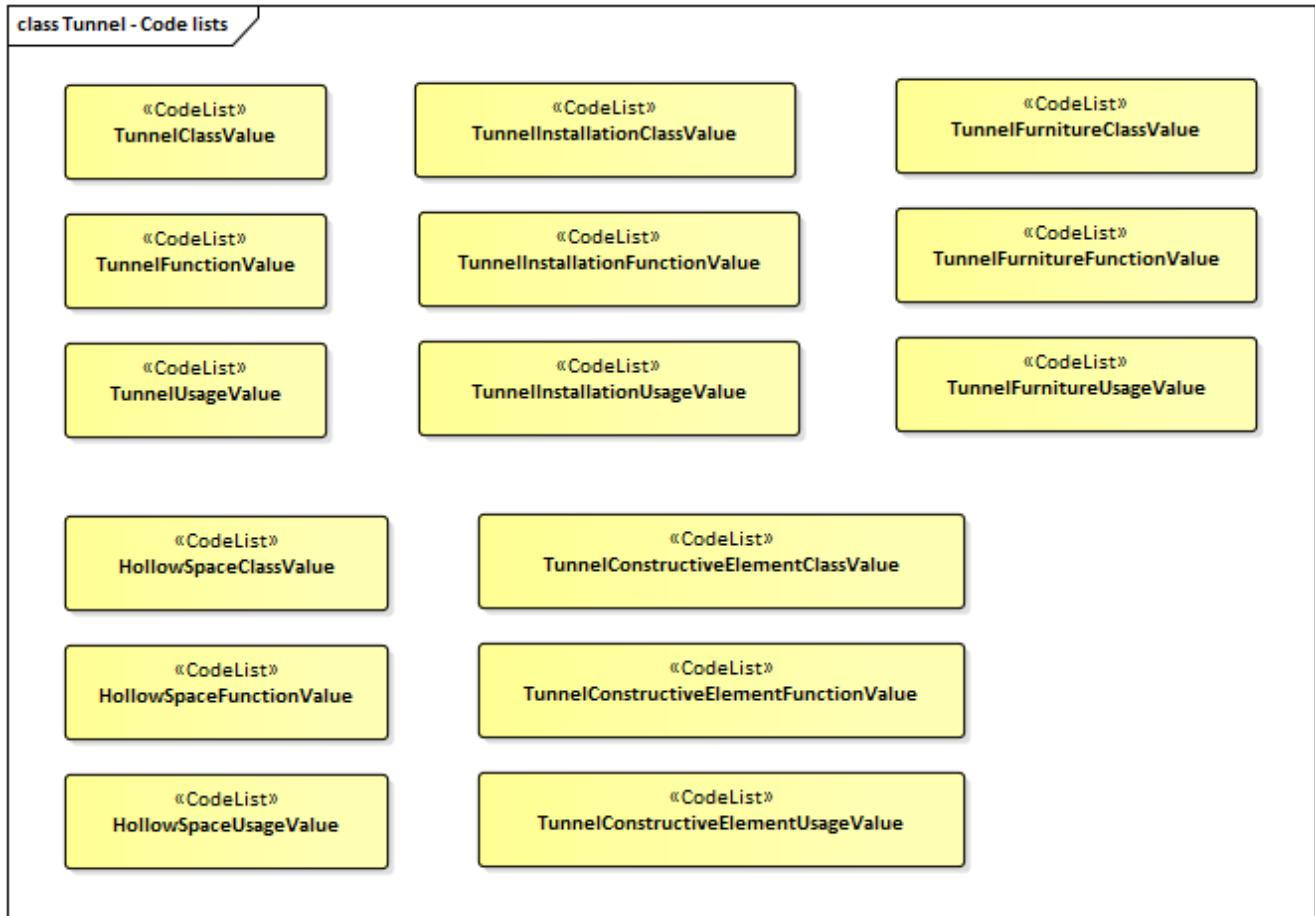


Figure 64. Codelists from the CityGML Tunnel module.

Table 70 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Tunnel module:

Table 70. Tunnel space classes and their allowed thematic surface boundaries

Space class	Allowed space boundaries
AbstractTunnel	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs

HollowSpace	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
Tunnel	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
TunnelConstructiveElement	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
TunnelFurniture	No boundaries allowed

TunnelInstallation	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
TunnelPart	<ul style="list-style-type: none"> • Construction::AbstractConstructionSurface and all subclasses, i.e. Construction::GroundSurface, Construction::RoofSurface, Construction::CeilingSurface, Construction::OuterCeilingSurface, Construction::FloorSurface, Construction::OuterFloorSurface, Construction::WallSurface, Construction::InteriorWallSurface • Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs

8.18.1. Requirements

The following requirement defines the rules governing implementation of the CityGML Tunnel Module as an Implementation Specification.

Requirement 45 /req/tunnel/classes	
For each UML class defined or referenced in the Tunnel Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.

D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and multiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how it observes all constraints the Conceptual Model imposes on the UML class.

The implementation of this Module is further constrained by the following spatial boundary requirement:

Requirement 46	/req/tunnel/boundaries
Table 70 lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Tunnel module. An Implementation Specification SHALL NOT specify boundaries except as specified in Table 70	

The use of extension capabilities by Tunnel elements is constrained by the following requirement:

Requirement 47	/req/Tunnel/ade/use
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

8.18.2. Class Definitions

Table 71. Classes used in Tunnel

Class	Description
Tunnel «TopLevelFeatureType»	A Tunnel represents a horizontal or sloping enclosed passage way of a certain length, mainly underground or underwater. [cf. ISO 6707-1]
AbstractTunnel «FeatureType»	AbstractTunnel is an abstract superclass representing the common attributes and associations of the classes Tunnel and TunnelPart.
HollowSpace «FeatureType»	A HollowSpace is a space within a Tunnel or TunnelPart intended for certain functions (e.g. transport or passage ways, service rooms, emergency shelters). A HollowSpace is bounded physically and/or virtually (e.g. by ClosureSurfaces or GenericSurfaces).
TunnelConstructiveElement «FeatureType»	A TunnelConstructiveElement is an element of a Tunnel which is essential from a structural point of view. Examples are walls, slabs, beams.
TunnelFurniture «FeatureType»	A TunnelFurniture is an equipment for occupant use, usually not fixed to the tunnel. [cf. ISO 6707-1]

TunnelInstallation «FeatureType»	A TunnelInstallation is a permanent part of a Tunnel (inside and/or outside) which does not have the significance of a TunnelPart. In contrast to TunnelConstructiveElement, a TunnelInstallation is not essential from a structural point of view. Examples are stairs, antennas or railings.
TunnelPart «FeatureType»	A TunnelPart is a physical or functional subdivision of a Tunnel. It would be considered a Tunnel, if it were not part of a collection of other TunnelParts.

Table 72. Data Types used in Tunnel

Name	Description
ADEOfAbstractTunnel «DataType»	ADEOfAbstractTunnel acts as a hook to define properties within an ADE that are to be added to AbstractTunnel.
ADEOfHollowSpace «DataType»	ADEOfHollowSpace acts as a hook to define properties within an ADE that are to be added to a HollowSpace.
ADEOfTunnel «DataType»	ADEOfTunnel acts as a hook to define properties within an ADE that are to be added to a Tunnel.
ADEOfTunnelConstructiveElement «DataType»	ADEOfTunnelConstructiveElement acts as a hook to define properties within an ADE that are to be added to a TunnelConstructiveElement.
ADEOfTunnelFurniture «DataType»	ADEOfTunnelFurniture acts as a hook to define properties within an ADE that are to be added to a TunnelFurniture.
ADEOfTunnelInstallation «DataType»	ADEOfTunnelInstallation acts as a hook to define properties within an ADE that are to be added to a TunnelInstallation.
ADEOfTunnelPart «DataType»	ADEOfTunnelPart acts as a hook to define properties within an ADE that are to be added to a TunnelPart.

Table 73. CodeList Classes used in Tunnel

Name	Description
HollowSpaceClassValue «CodeList»	HollowSpaceClassValue is a code list used to further classify a HollowSpace.
HollowSpaceFunctionValue «CodeList»	HollowSpaceFunctionValue is a code list that enumerates the different purposes of a HollowSpace.
HollowSpaceUsageValue «CodeList»	HollowSpaceUsageValue is a code list that enumerates the different uses of a HollowSpace.
TunnelClassValue «CodeList»	TunnelClassValue is a code list used to further classify a Tunnel.
TunnelConstructiveElementClassValue «CodeList»	TunnelConstructiveElementClassValue is a code list used to further classify a TunnelConstructiveElement.

TunnelConstructiveElementFunctionValue «CodeList»	TunnelConstructiveElementFunctionValue is a code list that enumerates the different purposes of a TunnelConstructiveElement.
TunnelConstructiveElementUsageValue «CodeList»	TunnelConstructiveElementUsageValue is a code list that enumerates the different uses of a TunnelConstructiveElement.
TunnelFunctionValue «CodeList»	TunnelFunctionValue is a code list that enumerates the different purposes of a Tunnel.
TunnelFurnitureClassValue «CodeList»	TunnelFurnitureClassValue is a code list used to further classify a TunnelFurniture.
TunnelFurnitureFunctionValue «CodeList»	TunnelFurnitureFunctionValue is a code list that enumerates the different purposes of a TunnelFurniture.
TunnelFurnitureUsageValue «CodeList»	TunnelFurnitureUsageValue is a code list that enumerates the different uses of a TunnelFurniture.
TunnelInstallationClassValue «CodeList»	TunnelInstallationClassValue is a code list used to further classify a TunnelInstallation.
TunnelInstallationFunctionValue «CodeList»	TunnelInstallationFunctionValue is a code list that enumerates the different purposes of a TunnelInstallation.
TunnelInstallationUsageValue «CodeList»	TunnelInstallationUsageValue is a code list that enumerates the different uses of a TunnelInstallation.
TunnelUsageValue «CodeList»	TunnelUsageValue is a code list that enumerates the different uses of a Tunnel.

8.18.3. Additional Information

Additional information about the Tunnel Module can be found in the [OGC CityGML 3.0 Users Guide](#)

Chapter 9. CityGML Data Dictionary

The CityGML UML model is the normative definition of the CityGML Conceptual Model. The Data Dictionary tables in this section were software generated from the UML model. As such, this section provides a normative representation of the CityGML Conceptual Model.

An alternate representation can be found in the Conceptual Model in [Chapter 8](#).

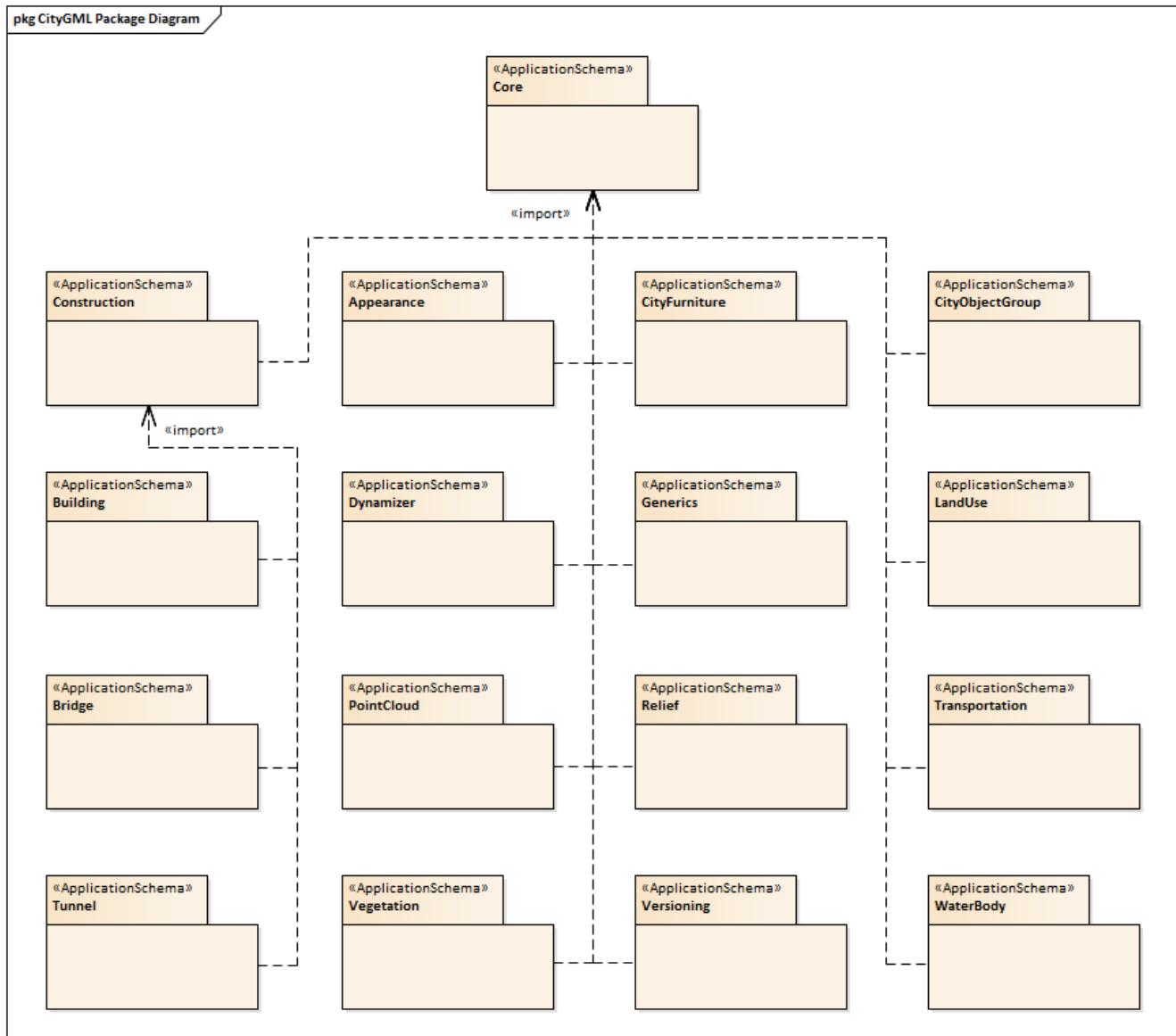


Figure 65. CityGML UML Packages

9.1. ISO Classes

The following classes are defined in ISO standards and used by the CityGML Conceptual Model.

9.1.1. Class AnyFeature ([ISO 19109:2015](#))

AnyFeature

Definition:	AnyFeature is an abstract class that is the generalization of all feature types. AnyFeature is an instance of the «metaclass» FeatureType [cf. ISO 19109].
Subclass Of:	none
Stereotype:	«FeatureType»
<hr/>	
Role name	Target class and multiplicity
	FeatureType [1..1]
<hr/>	
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»	

9.1.2. Class CV_DiscreteGridPointCoverage (ISO 19123:2005)

CV_DiscreteGridPointCoverage		
<hr/>		
Definition:	A coverage that returns the same feature attribute values for every direct position within any single spatial object, temporal object or spatiotemporal object in its domain.	
Subclass Of:	CV_DiscreteCoverage	
Stereotype:	«type»	
<hr/>		
Role name	Target class and multiplicity	Definition
element	CV_GridPointValue Pair [1..*]	
valueAssignm ent	CV_GridValuesMatr ix [1..1]	
<hr/>		
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»		

9.1.3. Class DirectPosition (ISO 19107: 2003)

DirectPosition
<hr/>

Definition:	DirectPosition object data types (Figure 14) hold the coordinates for a position within some coordinate reference system. The coordinate reference system is described in ISO 19111. Since DirectPositions, as data types, will often be included in larger objects (such as GM_Objects) that have references to ISO19111::SC_CRS, the DirectPosition::coordinateReferenceSystem may be left NULL if this particular DirectPosition is included in a larger object with such a reference to a SC_CRS. In this case, the DirectPosition::coordinateReferenceSystem is implicitly assumed to take on the value of the containing object's SC_CRS.	
Subclass Of:	None	
Stereotype:	None	
Role name Target class and multiplicity Definition		
CRS	CRS [0..1]	
CRS	SC_CRS [0..1]	
Attribute Value type and multiplicity Definition		
coordinate	Sequence< Number > [1..1]	
dimension	Integer [1..1]	
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»		

9.1.4. Class GM_Object ([ISO 19107: 2003](#))

GM_Object

Definition:	GM_Object is the root class of the geometric object taxonomy and supports interfaces common to all geographically referenced geometric objects. GM_Object instances are sets of direct positions in a particular coordinate reference system. A GM_Object can be regarded as an infinite set of points that satisfies the set operation interfaces for a set of direct positions, TransfiniteSet<DirectPosition>. Since an infinite collection class cannot be implemented directly, a Boolean test for inclusion shall be provided by the GM_Object interface. This international standard concentrates on vector geometry classes, but future work may use GM_Object as a root class without modification. NOTE As a type, GM_Object does not have a well-defined default state or value representation as a data type. Instantiated subclasses of GM_Object will.
Subclass Of:	none
Stereotype:	«type»
Constraint:	dimension() > boundary().dimension (Invariant):
Constraint:	boundary().notEmpty() implies boundary().dimension() = dimension() -1 (Invariant):
Constraint:	boundary().isEmpty() = isCycle() (Invariant):

Role name	Target class and multiplicity	Definition
	Geometry [1..1]	
	TransfiniteSet<DirectPosition> [1..1]	
	CV_DomainObject [1..1]	
CRS	CRS [0..1]	
CRS	SC_CRS [0..1]	

Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»

9.1.5. Class GM_MultiCurve (ISO 19107: 2003)

[GM_MultiCurve](#)

Definition:	An aggregate class containing only instances of GM_OrientableCurve. The association role “element” shall be the set of GM_OrientableCurves contained in this GM_MultiCurve.	
Subclass Of:	GM_MultiPrimitive	
Stereotype:	«type»	
<hr/>		
Attribute	Value type and multiplicity	Definition
length	Length [1..1]	
<hr/>		
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»		

9.1.6. Class GM_MultiPoint ([ISO 19107:2003](#))

GM_MultiPoint		
<hr/>		
Definition:		GM_MultiPoint is an aggregate class containing only points. The association role “element” shall be the set of GM_Points contained in this GM_MultiPoint.
Subclass Of:		GM_MultiPrimitive
Stereotype:		«type»
<hr/>		
Attribute	Value type and multiplicity	Definition
position	Set< DirectPosition > [1..1]	
<hr/>		
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»		

9.1.7. Class GM_MultiSurface ([ISO 19107:2003](#))

GM_MultiSurface		
<hr/>		
Definition:		An aggregate class containing only instances of GM_OrientableSurface. The association role “element” shall be the set of GM_OrientableSurfaces contained in this GM_MultiSurface.
Subclass Of:		GM_MultiPrimitive
Stereotype:		«type»

Attribute	Value type and multiplicity	Definition
area	Area [1..1]	
perimeter	Length [1..1]	

Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»

9.1.8. Class GM_Point (ISO 19107:2003)

GM_Point		
Definition:	GM_Point is the basic data type for a geometric object consisting of one and only one point.	
Subclass Of:	GM_Primitive	
Stereotype:	«type»	
Role name	Target class and multiplicity	Definition
	Point [1..1]	
composite	GM_CompositePoint [0..*]	
Attribute	Value type and multiplicity	Definition
position	DirectPosition [1..1]	The attribute "position" shall be the DirectPosition of this GM_Point. GM_Point::position [1] : DirectPosition NOTE In most cases, the state of a GM_Point is fully determined by its position attribute. The only exception to this is if the GM_Point has been subclassed to provide additional non-geometric information such as symbology.

Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»

9.1.9. Class GM_Solid (ISO 19107:2003)

GM_Solid

Definition:	GM_Solid, a subclass of GM_Primitive, is the basis for 3-dimensional geometry. The extent of a solid is defined by the boundary surfaces.							
Subclass Of:	GM_Primitive							
Stereotype:	«type»							
<table border="1"> <thead> <tr> <th>Role name</th> <th>Target class and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>composite</td> <td>GM_CompositeSoli d [0..*]</td> <td>Solid [1..1]</td> </tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	composite	GM_CompositeSoli d [0..*]	Solid [1..1]
Role name	Target class and multiplicity	Definition						
composite	GM_CompositeSoli d [0..*]	Solid [1..1]						
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»								

9.1.10. Class GM_Surface (ISO 19107:2003)

GM_Surface		
<p>Definition: GM_Surface is a subclass of GM_Primitive and is the basis for 2-dimensional geometry. Unorientable surfaces such as the Möbius band are not allowed. The orientation of a surface chooses an "up" direction through the choice of the upward normal, which, if the surface is not a cycle, is the side of the surface from which the exterior boundary appears counterclockwise. Reversal of the surface orientation reverses the curve orientation of each boundary component, and interchanges the conceptual "up" and "down" direction of the surface. If the surface is the boundary of a solid, the "up" direction is usually outward. For closed surfaces, which have no boundary, the up direction is that of the surface patches, which must be consistent with one another. Its included GM_SurfacePatches describe the interior structure of a GM_Surface. NOTE Other than the restriction on orientability, no other "validity" condition is required for GM_Surface.</p>		
Subclass Of:	GM_OrientableSurface	
Stereotype:	«type»	
Role name	Target class and multiplicity	Definition
	GM_GenericSurfac e [1..1]	Building [0..*]
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»		

9.1.11. Class GM_Tin (ISO 19107:2003)

GM_Tin

Definition: A GM_Tin is a GM_TriangulatedSurface that uses the Delaunay algorithm or a similar algorithm complemented with consideration for breaklines, stoplines and maximum length of triangle sides (Figure 22). These networks satisfy the Delaunay criterion away from the modifications: For each triangle in the network, the circle passing through its vertexes does not contain, in its interior, the vertex of any other triangle.

Subclass Of: [GM_TriangulatedSurface](#)

Stereotype: «type»

Attribute	Value type and multiplicity	Definition
------------------	------------------------------------	-------------------

breakLines Set<[GM_LineString](#)> [1..1]

controlPoint [GM_Position](#) [3..*]

maxLength [Distance](#) [1..1]

stopLines Set<[GM_LineString](#)> [1..1]

Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»

9.1.12. Class GM_TriangulatedSurface (ISO 19107:2003)

GM_TriangulatedSurface

Definition: A GM_TriangulatedSurface is a GM_PolyhedralSurface that is composed only of triangles (GM_Triangle). There is no restriction on how the triangulation is derived.

Subclass Of: [GM_PolyhedralSurface](#)

Stereotype: «type»

Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»

9.1.13. Class SC_CRS (ISO 19111:2019)

SC_CRS

Definition:	Coordinate reference system which is usually single but may be compound.	
Subclass Of:	IO_IdentifiedObjectBase , RS_ReferenceSystem	
Stereotype:	«type»	
Role name	Target class and multiplicity	Definition
coordOperationTo	CC_CoordinateOper [0..*]	Not-navigable association from a Coordinate Operation that uses ths CRS as its targetCRS.
grid	CV_ReferenceableGrid [0..*]	
Attribute	Value type and multiplicity	Definition
scope	CharacterString [1..*]	Description of usage, or limitations of usage, for which this CRS is valid. If unknown, enter "not known".
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»		

9.1.14. Class TM_Position (ISO 19108:2006)

TM_Position		
Definition: TM_Position is a union class that consists of one of the data types listed as its attributes. Date, Time, and DateTime are basic data types defined in ISO/TS 19103.		
Subclass Of:	None	
Stereotype:	«Union»	
Attribute	Value type and multiplicity	Definition
anyOther	TM_TemporalPosition [1..1]	
date8601	Date [1..1]	
time8601	Time [1..1]	
dateTime8601	DateTime [1..1]	
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»		

9.2. Core

Description: The Core module defines the basic components of the CityGML data model. The Core module defines abstract base classes that define the core properties of more specialized thematic classes defined in other modules. The Core module also defines concrete classes that are common to other modules, for example basic data types.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.2.1. Classes

AbstractAppearance

Definition: AbstractAppearance is the abstract superclass to represent any kind of appearance objects.

Subclass of: [AbstractFeatureWithLifespan](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

adeOfAbstract Appearance	ADEOfAbstractApp earance [0..*]	Augments AbstractAppearance with properties defined in an ADE.
--------------------------	---	--

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractCityObject

Definition: AbstractCityObject is the abstract superclass of all thematic classes within the CityGML conceptual model.

Subclass of: [AbstractFeatureWithLifespan](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
generalizesTo	AbstractCityObject [*]	Relates generalized representations of the same real-world object in different Levels of Detail to the city object. The direction of this relation is from the city object to the corresponding generalized city objects.
genericAttribute	AbstractGenericAttribute [*]	Relates generic attributes to the city object.
dynamizer	AbstractDynamizer [*]	Relates Dynamizer objects to the city object. These allow timeseries data to override static attribute values of the city object.
appearance	AbstractAppearance [*]	Relates appearances to the city object.
externalReference	ExternalReference [*]	References external objects in other information systems that have a relation to the city object.
relatedTo	AbstractCityObject [*]	Relates other city objects to the city object. It also describes how the city objects are related to each other.

Attribute	Value type and multiplicity	Definition
relativeToTerrain	RelativeToTerrain [0..1]	Describes the vertical position of the city object relative to the surrounding terrain.
relativeToWater	RelativeToWater [0..1]	Describes the vertical position of the city object relative to the surrounding water surface.
adeOfAbstractCityObject	ADEOfAbstractCityObject [0..*]	Augments AbstractCityObject with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractDynamizer

Definition:	AbstractDynamizer is the abstract superclass to represent Dynamizer objects.
Subclass of:	AbstractFeatureWithLifespan
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstractDynamizer	ADEOfAbstractDynamizer [0..*]	Augments AbstractDynamizer with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractFeature

Definition: AbstractFeature is the abstract superclass of all feature types within the CityGML conceptual model.

Subclass of: [AnyFeature](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
featureID	ID [1..1]	Specifies the unique identifier of the feature that is valid in the instance document within which it occurs.
identifier	ScopedName [0..1]	Specifies the unique identifier of the feature that is valid globally.
name	GenericName [0..*]	Specifies the name of the feature.
description	CharacterString [0..1]	Provides further information on the feature.
adeOfAbstract Feature	ADEOfAbstractFeature [0..*]	Augments AbstractFeature with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractFeatureWithLifespan

Definition: AbstractFeatureWithLifespan is the base class for all CityGML features. It allows the optional specification of the real-world and database times for the existence of each feature.

Subclass of: [AbstractFeature](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
creationDate	DateTime [0..1]	Indicates the date at which a CityGML feature was added to the CityModel.
terminationDate	DateTime [0..1]	Indicates the date at which a CityGML feature was removed from the CityModel.
validFrom	DateTime [0..1]	Indicates the date at which a CityGML feature started to exist in the real world.
validTo	DateTime [0..1]	Indicates the date at which a CityGML feature ended to exist in the real world.
adeOfAbstractFeatureWithLifespan	ADEOfAbstractFeatureWithLifespan [0..*]	Augments AbstractFeatureWithLifespan with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractLogicalSpace

Definition:	AbstractLogicalSpace is the abstract superclass for all types of logical spaces. Logical space refers to spaces that are not bounded by physical surfaces but are defined according to thematic considerations.
Subclass of:	AbstractSpace
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstractLogicalSpace	ADEOfAbstractLogicalSpace [0..*]	Augments AbstractLogicalSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractOccupiedSpace

Definition:	AbstractOccupiedSpace is the abstract superclass for all types of physically occupied spaces. Occupied space refers to spaces that are partially or entirely filled with matter.
Subclass of:	AbstractPhysicalSpace
Stereotype:	«FeatureType»

Role name	Target class and multiplicity	Definition
lod3ImplicitRepresentation	ImplicitGeometry [0..1]	Relates to an implicit geometry that represents the occupied space in Level of Detail 3.
lod1ImplicitRepresentation	ImplicitGeometry [0..1]	Relates to an implicit geometry that represents the occupied space in Level of Detail 1.
lod2ImplicitRepresentation	ImplicitGeometry [0..1]	Relates to an implicit geometry that represents the occupied space in Level of Detail 2.
Attribute	Value type and multiplicity	Definition
adeOfAbstractOccupiedSpace	ADEOfAbstractOccupiedSpace [0..*]	Augments AbstractOccupiedSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractPhysicalSpace

Definition:	AbstractPhysicalSpace is the abstract superclass for all types of physical spaces. Physical space refers to spaces that are fully or partially bounded by physical objects.	
Subclass of:	AbstractSpace	
Stereotype:	«FeatureType»	
Role name	Target class and multiplicity	Definition
lod3TerrainIntersectionCurve	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the terrain intersection curve of the physical space in Level of Detail 3.
lod2TerrainIntersectionCurve	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the terrain intersection curve of the physical space in Level of Detail 2.
pointCloud	AbstractPointCloud [0..1]	Relates to a 3D PointCloud that represents the physical space.
lod1TerrainIntersectionCurve	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the terrain intersection curve of the physical space in Level of Detail 1.

Attribute	Value type and multiplicity	Definition
adeOfAbstract PhysicalSpace	ADEOfAbstractPhysicalSpace [0..*]	Augments AbstractPhysicalSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractPointCloud

Definition: AbstractPointCloud is the abstract superclass to represent PointCloud objects.
Subclass of: [AbstractFeature](#)
Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstract PointCloud	ADEOfAbstractPointCloud [0..*]	Augments AbstractPointCloud with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractSpace

Definition: AbstractSpace is the abstract superclass for all types of spaces. A space is an entity of volumetric extent in the real world.
Subclass of: [AbstractCityObject](#)
Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
lod2MultiCurve	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the space in Level of Detail 2.
lod0MultiCurve	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the space in Level of Detail 0.
lod0MultiSurface	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the space in Level of Detail 0.
lod2MultiSurface	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the space in Level of Detail 2.
lod3MultiSurface	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the space in Level of Detail 3.
lod0Point	GM_Point [0..1]	Relates to a 3D Point geometry that represents the space in Level of Detail 0.
lod3Solid	GM_Solid [0..1]	Relates to a 3D Solid geometry that represents the space in Level of Detail 3.
lod3MultiCurve	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the space in Level of Detail 3.
lod2Solid	GM_Solid [0..1]	Relates to a 3D Solid geometry that represents the space in Level of Detail 2.
boundary	AbstractSpaceBoundary [*]	Relates to surfaces that bound the space.
lod1Solid	GM_Solid [0..1]	Relates to a 3D Solid geometry that represents the space in Level of Detail 1.
Attribute	Value type and multiplicity	Definition
spaceType	SpaceType [0..1]	Specifies the degree of openness of a space.
volume	QualifiedVolume [0..*]	Specifies qualified volumes related to the space.
area	QualifiedArea [0..*]	Specifies qualified areas related to the space.
adeOfAbstractSpace	ADEOfAbstractSpace [0..*]	Augments AbstractSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractSpaceBoundary

Definition:	AbstractSpaceBoundary is the abstract superclass for all types of space boundaries. A space boundary is an entity with areal extent in the real world. Space boundaries are objects that bound a Space. They also realize the contact between adjacent spaces.										
Subclass of:	AbstractCityObject										
Stereotype:	«FeatureType»										
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>adeOfAbstract SpaceBoundary</td><td>ADEOfAbstractSpaceBoundary [0..*]</td><td>Augments AbstractSpaceBoundary with properties defined in an ADE.</td></tr> <tr> <td>y</td><td></td><td></td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	adeOfAbstract SpaceBoundary	ADEOfAbstractSpaceBoundary [0..*]	Augments AbstractSpaceBoundary with properties defined in an ADE.	y		
Attribute	Value type and multiplicity	Definition									
adeOfAbstract SpaceBoundary	ADEOfAbstractSpaceBoundary [0..*]	Augments AbstractSpaceBoundary with properties defined in an ADE.									
y											
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».											

AbstractThematicSurface

Definition:	AbstractThematicSurface is the abstract superclass for all types of thematic surfaces.																						
Subclass of:	AbstractSpaceBoundary																						
Stereotype:	«FeatureType»																						
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>lod1MultiSurf ace</td><td>GM_MultiSurface [0..1]</td><td>Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 1.</td></tr> <tr> <td>pointCloud</td><td>AbstractPointCloud [0..1]</td><td>Relates to a 3D PointCloud that represents the thematic surface.</td></tr> <tr> <td>lod0MultiCurv e</td><td>GM_MultiCurve [0..1]</td><td>Relates to a 3D MultiCurve geometry that represents the thematic surface in Level of Detail 0.</td></tr> <tr> <td>lod3MultiSurf ace</td><td>GM_MultiSurface [0..1]</td><td>Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 3.</td></tr> <tr> <td>lod0MultiSurf ace</td><td>GM_MultiSurface [0..1]</td><td>Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 0.</td></tr> <tr> <td>lod2MultiSurf ace</td><td>GM_MultiSurface [0..1]</td><td>Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 2.</td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	lod1MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 1.	pointCloud	AbstractPointCloud [0..1]	Relates to a 3D PointCloud that represents the thematic surface.	lod0MultiCurv e	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the thematic surface in Level of Detail 0.	lod3MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 3.	lod0MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 0.	lod2MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 2.
Role name	Target class and multiplicity	Definition																					
lod1MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 1.																					
pointCloud	AbstractPointCloud [0..1]	Relates to a 3D PointCloud that represents the thematic surface.																					
lod0MultiCurv e	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the thematic surface in Level of Detail 0.																					
lod3MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 3.																					
lod0MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 0.																					
lod2MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 2.																					

Attribute	Value type and multiplicity	Definition
area	QualifiedArea [0..*]	Specifies qualified areas related to the thematic surface.
adeOfAbstractThematicSurface	ADEOfAbstractThematicSurface [0..*]	Augments AbstractThematicSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractUnoccupiedSpace

Definition:	AbstractUnoccupiedSpace is the abstract superclass for all types of physically unoccupied spaces. Unoccupied space refers to spaces that are entirely or mostly free of matter.	
Subclass of:	AbstractPhysicalSpace	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
adeOfAbstractUnoccupiedSpace	ADEOfAbstractUnoccupiedSpace [0..*]	Augments AbstractUnoccupiedSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractVersion

Definition:	AbstractVersion is the abstract superclass to represent Version objects.	
Subclass of:	AbstractFeatureWithLifespan	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
adeOfAbstractVersion	ADEOfAbstractVersion [0..*]	Augments AbstractVersion with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractVersionTransition

Definition: AbstractVersionTransition is the abstract superclass to represent VersionTransition objects.

Subclass of: [AbstractFeatureWithLifespan](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

adeOfAbstractVersionTransition [ADEOfAbstractVersionTransition](#) [0..*] Augments AbstractVersionTransition with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Address

Definition: Address represents an address of a city object.

Subclass of: [AbstractFeature](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
-----------	-------------------------------	------------

multiPoint [GM_MultiPoint](#) [0..1] Relates to the MultiPoint geometry of the Address. The geometry relates the address spatially to a city object.

xalAddress [XALAddress](#) [1..1] Relates an OASIS address object to the Address.

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

adeOfAddress [ADEOfAddress](#) [0..*] Augments the Address with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

CityModel

Definition:	CityModel is the container for all objects belonging to a city model.										
Subclass of:	AbstractFeatureWithLifespan										
Stereotype:	«FeatureType»										
<table border="1"> <thead> <tr> <th>Role name</th> <th>Target class and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>cityModelMember</td> <td>CityModelMember [*]</td> <td>Relates to all objects that are part of the CityModel.</td> </tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	cityModelMember	CityModelMember [*]	Relates to all objects that are part of the CityModel.			
Role name	Target class and multiplicity	Definition									
cityModelMember	CityModelMember [*]	Relates to all objects that are part of the CityModel.									
<table border="1"> <thead> <tr> <th>Attribute</th> <th>Value type and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>engineeringCRS</td> <td>EngineeringCRS [0..1]</td> <td>Specifies the local engineering coordinate reference system of the CityModel that can be provided inline the CityModel instead of referencing a well-known CRS definition. The definition of an engineering CRS requires an anchor point which relates the origin of the local coordinate system to a point on the earth's surface in order to facilitate the transformation of coordinates from the local engineering CRS.</td> </tr> <tr> <td>adeOfCityModel</td> <td>ADEOfCityModel [0..*]</td> <td>Augments the CityModel with properties defined in an ADE.</td> </tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	engineeringCRS	EngineeringCRS [0..1]	Specifies the local engineering coordinate reference system of the CityModel that can be provided inline the CityModel instead of referencing a well-known CRS definition. The definition of an engineering CRS requires an anchor point which relates the origin of the local coordinate system to a point on the earth's surface in order to facilitate the transformation of coordinates from the local engineering CRS.	adeOfCityModel	ADEOfCityModel [0..*]	Augments the CityModel with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition									
engineeringCRS	EngineeringCRS [0..1]	Specifies the local engineering coordinate reference system of the CityModel that can be provided inline the CityModel instead of referencing a well-known CRS definition. The definition of an engineering CRS requires an anchor point which relates the origin of the local coordinate system to a point on the earth's surface in order to facilitate the transformation of coordinates from the local engineering CRS.									
adeOfCityModel	ADEOfCityModel [0..*]	Augments the CityModel with properties defined in an ADE.									
<p>Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».</p>											

CityObjectRelation								
Definition:	CityObjectRelation represents a specific relation from the city object in which it is included to another city object.							
Subclass of:	None							
Stereotype:	«ObjectType»							
<table border="1"> <thead> <tr> <th>Role name</th> <th>Target class and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>genericAttribute</td> <td>AbstractGenericAttribute [*]</td> <td>Relates generic attributes to the CityObjectRelation.</td> </tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	genericAttribute	AbstractGenericAttribute [*]	Relates generic attributes to the CityObjectRelation.
Role name	Target class and multiplicity	Definition						
genericAttribute	AbstractGenericAttribute [*]	Relates generic attributes to the CityObjectRelation.						

Attribute	Value type and multiplicity	Definition
relationType	RelationTypeValue [1..1]	Indicates the specific type of the CityObjectRelation.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

ClosureSurface

Definition:	ClosureSurface is a special type of thematic surface used to close holes in volumetric objects. Closure surfaces are virtual (non-physical) surfaces.	
Subclass of:	AbstractThematicSurface	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
adeOfClosure Surface	ADEOfClosureSurface [0..*]	Augments the ClosureSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

ImplicitGeometry

Definition:	ImplicitGeometry is a geometry representation where the shape is stored only once as a prototypical geometry, for example a tree or other vegetation object, a traffic light or a traffic sign. This prototypic geometry object can be re-used or referenced many times, wherever the corresponding feature occurs in the 3D city model.
Subclass of:	None
Stereotype:	«ObjectType»

Role name	Target class and multiplicity	Definition
relativeGeom	GM_Object [0..1]	Relates to a prototypical geometry in a local coordinate system stored inline with the city model.
referencePoint	GM_Point [1..1]	Relates to a 3D Point geometry that represents the base point of the object in the world coordinate system.
appearance	AbstractAppearance [*]	Relates appearances to the ImplicitGeometry.
Attribute	Value type and multiplicity	Definition
objectID	ID [1..1]	Specifies the unique identifier of the ImplicitGeometry.
transformationMatrix	TransformationMatrix4x4 [1..1]	Specifies the mathematical transformation (translation, rotation, and scaling) between the prototypical geometry and the actual spatial position of the object.
mimeType	MimeTypeValue [0..1]	Specifies the MIME type of the external file that stores the prototypical geometry.
libraryObject	URI [0..1]	Specifies the URI that points to the prototypical geometry stored in an external file.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.2.2. Data Types

AbstractGenericAttribute

Definition:	AbstractGenericAttribute is the abstract superclass for all types of generic attributes.
Subclass of:	None
Stereotype:	«DataType»

ADEOfAbstractAppearance

Definition:	ADEOfAbstractAppearance acts as a hook to define properties within an ADE that are to be added to AbstractAppearance.
Subclass of:	None
Stereotype:	«DataType»

ADEOfAbstractCityObject

Definition: ADEOfAbstractCityObject acts as a hook to define properties within an ADE that are to be added to AbstractCityObject.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractDynamizer

Definition: ADEOfAbstractDynamizer acts as a hook to define properties within an ADE that are to be added to AbstractDynamizer.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractFeature

Definition: ADEOfAbstractFeature acts as a hook to define properties within an ADE that are to be added to AbstractFeature.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractFeatureWithLifespan

Definition: ADEOfAbstractFeatureWithLifespan acts as a hook to define properties within an ADE that are to be added to AbstractFeatureWithLifespan.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractLogicalSpace

Definition: ADEOfAbstractLogicalSpace acts as a hook to define properties within an ADE that are to be added to AbstractLogicalSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractOccupiedSpace

Definition: ADEOfAbstractOccupiedSpace acts as a hook to define properties within an ADE that are to be added to AbstractOccupiedSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractPhysicalSpace

Definition: ADEOfAbstractPhysicalSpace acts as a hook to define properties within an ADE that are to be added to AbstractPhysicalSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractPointCloud

Definition: ADEOfAbstractPointCloud acts as a hook to define properties within an ADE that are to be added to AbstractPointCloud.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractSpace

Definition: ADEOfAbstractSpace acts as a hook to define properties within an ADE that are to be added to AbstractSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractSpaceBoundary

Definition: ADEOfAbstractSpaceBoundary acts as a hook to define properties within an ADE that are to be added to AbstractSpaceBoundary.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractThematicSurface

Definition: ADEOfAbstractThematicSurface acts as a hook to define properties within an ADE that are to be added to AbstractThematicSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractUnoccupiedSpace

Definition: ADEOfAbstractUnoccupiedSpace acts as a hook to define properties within an ADE that are to be added to AbstractUnoccupiedSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractVersion

Definition: ADEOfAbstractVersion acts as a hook to define properties within an ADE that are to be added to AbstractVersion.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractVersionTransition

Definition: ADEOfAbstractVersionTransition acts as a hook to define properties within an ADE that are to be added to AbstractVersionTransition.

Subclass of: None

Stereotype: «DataType»

ADEOfAddress

Definition: ADEOfAddress acts as a hook to define properties within an ADE that are to be added to an Address.

Subclass of: None

Stereotype: «DataType»

ADEOfCityModel

Definition: ADEOfCityModel acts as a hook to define properties within an ADE that are to be added to a CityModel.

Subclass of: None

Stereotype: «DataType»

ADEOfClosureSurface

Definition: ADEOfClosureSurface acts as a hook to define properties within an ADE that are to be added to a ClosureSurface.

Subclass of: None

Stereotype: «DataType»

ExternalReference

Definition: ExternalReference is a reference to a corresponding object in another information system, for example in the German cadastre (ALKIS), the German topographic information system (ATKIS), or the OS UK MasterMap®.

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
targetResource	URI [1..1]	Specifies the URI that points to the object in the external information system.
informationSystem	URI [0..1]	Specifies the URI that points to the external information system.
relationType	URI [0..1]	Specifies a URI that additionally qualifies the ExternalReference. The URI can point to a definition from an external ontology (e.g. the sameAs relation from OWL) and allows for mapping the ExternalReference to RDF triples.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Occupancy

Definition: Occupancy is an application-dependent indication of what is contained by a feature.

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
numberOfOccupants	Integer [1..1]	Indicates the number of occupants contained by a feature.
interval	IntervalValue [0..1]	Indicates the time period the occupants are contained by a feature.
occupantType	OccupantTypeValue [0..1]	Indicates the specific type of the occupants that are contained by a feature.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

QualifiedArea

Definition: QualifiedArea is an application-dependent measure of the area of a space or of a thematic surface.

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
area	Area [1..1]	Specifies the value of the QualifiedArea.
typeOfArea	QualifiedAreaType Value [1..1]	Indicates the specific type of the QualifiedArea.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

QualifiedVolume

Definition: QualifiedVolume is an application-dependent measure of the volume of a space.

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
volume	Volume [1..1]	Specifies the value of the QualifiedVolume.
typeOfVolume	QualifiedVolumeType peValue [1..1]	Indicates the specific type of the QualifiedVolume.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

XALAddress

Definition: XALAddress represents address details according to the OASIS xAL standard.

Subclass of: None

Stereotype: «DataType»

9.2.3. Basic Types

Code		
Attribute	Value type and multiplicity	Definition
Definition:	Code is a basic type for a String-based term, keyword, or name that can additionally have a code space.	
Subclass of:	None	
Stereotype:	«BasicType»	
Attribute	Value type and multiplicity	Definition
codeSpace	URI [0..1]	Associates the Code with an authority that controls the term, keyword, or name.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

DoubleBetween0and1

Definition:	DoubleBetween0and1 is a basic type for values, which are greater or equal than 0 and less or equal than 1. The type is used for color encoding, for example.
Subclass of:	None
Stereotype:	«BasicType»
Constraint:	valueBetween0and1 (OCL): inv: DoubleBetween0and1.allInstances() → forAll(p p > = 0 and p < = 1)

DoubleBetween0and1List

Definition:	DoubleBetween0and1List is a basic type that represents a list of double values greater or equal than 0 and less or equal than 1. The type is used for color encoding, for example.
Subclass of:	None
Stereotype:	«BasicType»

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

list	DoubleBetween0an d1 [1..1]	Specifies the list of double values.
------	---	--------------------------------------

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

DoubleList

Definition:	DoubleList is an ordered sequence of double values.
Subclass of:	None
Stereotype:	«BasicType»

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

list	Real [1..1]	Specifies the list of double values.
------	-----------------------------	--------------------------------------

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

DoubleOrNilReasonList

Definition:	DoubleOrNilReasonList is a basic type that represents a list of double values and/or nil reasons.	
Subclass of:	None	
Stereotype:	«BasicType»	
<hr/>		
Attribute	Value type and multiplicity	Definition
list	DoubleOrNilReasonList n [1..1]	Specifies the list of double values and/or nil reasons.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

ID

Definition:	ID is a basic type that represents a unique identifier.
Subclass of:	None
Stereotype:	«BasicType»

IntegerBetween0and3

Definition:	IntegerBetween0and3 is a basic type for integer values, which are greater or equal than 0 and less or equal than 3. The type is used for encoding the LOD number.
Subclass of:	None
Stereotype:	«BasicType»
Constraint:	valueBetween0and3 (OCL): inv: IntegerBetween0and3.allInstances() → forAll(p p > = 0 and p < = 3)

MeasureOrNilReasonList

Definition:	MeasureOrNilReasonList is a basic type that represents a list of double values and/or nil reasons together with a unit of measurement.
Subclass of:	DoubleOrNilReasonList
Stereotype:	«BasicType»

Attribute	Value type and multiplicity	Definition
uom	UnitOfMeasure [1..1]	Specifies the unit of measurement of the double values.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TransformationMatrix2x2

Definition:	TransformationMatrix2x2 is a 2 by 2 matrix represented as a list of four double values in row major order.
Subclass of:	DoubleList
Stereotype:	«BasicType»
Constraint:	lengthOfList (OCL): inv: list → size() = 4

TransformationMatrix3x4

Definition:	TransformationMatrix3x4 is a 3 by 4 matrix represented as a list of twelve double values in row major order.
Subclass of:	DoubleList
Stereotype:	«BasicType»
Constraint:	lengthOfList (OCL): inv: list → size() = 12

TransformationMatrix4x4

Definition:	TransformationMatrix4x4 is a 4 by 4 matrix represented as a list of sixteen double values in row major order.
Subclass of:	DoubleList
Stereotype:	«BasicType»
Constraint:	lengthOfList (OCL): inv: list → size() = 16

9.2.4. Unions

CityModelMember

Definition:	CityModelMember is a union type that enumerates the different types of objects that can occur as members of a city model.	
Stereotype:	«Union»	
Member name	Type	Definition
cityObjectMember	AbstractCityObject [1..1]	Specifies the city objects that are part of the CityModel.
appearanceMember	AbstractAppearance [1..1]	Specifies the appearances of the CityModel.
versionMember	AbstractVersion [1..1]	Specifies the different versions of the CityModel.
versionTransitionMember	AbstractVersionTransition [1..1]	Specifies the transitions between the different versions of the CityModel.
featureMember	AbstractFeature [1..1]	Specifies the feature objects that are part of the CityModel. It allows to include objects that are not derived from a class defined in the CityGML conceptual model, but from the ISO 19109 class AnyFeature.

DoubleOrNilReason

Definition:	DoubleOrNilReason is a union type that allows for choosing between a double value and a nil reason.	
Stereotype:	«Union»	
Member name	Type	Definition
value	Real [1..1]	Specifies the double value.
nilReason	NilReason [1..1]	Specifies the nil reason.

NilReason

Definition:	NilReason is a union type that allows for choosing between two different types of nil reason.	
Stereotype:	«Union»	

Member name	Type	Definition
nilReasonEnum eration	NilReasonEnumera tion [1..1]	Indicates a nil reason that is provided in a code list.
URI	URI [1..1]	Specifies a URI that points to a resource that describes the nil reason.

9.2.5. Code Lists

IntervalValue

Definition: IntervalValue is a code list used to specify a time period.

Stereotype: «CodeList»

MimeTypeValue

Definition:MimeTypeValue is a code list used to specify the MIME type of a referenced resource.

Stereotype: «CodeList»

NilReasonEnumeration

Definition: NilReasonEnumeration is a code list that enumerates the different nil reasons.

Stereotype: «CodeList»

OccupantTypeValue

Definition: OccupantTypeValue is a code list used to classify occupants.

Stereotype: «CodeList»

OtherRelationTypeValue

Definition: OtherRelationTypeValue is a code list used to classify other types of city object relations.

Stereotype: «CodeList»

QualifiedAreaTypeValue

Definition: QualifiedAreaTypeValue is a code list used to specify area types.
Stereotype: «CodeList»

QualifiedVolumeTypeValue

Definition: QualifiedVolumeTypeValue is a code list used to specify volume types.
Stereotype: «CodeList»

RelationTypeValue

Definition: RelationTypeValue is a code list used to classify city object relations.
Stereotype: «CodeList»

TemporalRelationTypeValue

Definition: TemporalRelationTypeValue is a code list used to classify temporal city object relations.
Stereotype: «CodeList»

TopologicalRelationTypeValue

Definition: TopologicalRelationTypeValue is a code list used to classify topological city object relations.
Stereotype: «CodeList»

9.2.6. Enumerations

RelativeToTerrain

Definition: RelativeToTerrain enumerates the spatial relations of a city object relative to terrain in a qualitative way.
StereoType: <<Enumeration>>

Literal value Definition

entirelyAboveTerr	Indicates that the city object is located entirely above the terrain. ain
substantiallyAbov	Indicates that the city object is for the most part located above the terrain. eTerrain
substantiallyAbov	Indicates that the city object is located half above the terrain and half below eAndBelowTerrai the terrain. n
substantiallyBelo	Indicates that the city object is for the most part located below the terrain. wTerrain
entirelyBelowTerr	Indicates that the city object is located entirely below the terrain. ain

RelativeToWater

Definition: RelativeToWater enumerates the spatial relations of a city object relative to the water surface in a qualitative way.

Stereotype: <<Enumeration>>

Literal value Definition

entirelyAboveWat

erSurface

Indicates that the city object is located entirely above the water surface.

substantiallyAbov

eWaterSurface

Indicates that the city object is for the most part located above the water surface.

substantiallyAbov

eAndBelowWater

Surface

Indicates that the city object is located half above the water surface and half below the water surface.

substantiallyBelo

wWaterSurface

Indicates that the city object is for the most part located below the water surface.

entirelyBelowWat

erSurface

Indicates that the city object is located entirely below the water surface.

temporarilyAbove

AndBelowWaterS

urface

Indicates that the city object is temporarily located above or below the water level, because the height of the water surface is varying.

SpaceType

Definition:	SpaceType is an enumeration that characterises a space according to its closure properties.
Stereotype:	<>Enumeration>>

Literal value	Definition
closed	Indicates that the space has boundaries at the bottom, at the top, and on all sides.
open	Indicates that the space has at maximum a boundary at the bottom.
semiOpen	Indicates that the space has a boundary at the bottom and on at least one side.

9.3. Appearance

Description:	The Appearance module supports the modelling of the observable surface properties of CityGML features in the form of textures and material.
Parent Package:	CityGML
Stereotype:	«ApplicationSchema»

9.3.1. Classes

AbstractSurfaceData

Attribute	Value type and multiplicity	Definition
isFront	Boolean [0..1]	Indicates whether the texture or material is assigned to the front side or the back side of the surface geometry object.
adeOfAbstractSurfaceData	ADEOfAbstractSurf aceData [0..*]	Augments AbstractSurfaceData with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractTexture

Definition: AbstractTexture is the abstract superclass to represent the common attributes of the classes ParameterizedTexture and GeoreferencedTexture.

Subclass of: [AbstractSurfaceData](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
imageURI	URI [1..1]	Specifies the URI that points to the external image data file.
mimeType	MimeTypeValue [0..1]	Specifies the MIME type of the external point cloud file.
textureType	TextureType [0..1]	Indicates the specific type of the texture.
wrapMode	WrapMode [0..1]	Specifies the behaviour of the texture when the texture is smaller than the surface to which it is applied.
borderColor	ColorPlusOpacity [0..1]	Specifies the color of that part of the surface that is not covered by the texture.
adeOfAbstract Texture	ADEOfAbstractTexture [0..*]	Augments AbstractTexture with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Appearance

Definition: An Appearance is a collection of surface data, i.e. observable properties for surface geometry objects in the form of textures and material.

Subclass of: [AbstractAppearance](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
surfaceData	AbstractSurfaceData a [*]	Relates to the surface data that are part of the Appearance.

Attribute	Value type and multiplicity	Definition
theme	CharacterString [0..1]	Specifies the topic of the Appearance. Each Appearance contains surface data for one theme only. Examples of themes are infrared radiation, noise pollution, or earthquake-induced structural stress.
adeOfAppearance	ADEOfAppearance [0..*]	Augments the Appearance with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

GeoreferencedTexture

Definition:	A GeoreferencedTexture is a texture that uses a planimetric projection. It contains an implicit parameterization that is either stored within the image file, an accompanying world file or specified using the orientation and referencePoint elements.
Subclass of:	AbstractTexture
Stereotype:	«FeatureType»

Role name	Target class and multiplicity	Definition
referencePoint	GM_Point [0..1]	Relates to the 2D Point geometry that represents the center of the upper left image pixel in world space.

Attribute	Value type and multiplicity	Definition
preferWorldFile	Boolean [0..1]	Indicates whether the georeference from the image file or the accompanying world file should be preferred.
orientation	TransformationMatrix2x2 [0..1]	Specifies the rotation and scaling of the image in form of a 2x2 matrix.
target	URI [0..*]	Specifies the URI that points to the surface geometry objects to which the texture is applied.
adeOfGeoreferencedTexture	ADEOfGeoreferencedTexture [0..*]	Augments the GeoreferencedTexture with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

ParameterizedTexture

Definition: A ParameterizedTexture is a texture that uses texture coordinates or a transformation matrix for parameterization.

Subclass of: [AbstractTexture](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
textureParam eterization	AbstractTexturePar ameterization [*]	Relates to the texture coordinates or transformation matrices used for parameterization.
Attribute	Value type and multiplicity	Definition
adeOfParamet erizedTexture	ADEOfParameteriz edTexture [0..*]	Augments the ParameterizedTexture with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TextureAssociation

Definition: TextureAssociation denotes the relation of a texture to a surface geometry object.

Subclass of: None

Stereotype: «ObjectType»

Attribute	Value type and multiplicity	Definition
target	URI [1..1]	Specifies the URI that points to the surface geometry object to which the texture is applied.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

X3DMaterial

Definition:	X3DMaterial defines properties for surface geometry objects based on the material definitions from the standards X3D and COLLADA.	
Subclass of:	AbstractSurfaceData	
Stereotype:	«FeatureType»	
<hr/>		
Attribute	Value type and multiplicity	Definition
ambientIntensity	DoubleBetween0and1 [0..1]	Specifies the minimum percentage of diffuseColor that is visible regardless of light sources.
diffuseColor	Color [0..1]	Specifies the color of the light diffusely reflected by the surface geometry object.
emissiveColor	Color [0..1]	Specifies the color of the light emitted by the surface geometry object.
specularColor	Color [0..1]	Specifies the color of the light directly reflected by the surface geometry object.
shininess	DoubleBetween0and1 [0..1]	Specifies the sharpness of the specular highlight.
transparency	DoubleBetween0and1 [0..1]	Specifies the degree of transparency of the surface geometry object.
isSmooth	Boolean [0..1]	Specifies which interpolation method is used for the shading of the surface geometry object. If the attribute is set to true, vertex normals should be used for shading (Gouraud shading). Otherwise, normals should be constant for a surface patch (flat shading).
target	URI [0..*]	Specifies the URI that points to the surface geometry objects to which the material is applied.
adeOfX3DMaterial	ADEOfX3DMaterial [0..*]	Augments the X3DMaterial with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

9.3.2. Data Types

AbstractTextureParameterization

Definition:	AbstractTextureParameterization is the abstract superclass for different kinds of texture parameterizations.
Subclass of:	None
Stereotype:	«DataType»

ADEOfAbstractSurfaceData

Definition: ADEOfAbstractSurfaceData acts as a hook to define properties within an ADE that are to be added to AbstractSurfaceData.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractTexture

Definition: ADEOfAbstractTexture acts as a hook to define properties within an ADE that are to be added to AbstractTexture.

Subclass of: None

Stereotype: «DataType»

ADEOfAppearance

Definition: ADEOfAppearance acts as a hook to define properties within an ADE that are to be added to an Appearance.

Subclass of: None

Stereotype: «DataType»

ADEOfGeoreferencedTexture

Definition: ADEOfGeoreferencedTexture acts as a hook to define properties within an ADE that are to be added to a GeoreferencedTexture.

Subclass of: None

Stereotype: «DataType»

ADEOfParameterizedTexture

Definition: ADEOfParameterizedTexture acts as a hook to define properties within an ADE that are to be added to a ParameterizedTexture.

Subclass of: None

Stereotype: «DataType»

ADEOfX3DMaterial

Definition: ADEOfX3DMaterial acts as a hook to define properties within an ADE that are to be added to an X3DMaterial.

Subclass of: None

Stereotype: «DataType»

TexCoordGen

Definition: TexCoordGen defines texture parameterization using a transformation matrix.

Subclass of: None

Stereotype: «DataType»

Role name	Target class and multiplicity	Definition
crs	SC_CRS [0..1]	Relates to the coordinate reference system of the transformation matrix.

Attribute	Value type and multiplicity	Definition
worldToTextu re	TransformationMa trix3x4 [1..1]	Specifies the 3x4 transformation matrix that defines the transformation between world coordinates and texture coordinates.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TexCoordList

Definition: TexCoordList defines texture parameterization using texture coordinates.

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
textureCoordinates	DoubleList [1..*]	Specifies the coordinates of texture used for parameterization. The texture coordinates are provided separately for each LinearRing of the surface geometry object.
ring	URI [1..*]	Specifies the URIs that point to the LinearRings that are parameterized using the given texture coordinates.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

9.3.3. Basic Types

Color	
Definition:	Color is a list of three double values between 0 and 1 defining an RGB color value.
Subclass of:	DoubleBetween0and1List
Stereotype:	«BasicType»
Constraint:	<code>lengthOfList (OCL): inv: list → size() = 3</code>

ColorPlusOpacity	
Definition:	Color is a list of four double values between 0 and 1 defining an RGBA color value. Opacity value of 0 means transparent.
Subclass of:	DoubleBetween0and1List
Stereotype:	«BasicType»
Constraint:	<code>lengthOfList (OCL): inv: list → size() = 3 or list → size() = 4</code>

9.3.4. Unions

none

9.3.5. Code Lists

none

9.3.6. Enumerations

TextureType

Definition: TextureType enumerates the different texture types.

Stereotype: <>Enumeration>>

Literal value	Definition
specific	Indicates that the texture is specific to a single surface.
typical	Indicates that the texture is characteristic of a surface and can be used repeatedly.
unknown	Indicates that the texture type is not known.

WrapMode

Definition: WrapMode enumerates the different fill modes for textures.

Stereotype: <>Enumeration>>

Literal value	Definition
none	Indicates that the texture is applied to the surface "as is". The part of the surface that is not covered by the texture is shown fully transparent. [cf. COLLADA]
wrap	Indicates that the texture is repeated until the surface is fully covered. [cf. COLLADA]
mirror	Indicates that the texture is repeated and mirrored. [cf. COLLADA]
clamp	Indicates that the texture is stretched to the edges of the surface. [cf. COLLADA]
border	Indicates that the texture is applied to the surface "as is". The part of the surface that is not covered by the texture is filled with the RGBA color that is specified in the attribute borderColor. [cf. COLLADA]

9.4. CityFurniture

Description: The CityFurniture module supports representation of city furniture objects. City furniture objects are immovable objects like lanterns, traffic signs, advertising columns, benches, or bus stops that can be found in traffic areas, residential areas, on squares, or in built-up areas.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.4.1. Classes

CityFurniture

Definition:	CityFurniture is an object or piece of equipment installed in the outdoor environment for various purposes. Examples include street signs, traffic signals, street lamps, benches, fountains.	
Subclass of:	AbstractOccupiedSpace	
Stereotype:	«TopLevelFeatureType»	
Attribute	Value type and multiplicity	Definition
class	CityFurnitureClass Value [0..1]	Indicates the specific type of the CityFurniture.
function	CityFurnitureFunct ionValue [0..*]	Specifies the intended purposes of the CityFurniture.
usage	CityFurnitureUsage Value [0..*]	Specifies the actual uses of the CityFurniture.
adeOfCityFurniture	ADEOfCityFurniture e [0..*]	Augments the CityFurniture with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.4.2. Data Types

ADEOfCityFurniture

Definition:	ADEOfCityFurniture acts as a hook to define properties within an ADE that are to be added to a CityFurniture.
Subclass of:	None
Stereotype:	«DataType»

9.4.3. Basic Types

none

9.4.4. Unions

none

9.4.5. Code Lists

CityFurnitureClassValue

Definition: CityFurnitureClassValue is a code list used to further classify a CityFurniture.
Stereotype: «CodeList»

CityFurnitureFunctionValue

Definition: CityFurnitureFunctionValue is a code list that enumerates the different purposes of a CityFurniture.
Stereotype: «CodeList»

CityFurnitureUsageValue

Definition: CityFurnitureUsageValue is a code list that enumerates the different uses of a CityFurniture.
Stereotype: «CodeList»

9.4.6. Enumerations

none

9.5. CityObjectGroup

Description: The CityObjectGroup module supports grouping of city objects. Arbitrary city objects may be aggregated in groups according to user-defined criteria. A group may be further classified by application-specific attributes.
Parent Package: CityGML
Stereotype: «ApplicationSchema»

9.5.1. Classes

CityObjectGroup

Definition:	A CityObjectGroup represents an application-specific aggregation of city objects according to some user-defined criteria. Examples for groups are the buildings in a specific region, the result of a query, or objects put together for visualization purposes. Each member of a group may be qualified by a role name, reflecting the role each city object plays in the context of the group.	
Subclass of:	AbstractLogicalSpace	
Stereotype:	«TopLevelFeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
parent	AbstractCityObject [0..1]	Relates to a city object to which the CityObjectGroup belongs.
groupMember	AbstractCityObject [*]	Relates to the city objects that are part of the CityObjectGroup.
<hr/>		
Attribute	Value type and multiplicity	Definition
class	CityObjectGroupClass [0..1]	Indicates the specific type of the CityObjectGroup.
function	CityObjectGroupFunction [0..*]	Specifies the intended purposes of the CityObjectGroup.
usage	CityObjectGroupUsage [0..*]	Specifies the actual usages of the CityObjectGroup.
adeOfCityObjectGroup	ADEOfCityObjectGroup [0..*]	Augments the CityObjectGroup with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

Role		
Attribute	Value type and multiplicity	Definition
Definition:	Role qualifies the function of a city object within the CityObjectGroup.	
Subclass of:	None	
Stereotype:	«ObjectType»	
<hr/>		
Attribute	Value type and multiplicity	Definition
role	CharacterString [0..1]	Describes the role the city object plays within the CityObjectGroup.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.5.2. Data Types

ADEOfCityObjectGroup

Definition:	ADEOfCityObjectGroup acts as a hook to define properties within an ADE that are to be added to a CityObjectGroup.
Subclass of:	None
Stereotype:	«DataType»

9.5.3. Basic Types

none

9.5.4. Unions

none

9.5.5. Code Lists

CityObjectGroupClassValue

Definition:	CityObjectGroupClassValue is a code list used to further classify a CityObjectGroup.
Stereotype:	«CodeList»

CityObjectGroupFunctionValue

Definition:	CityObjectGroupFunctionValue is a code list that enumerates the different purposes of a CityObjectGroup.
Stereotype:	«CodeList»

CityObjectGroupUsageValue

Definition:	CityObjectGroupUsageValue is a code list that enumerates the different uses of a CityObjectGroup.
Stereotype:	«CodeList»

9.5.6. Enumerations

none

9.6. Dynamizer

- Description: The Dynamizer module supports the injection of timeseries data for individual attributes of CityGML features. Timeseries data can either be retrieved from external Sensor APIs (e.g. OGC SensorThings API, OGC Sensor Observation Services, MQTT, proprietary platforms), external standardized timeseries files (e.g. OGC TimeseriesML or OGC Observations & Measurements), external tabulated files (e.g CSV) or can be represented inline as basic time-value pairs.
- Parent Package: CityGML
- Stereotype: «ApplicationSchema»

9.6.1. Classes

AbstractAtomicTimeseries

- Definition: AbstractAtomicTimeseries represents the attributes and relationships that are common to all kinds of atomic timeseries (GenericTimeseries, TabulatedFileTimeseries, StandardFileTimeseries). An atomic timeseries represents time-varying data of a specific data type for a single contiguous time interval.
- Subclass of: [AbstractTimeseries](#)
- Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
observationProperty	CharacterString [1..1]	Specifies the phenomenon for which the atomic timeseries provides observation values.
uom	CharacterString [0..1]	Specifies the unit of measurement of the observation values.
adeOfAbstractAtomicTimeseries	ADEOfAbstractAtomicTimeseries [0..*]	Augments AbstractAtomicTimeseries with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractTimeseries

Definition:	AbstractTimeseries is the abstract superclass representing any type of timeseries data.																			
Subclass of:	AbstractFeature																			
Stereotype:	«FeatureType»																			
<table border="1"> <thead> <tr> <th>Attribute</th> <th>Value type and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>firstTimestamp</td> <td>TM_Position [0..1]</td> <td>Specifies the beginning of the timeseries.</td> </tr> <tr> <td>p</td> <td></td> <td></td> </tr> <tr> <td>lastTimestamp</td> <td>TM_Position [0..1]</td> <td>Specifies the end of the timeseries.</td> </tr> <tr> <td>p</td> <td></td> <td></td> </tr> <tr> <td>adeOfAbstractTimeseries</td> <td>ADEOfAbstractTimeseries [0..*]</td> <td>Augments AbstractTimeseries with properties defined in an ADE.</td> </tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	firstTimestamp	TM_Position [0..1]	Specifies the beginning of the timeseries.	p			lastTimestamp	TM_Position [0..1]	Specifies the end of the timeseries.	p			adeOfAbstractTimeseries	ADEOfAbstractTimeseries [0..*]	Augments AbstractTimeseries with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition																		
firstTimestamp	TM_Position [0..1]	Specifies the beginning of the timeseries.																		
p																				
lastTimestamp	TM_Position [0..1]	Specifies the end of the timeseries.																		
p																				
adeOfAbstractTimeseries	ADEOfAbstractTimeseries [0..*]	Augments AbstractTimeseries with properties defined in an ADE.																		
<p>Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».</p>																				

CompositeTimeseries								
Definition:	A CompositeTimeseries is a (possibly recursive) aggregation of atomic and composite timeseries. The components of a composite timeseries must have non-overlapping time intervals.							
Subclass of:	AbstractTimeseries							
Stereotype:	«FeatureType»							
<table border="1"> <thead> <tr> <th>Role name</th> <th>Target class and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>component</td> <td>TimeseriesComponent [1..*]</td> <td>Relates to the atomic and composite timeseries that are part of the CompositeTimeseries. The referenced timeseries are sequentially ordered.</td> </tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	component	TimeseriesComponent [1..*]	Relates to the atomic and composite timeseries that are part of the CompositeTimeseries. The referenced timeseries are sequentially ordered.
Role name	Target class and multiplicity	Definition						
component	TimeseriesComponent [1..*]	Relates to the atomic and composite timeseries that are part of the CompositeTimeseries. The referenced timeseries are sequentially ordered.						
Attribute	Value type and multiplicity	Definition						
adeOfCompositeTimeseries	ADEOfCompositeTimeseries [0..*]	Augments the CompositeTimeseries with properties defined in an ADE.						
<p>Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».</p>								

Dynamizer

Definition:	A Dynamizer is an object that injects timeseries data for an individual attribute of the city object in which it is included. The timeseries data overrides the static value of the referenced city object attribute in order to represent dynamic, i.e. time-dependent, variations of its value.	
Subclass of:	AbstractDynamizer	
Stereotype:	«FeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
dynamicData	AbstractTimeseries [0..1]	Relates to the timeseries data that is given either inline within a CityGML dataset or by a link to an external file containing timeseries data.
sensorConnection	SensorConnection [0..1]	Relates to the sensor API that delivers timeseries data.
<hr/>		
Attribute	Value type and multiplicity	Definition
attributeRef	CharacterString [1..1]	Specifies the attribute of a CityGML feature whose value is overridden or replaced by the (dynamic) values specified by the Dynamizer.
startTime	TM_Position [0..1]	Specifies the beginning of the time span for which the Dynamizer provides dynamic values.
endTime	TM_Position [0..1]	Specifies the end of the time span for which the Dynamizer provides dynamic values.
adeOfDynamizer	ADEOfDynamizer [0..*]	Augments the Dynamizer with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

GenericTimeseries

Definition:	A GenericTimeseries represents time-varying data in the form of embedded time-value-pairs of a specific data type for a single contiguous time interval.
Subclass of:	AbstractAtomicTimeseries
Stereotype:	«FeatureType»
Constraint:	dataTypeOfValue (OCL): inv: if valueType = TimeseriesTypeValue::integer then TimeValuePair → forAll(c c.intValue → size()=1) else if valueType = TimeseriesTypeValue::double then TimeValuePair → forAll(c c.doubleValue → size()=1) else if valueType = TimeseriesTypeValue::string then TimeValuePair → forAll(c c.stringValue → size()=1) else if valueType = TimeseriesTypeValue::geometry then TimeValuePair → forAll(c c.geometryValue → size()=1) else if valueType = TimeseriesTypeValue::uri then TimeValuePair → forAll(c c.uriValue → size()=1) else if valueType = TimeseriesTypeValue::bool then TimeValuePair → forAll(c c.boolValue → size()=1) else if valueType = TimeseriesTypeValue::implicitGeometry then TimeValuePair → forAll(c c.implicitGeometryValue → size()=1) else TimeValuePair → forAll(c c.appearanceValue → size()=1) endif endif endif endif endif endif endif endif

Role name	Target class and multiplicity	Definition
timeValuePair	TimeValuePair [1..*]	Relates to the time-value-pairs that are part of the GenericTimeseries.

Attribute	Value type and multiplicity	Definition
valueType	TimeseriesTypeValue [1..1]	Indicates the specific type of all time-value-pairs that are part of the GenericTimeseries.
adeOfGenericTimeseries	ADEOfGenericTimeseries [0..*]	Augments the GenericTimeseries with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

StandardFileTimeseries

Definition:	A StandardFileTimeseries represents time-varying data for a single contiguous time interval. The data is provided in an external file referenced in the StandardFileTimeseries. The data within the external file shall be encoded according to a dedicated format for the representation of timeseries data, for example, the OGC TimeseriesML or OGC Observations & Measurements standard. The data type of the data has to be specified within the external file.	
Subclass of:	AbstractAtomicTimeseries	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
fileLocation	URI [1..1]	Specifies the URI that points to the external timeseries file.
fileType	StandardFileTypeValue [1..1]	Specifies the format used to represent the timeseries data.
mimeType	MimeTypeValue [0..1]	Specifies the MIME type of the external timeseries file.
adeOfStandar dFileTimeseri es	ADEOfStandardFile Timeseries [0..*]	Augments the StandardFileTimeseries with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TabulatedFileTimeseries

Definition:	A TabulatedFileTimeseries represents time-varying data of a specific data type for a single contiguous time interval. The data is provided in an external file referenced in the TabulatedFileTimeseries. The file shall contain table structured data using an appropriate file format like comma-separated values (CSV), Microsoft Excel (XLSX) or Google Spreadsheet. The timestamps and the values are given in specific columns of the table. Each row represents a single time-value-pair. A subset of rows can be selected using the idColumn and idValue attributes.
Subclass of:	AbstractAtomicTimeseries
Stereotype:	«FeatureType»
Constraint:	columnNumberOrColumnName (OCL): inv: (timeColumnNo → notEmpty() or timeColumnName → notEmpty()) and (valueColumnNo → notEmpty() or valueColumnName → notEmpty()) and (idValue → notEmpty()) implies idColumnNo → notEmpty() or idColumnName → notEmpty()

Attribute	Value type and multiplicity	Definition
fileLocation	URI [1..1]	Specifies the URI that points to the external timeseries file.
fileType	TabulatedFileType Value [1..1]	Specifies the format used to represent the timeseries data.
mimeType	MimeTypeValue [0..1]	Specifies the MIME type of the external timeseries file.
valueType	TimeseriesTypeVal ue [1..1]	Indicates the specific type of the timeseries data.
numberOfHea derLines	Integer [0..1]	Indicates the number of lines at the beginning of the tabulated file that represent headers.
fieldSeparator	CharacterString [1..1]	Indicates which symbol is used to separate the individual values in the tabulated file.
decimalSymb ol	Character [0..1]	Indicates which symbol is used to separate the integer part from the fractional part of a decimal number.
idColumnNo	Integer [0..1]	Specifies the number of the column that stores the identifier of the time-value-pair.
idColumnName	CharacterString e [0..1]	Specifies the name of the column that stores the identifier of the time-value-pair.
idValue	CharacterString [0..1]	Specifies the value of the identifier for which the time-value-pairs are to be selected.
timeColumnN o	Integer [0..1]	Specifies the number of the column that stores the timestamp of the time-value-pair.
timeColumnName	CharacterString ame [0..1]	Specifies the name of the column that stores the timestamp of the time-value-pair.
valueColumn No	Integer [0..1]	Specifies the number of the column that stores the value of the time-value-pair.
valueColumnName	CharacterString Name [0..1]	Specifies the name of the column that stores the value of the time-value-pair.
adeOfTabulat edFileTimeser ies	ADEOfTabulatedFil eTimeseries [0..*]	Augments the TabulatedFileTimeseries with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.6.2. Data Types

ADEOfAbstractAtomicTimeseries

Definition: ADEOfAbstractAtomicTimeseries acts as a hook to define properties within an ADE that are to be added to AbstractAtomicTimeseries.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractTimeseries

Definition: ADEOfAbstractTimeseries acts as a hook to define properties within an ADE that are to be added to AbstractTimeseries.

Subclass of: None

Stereotype: «DataType»

ADEOfCompositeTimeseries

Definition: ADEOfCompositeTimeseries acts as a hook to define properties within an ADE that are to be added to a CompositeTimeseries.

Subclass of: None

Stereotype: «DataType»

ADEOfDynamizer

Definition: ADEOfDynamizer acts as a hook to define properties within an ADE that are to be added to a Dynamizer.

Subclass of: None

Stereotype: «DataType»

ADEOfGenericTimeseries

Definition: ADEOfGenericTimeseries acts as a hook to define properties within an ADE that are to be added to a GenericTimeseries.

Subclass of: None

Stereotype: «DataType»

ADEOfStandardFileTimeseries

Definition: ADEOfStandardFileTimeseries acts as a hook to define properties within an ADE that are to be added to a StandardFileTimeseries.

Subclass of: None

Stereotype: «DataType»

ADEOfTabulatedFileTimeseries

Definition: ADEOfTabulatedFileTimeseries acts as a hook to define properties within an ADE that are to be added to a TabulatedFileTimeseries.

Subclass of: None

Stereotype: «DataType»

SensorConnection

Definition: A SensorConnection provides all details that are required to retrieve a specific datastream from an external sensor web service. It comprises the service type (e.g. OGC SensorThings API, OGC Sensor Observation Services, MQTT, proprietary platforms), the URL of the sensor service, the identifier for the sensor or thing, and its observed property as well as information about the required authentication method.

Subclass of: None

Stereotype: «DataType»

Role name	Target class and multiplicity	Definition
-----------	-------------------------------	------------

sensorLocation	AbstractCityObject n	Relates the sensor to the city object where it is located. [0..1]
----------------	---	--

Attribute	Value type and multiplicity	Definition
connectionType	SensorConnectionTypeValue [1..1]	Indicates the type of Sensor API to which the SensorConnection refers.
observationProperty	CharacterString [1..1]	Specifies the phenomenon for which the SensorConnection provides observations.
uom	CharacterString [0..1]	Specifies the unit of measurement of the observations.
sensorID	CharacterString [0..1]	Specifies the unique identifier of the sensor from which the SensorConnection retrieves observations.
sensorName	CharacterString [0..1]	Specifies the name of the sensor from which the SensorConnection retrieves observations.
observationID	CharacterString [0..1]	Specifies the unique identifier of the observation that is retrieved by the SensorConnection.
datastreamID	CharacterString [0..1]	Specifies the datastream that is retrieved by the SensorConnection.
baseURL	URI [0..1]	Specifies the base URL of the Sensor API request.
authType	AuthenticationTypeValue [0..1]	Specifies the type of authentication required to be able to access the Sensor API.
mqttServer	CharacterString [0..1]	Specifies the name of the MQTT Server. This attribute is relevant when the MQTT Protocol is used to connect to a Sensor API.
mqttTopic	CharacterString [0..1]	Names the specific datastream that is retrieved by the SensorConnection. This attribute is relevant when the MQTT Protocol is used to connect to a Sensor API.
linkToObservation	CharacterString [0..1]	Specifies the complete URL to the observation request.
linkToSensorDescription	CharacterString [0..1]	Specifies the complete URL to the sensor description request.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TimeseriesComponent

Definition:	TimeseriesComponent represents an element of a CompositeTimeseries.
Subclass of:	None
Stereotype:	«DataType»

Role name	Target class and multiplicity	Definition
timeseries	AbstractTimeseries [1..1]	Relates a timeseries to the TimeseriesComponent.
Attribute	Value type and multiplicity	Definition
repetitions	Integer [1..1]	Specifies how often the timeseries that is referenced by the TimeseriesComponent should be iterated.
additionalGap	TM_Duration [0..1]	Specifies how much extra time is added after all repetitions as an additional gap.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TimeValuePair

Definition:	A TimeValuePair represents a value that is valid for a given timepoint. For each TimeValuePair, only one of the value properties can be used mutually exclusive. Which value property has to be provided depends on the selected value type in the GenericTimeSeries feature, in which the TimeValuePair is included.
Subclass of:	None
Stereotype:	«DataType»
Constraint:	singleValue (OCL): inv: intValue → size() + doubleValue → size() + stringValue → size() + geometryValue → size() + uriValue → size() + boolValue → size() + implicitGeometryValue → size() + appearanceValue → size() = 1

Attribute	Value type and multiplicity	Definition
timestamp	TM_Position [1..1]	Specifies the timepoint at which the value of the TimeValuePair is valid.
intValue	Integer [0..1]	Specifies the "Integer" value of the TimeValuePair.
doubleValue	Real [0..1]	Specifies the "Double" value of the TimeValuePair.
stringValue	CharacterString [0..1]	Specifies the "String" value of the TimeValuePair.
geometryValue	GM_Object [0..1]	Specifies the geometry value of the TimeValuePair.
uriValue	URI [0..1]	Specifies the "URI" value of the TimeValuePair.
boolValue	Boolean [0..1]	Specifies the "Boolean" value of the TimeValuePair.
implicitGeometryValue	ImplicitGeometry [0..1]	Specifies the "ImplicitGeometry" value of the TimeValuePair.
appearanceValue	AbstractAppearance [0..1]	Specifies the "Appearance" value of the TimeValuePair.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.6.3. Basic Types

none

9.6.4. Unions

none

9.6.5. Code Lists

AuthenticationTypeValue	
Definition:	AuthenticationTypeValue is a code list used to specify the authentication method to be used to access the referenced sensor service. Each value shall provide enough information such that a software application could determine the required access credentials.
Stereotype:	«CodeList»

SensorConnectionTypeValue

Definition:	SensorConnectionTypeValue is a code list used to specify the type of the referenced sensor service. Each value shall provide enough information such that a software application would be able to identify the API type and version.
Stereotype:	«CodeList»

StandardFileTypeValue

Definition:	StandardFileTypeValue is a code list used to specify the type of the referenced external timeseries data file. Each value shall provide information about the standard and version.
Stereotype:	«CodeList»

TabulatedFileTypeValue

Definition:	TabulatedFileTypeValue is a code list used to specify the data format of the referenced external tabulated data file.
Stereotype:	«CodeList»

9.6.6. Enumerations

TimeseriesTypeValue

Definition:	TimeseriesTypeValue enumerates the possible value types for GenericTimeseries and TimeValuePair.
Stereotype:	<>Enumeration>

Literal value	Definition
int	Indicates that the values of the GenericTimeseries are of type "Integer".
double	Indicates that the values of the GenericTimeseries are of type "Double".
string	Indicates that the values of the GenericTimeseries are of type "String".
geometry	Indicates that the values of the GenericTimeseries are geometries.
uri	Indicates that the values of the GenericTimeseries are of type "URI".
bool	Indicates that the values of the GenericTimeseries are of type "Boolean".
implicitGeometry	Indicates that the values of the GenericTimeseries are of type "ImplicitGeometry".
appearance	Indicates that the values of the GenericTimeseries are of type "Appearance".

9.7. Generics

Description:	The Generics module supports application-specific extensions to the CityGML data model. These extensions may be used to model and exchange additional attributes and features not covered by the predefined thematic classes of CityGML. Generic extensions shall only be used if appropriate thematic classes or attributes are not provided by any other CityGML module.
Parent Package:	CityGML
Stereotype:	«ApplicationSchema»

9.7.1. Classes

GenericLogicalSpace

Definition: A GenericLogicalSpace is a space that is not represented by any explicitly modelled AbstractLogicalSpace subclass within CityGML.

Subclass of: [AbstractLogicalSpace](#)

Stereotype: «TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
class	GenericLogicalSpace eClassValue [0..1]	Indicates the specific type of the GenericLogicalSpace.
function	GenericLogicalSpace eFunctionValue [0..*]	Specifies the intended purposes of the GenericLogicalSpace.
usage	GenericLogicalSpace eUsageValue [0..*]	Specifies the actual uses of the GenericLogicalSpace.
adeOfGenericLogicalSpace	ADEOfGenericLogicalSpace [0..*]	Augments the GenericLogicalSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

GenericOccupiedSpace

Definition:	A GenericOccupiedSpace is a space that is not represented by any explicitly modelled AbstractOccupiedSpace subclass within CityGML.
Subclass of:	AbstractOccupiedSpace
Stereotype:	«TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
class	GenericOccupiedSp [0..1]	Indicates the specific type of the GenericOccupiedSpace. aceClassValue
function	GenericOccupiedSp [0..*]	Specifies the intended purposes of the aceFunctionValue GenericOccupiedSpace.
usage	GenericOccupiedSp [0..*]	Specifies the actual uses of the GenericOccupiedSpace. aceUsageValue
adeOfGeneric OccupiedSpace	ADEOfGenericOcc piedSpace [0..*]	Augments the GenericOccupiedSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

GenericThematicSurface

Definition:	A GenericThematicSurface is a surface that is not represented by any explicitly modelled AbstractThematicSurface subclass within CityGML.
Subclass of:	AbstractThematicSurface
Stereotype:	«TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
class	GenericThematicSu [0..1]	Indicates the specific type of the GenericThematicSurface. rfaceClassValue
function	GenericThematicSu [0..*]	Specifies the intended purposes of the rfaceFunctionValue GenericThematicSurface.
usage	GenericThematicSu [0..*]	Specifies the actual uses of the GenericThematicSurface. rfaceUsageValue
adeOfGeneric ThematicSurface	ADEOfGenericThe maticSurface [0..*]	Augments the GenericThematicSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

GenericUnoccupiedSpace

Definition: A GenericUnoccupiedSpace is a space that is not represented by any explicitly modelled AbstractUnoccupiedSpace subclass within CityGML.

Subclass of: [AbstractUnoccupiedSpace](#)

Stereotype: «TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
class	GenericUnoccupied SpaceClassValue [0..1]	Indicates the specific type of the GenericUnoccupiedSpace.
function	GenericUnoccupied SpaceFunctionValue e [0..*]	Specifies the intended purposes of the GenericUnoccupiedSpace.
usage	GenericUnoccupied SpaceUsageValue [0..*]	Specifies the actual uses of the GenericUnoccupiedSpace.
adeOfGeneric UnoccupiedSp ace	ADEOfGenericUnoc cupiedSpace [0..*]	Augments the GenericUnoccupiedSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.7.2. Data Types

ADEOfGenericLogicalSpace

Definition: ADEOfGenericLogicalSpace acts as a hook to define properties within an ADE that are to be added to a GenericLogicalSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfGenericOccupiedSpace

Definition: ADEOfGenericOccupiedSpace acts as a hook to define properties within an ADE that are to be added to a GenericOccupiedSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfGenericThematicSurface

Definition: ADEOfGenericThematicSurface acts as a hook to define properties within an ADE that are to be added to a GenericThematicSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfGenericUnoccupiedSpace

Definition: ADEOfGenericUnoccupiedSpace acts as a hook to define properties within an ADE that are to be added to a GenericUnoccupiedSpace.

Subclass of: None

Stereotype: «DataType»

CodeAttribute

Definition: CodeAttribute is a data type used to define generic attributes of type "Code".

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the CodeAttribute.
value	Code [1..1]	Specifies the "Code" value.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

DateAttribute

Definition: DateAttribute is a data type used to define generic attributes of type "Date".

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the DateAttribute.
value	Date [1..1]	Specifies the "Date" value.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

DoubleAttribute

Definition: DoubleAttribute is a data type used to define generic attributes of type "Double".

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the DoubleAttribute.
value	Real [1..1]	Specifies the "Double" value.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

GenericAttributeSet

Definition: A GenericAttributeSet is a named collection of generic attributes.

Subclass of: None

Stereotype: «DataType»

Role name	Target class and multiplicity	Definition
genericAttribute	AbstractGenericAttribute [1..*]	Relates to the generic attributes that are part of the GenericAttributeSet.

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the GenericAttributeSet.
codeSpace	URI [0..1]	Associates the GenericAttributeSet with an authority that maintains the collection of generic attributes.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

IntAttribute

Definition: IntAttribute is a data type used to define generic attributes of type "Integer".
 Subclass of: None
 Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the IntAttribute.
value	Integer [1..1]	Specifies the "Integer" value.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

MeasureAttribute

Definition: MeasureAttribute is a data type used to define generic attributes of type "Measure".
 Subclass of: None
 Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the MeasureAttribute.
value	Measure [1..1]	Specifies the value of the MeasureAttribute. The value is of type "Measure", which can additionally provide the units of measure. [cf. ISO 19103]

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

StringAttribute

Definition: StringAttribute is a data type used to define generic attributes of type "String".

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the StringAttribute.
value	CharacterString [1..1]	Specifies the "String" value.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

UriAttribute

Definition: UriAttribute is a data type used to define generic attributes of type "URI".

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the UriAttribute.
value	URI [1..1]	Specifies the "URI" value.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.7.3. Basic Types

none

9.7.4. Unions

none

9.7.5. Code Lists

GenericLogicalSpaceClassValue

Definition: GenericLogicalSpaceClassValue is a code list used to further classify a GenericLogicalSpace.

Stereotype: «CodeList»

GenericLogicalSpaceFunctionValue

Definition: GenericLogicalSpaceFunctionValue is a code list that enumerates the different purposes of a GenericLogicalSpace.

Stereotype: «CodeList»

GenericLogicalSpaceUsageValue

Definition: GenericLogicalSpaceUsageValue is a code list that enumerates the different uses of a GenericLogicalSpace.

Stereotype: «CodeList»

GenericOccupiedSpaceClassValue

Definition: GenericOccupiedSpaceClassValue is a code list used to further classify a GenericOccupiedSpace.

Stereotype: «CodeList»

GenericOccupiedSpaceFunctionValue

Definition: GenericOccupiedSpaceFunctionValue is a code list that enumerates the different purposes of a GenericOccupiedSpace.

Stereotype: «CodeList»

GenericOccupiedSpaceUsageValue

Definition: GenericOccupiedSpaceUsageValue is a code list that enumerates the different uses of a GenericOccupiedSpace.

Stereotype: «CodeList»

GenericThematicSurfaceClassValue

Definition: GenericThematicSurfaceClassValue is a code list used to further classify a GenericThematicSurface.

Stereotype: «CodeList»

GenericThematicSurfaceFunctionValue

Definition: GenericThematicSurfaceFunctionValue is a code list that enumerates the different purposes of a GenericThematicSurface.

Stereotype: «CodeList»

GenericThematicSurfaceUsageValue

Definition: GenericThematicSurfaceUsageValue is a code list that enumerates the different uses of a GenericThematicSurface.

Stereotype: «CodeList»

GenericUnoccupiedSpaceClassValue

Definition: GenericUnoccupiedSpaceClassValue is a code list used to further classify a GenericUnoccupiedSpace.

Stereotype: «CodeList»

GenericUnoccupiedSpaceFunctionValue

Definition: GenericUnoccupiedSpaceFunctionValue is a code list that enumerates the different purposes of a GenericUnoccupiedSpace.

Stereotype: «CodeList»

GenericUnoccupiedSpaceUsageValue

Definition: GenericUnoccupiedSpaceUsageValue is a code list that enumerates the different uses of a GenericUnoccupiedSpace.

Stereotype: «CodeList»

9.7.6. Enumerations

none

9.8. LandUse

Description: The LandUse module supports representation of areas of the earth's surface dedicated to a specific land use.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.8.1. Classes

LandUse

Definition: A LandUse object is an area of the earth's surface dedicated to a specific land use or having a specific land cover with or without vegetation, such as sand, rock, mud flats, forest, grasslands, or wetlands.

Subclass of: [AbstractThematicSurface](#)

Stereotype: «TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
class	LandUseClassValue [0..1]	Indicates the specific type of the LandUse.
function	LandUseFunctionValue [0..*]	Specifies the intended purposes of the LandUse.
usage	LandUseUsageValue [0..*]	Specifies the actual uses of the LandUse.
adeOfLandUse	ADEOfLandUse [0..*]	Augments the LandUse with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.8.2. Data Types

ADEOfLandUse

Definition:	ADEOfLandUse acts as a hook to define properties within an ADE that are to be added to a LandUse.
Subclass of:	None
Stereotype:	«DataType»

9.8.3. Basic Types

none

9.8.4. Unions

none

9.8.5. Code Lists

LandUseClassValue

Definition:	LandUseClassValue is a code list used to further classify a LandUse.
Stereotype:	«CodeList»

LandUseFunctionValue

Definition:	LandUseFunctionValue is a code list that enumerates the different purposes of a LandUse.
Stereotype:	«CodeList»

LandUseUsageValue

Definition:	LandUseUsageValue is a code list that enumerates the different uses of a LandUse.
Stereotype:	«CodeList»

9.8.6. Enumerations

none

9.9. PointCloud

Description: The PointCloud module supports representation of CityGML features by a collection of points.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.9.1. Classes

PointCloud

Definition: A PointCloud is an unordered collection of points that is a sampling of the geometry of a space or space boundary.

Subclass of: [AbstractPointCloud](#)

Stereotype: «FeatureType»

Constraint: `inlineOrExternalPointCloud (OCL): inv: (points → notEmpty() and mimeType → isEmpty() and pointFile → isEmpty() and pointFileSrsName → isEmpty()) xor (points → isEmpty() and mimeType → notEmpty() and pointFile → notEmpty())`

Role name	Target class and multiplicity	Definition
points	GM_MultiPoint [0..1]	Relates to the 3D MultiPoint geometry of the PointCloud stored inline with the city model.

Attribute	Value type and multiplicity	Definition
mimeType	MimeTypeValue [0..1]	Specifies the MIME type of the external point cloud file.
pointFile	URI [0..1]	Specifies the URI that points to the external point cloud file.
pointFileSrsName	CharacterString [0..1]	Indicates the coordinate reference system used by the external point cloud file.
adeOfPointCloud	ADEOfPointCloud [0..*]	Augments the PointCloud with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.9.2. Data Types

ADEOfPointCloud

Definition:	ADEOfPointCloud acts as a hook to define properties within an ADE that are to be added to a PointCloud.
Subclass of:	None
Stereotype:	«DataType»

9.9.3. Basic Types

none

9.9.4. Unions

none

9.9.5. Code Lists

none

9.9.6. Enumerations

none

9.10. Relief

Description:	The Relief module supports representation of the terrain. CityGML supports terrain representations at different levels of detail, reflecting different accuracies or resolutions. Terrain may be specified as a regular raster or grid, as a TIN, by break lines, and/or by mass points.
Parent Package:	CityGML
Stereotype:	«ApplicationSchema»

9.10.1. Classes

AbstractReliefComponent

Definition:	An AbstractReliefComponent represents an element of the terrain surface - either a TIN, a raster or grid, mass points or break lines.
Subclass of:	AbstractSpaceBoundary
Stereotype:	«FeatureType»
Constraint:	<code>polygonGeometry (OCL): inv: extent.patch → size()=1 and extent.patch → forAll(oclIsKindOf(GM_Polygon))</code>

Role name	Target class and multiplicity	Definition
extent	GM_Surface [0..1]	Indicates the geometrical extent of the terrain component. The geometrical extent is provided as a 2D Surface geometry.
Attribute	Value type and multiplicity	Definition
lod	IntegerBetween0and3 [1..1]	Indicates the Level of Detail of the terrain component.
adeOfAbstractReliefComponent	ADEOfAbstractReliefComponent [0..*]	Augments AbstractReliefComponent with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

BreaklineRelief

Definition:	A BreaklineRelief represents a terrain component with 3D lines. These lines denote break lines or ridge/valley lines.	
Subclass of:	AbstractReliefComponent	
Stereotype:	«FeatureType»	
Role name	Target class and multiplicity	Definition
breaklines	GM_MultiCurve [0..1]	Relates to the 3D MultiCurve geometry of the MassPointRelief. This association role is used to represent break lines.
ridgeOrValleyLines	GM_MultiCurve [0..1]	Relates to the 3D MultiCurve geometry of the MassPointRelief. This association role is used to represent ridge or valley lines.

Attribute	Value type and multiplicity	Definition
adeOfBreaklineRelief	ADEOfBreaklineRelief [0..*]	Augments the BreaklineRelief with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

MassPointRelief

Definition:	A MassPointRelief represents a terrain component as a collection of 3D points.										
Subclass of:	AbstractReliefComponent										
Stereotype:	«FeatureType»										
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>pointCloud</td><td>AbstractPointCloud [0..1]</td><td>Relates to the 3D PointCloud of the MassPointRelief.</td></tr> <tr> <td>reliefPoints</td><td>GM_MultiPoint [0..1]</td><td>Relates to the 3D MultiPoint geometry of the MassPointRelief.</td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	pointCloud	AbstractPointCloud [0..1]	Relates to the 3D PointCloud of the MassPointRelief.	reliefPoints	GM_MultiPoint [0..1]	Relates to the 3D MultiPoint geometry of the MassPointRelief.
Role name	Target class and multiplicity	Definition									
pointCloud	AbstractPointCloud [0..1]	Relates to the 3D PointCloud of the MassPointRelief.									
reliefPoints	GM_MultiPoint [0..1]	Relates to the 3D MultiPoint geometry of the MassPointRelief.									
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>adeOfMassPoi ntRelief</td><td>ADEOfMassPointRe lief [0..*]</td><td>Augments the MassPointRelief with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	adeOfMassPoi ntRelief	ADEOfMassPointRe lief [0..*]	Augments the MassPointRelief with properties defined in an ADE.			
Attribute	Value type and multiplicity	Definition									
adeOfMassPoi ntRelief	ADEOfMassPointRe lief [0..*]	Augments the MassPointRelief with properties defined in an ADE.									
<p>Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».</p>											

RasterRelief								
Definition:	A RasterRelief represents a terrain component as a regular raster or grid.							
Subclass of:	AbstractReliefComponent							
Stereotype:	«FeatureType»							
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>grid</td><td>CV_DiscreteGridPoi ntCoverage [1]</td><td>Relates to the DiscreteGridPointCoverage of the RasterRelief.</td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	grid	CV_DiscreteGridPoi ntCoverage [1]	Relates to the DiscreteGridPointCoverage of the RasterRelief.
Role name	Target class and multiplicity	Definition						
grid	CV_DiscreteGridPoi ntCoverage [1]	Relates to the DiscreteGridPointCoverage of the RasterRelief.						
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>adeOfRasterR elief</td><td>ADEOfRasterRelief [0..*]</td><td>Augments the RasterRelief with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	adeOfRasterR elief	ADEOfRasterRelief [0..*]	Augments the RasterRelief with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition						
adeOfRasterR elief	ADEOfRasterRelief [0..*]	Augments the RasterRelief with properties defined in an ADE.						
<p>Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».</p>								

ReliefFeature

Definition:	A ReliefFeature is a collection of terrain components representing the Earth's surface, also known as the Digital Terrain Model.										
Subclass of:	AbstractSpaceBoundary										
Stereotype:	«TopLevelFeatureType»										
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>reliefComponent</td><td>AbstractReliefComponent [1..*]</td><td>Relates to the terrain components that are part of the ReliefFeature.</td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	reliefComponent	AbstractReliefComponent [1..*]	Relates to the terrain components that are part of the ReliefFeature.			
Role name	Target class and multiplicity	Definition									
reliefComponent	AbstractReliefComponent [1..*]	Relates to the terrain components that are part of the ReliefFeature.									
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>lod</td><td>IntegerBetween0andd3 [1..1]</td><td>Indicates the Level of Detail of the ReliefFeature.</td></tr> <tr> <td>adeOfReliefFeature</td><td>ADEOfReliefFeature [0..*]</td><td>Augments the ReliefFeature with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	lod	IntegerBetween0andd3 [1..1]	Indicates the Level of Detail of the ReliefFeature.	adeOfReliefFeature	ADEOfReliefFeature [0..*]	Augments the ReliefFeature with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition									
lod	IntegerBetween0andd3 [1..1]	Indicates the Level of Detail of the ReliefFeature.									
adeOfReliefFeature	ADEOfReliefFeature [0..*]	Augments the ReliefFeature with properties defined in an ADE.									
<p>Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».</p>											

TINRelief

Definition:	A TINRelief represents a terrain component as a triangulated irregular network.							
Subclass of:	AbstractReliefComponent							
Stereotype:	«FeatureType»							
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>tin</td><td>GM_TriangulatedSurface [1]</td><td>Relates to the triangulated surface of the TINRelief.</td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	tin	GM_TriangulatedSurface [1]	Relates to the triangulated surface of the TINRelief.
Role name	Target class and multiplicity	Definition						
tin	GM_TriangulatedSurface [1]	Relates to the triangulated surface of the TINRelief.						
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>adeOfTINRelief</td><td>ADEOfTINRelief [0..*]</td><td>Augments the TINRelief with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	adeOfTINRelief	ADEOfTINRelief [0..*]	Augments the TINRelief with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition						
adeOfTINRelief	ADEOfTINRelief [0..*]	Augments the TINRelief with properties defined in an ADE.						
<p>Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».</p>								

9.10.2. Data Types

ADEOfAbstractReliefComponent

Definition: ADEOfAbstractReliefComponent acts as a hook to define properties within an ADE that are to be added to AbstractReliefComponent.

Subclass of: None

Stereotype: «DataType»

ADEOfBreaklineRelief

Definition: ADEOfBreaklineRelief acts as a hook to define properties within an ADE that are to be added to a BreaklineRelief.

Subclass of: None

Stereotype: «DataType»

ADEOfMassPointRelief

Definition: ADEOfMassPointRelief acts as a hook to define properties within an ADE that are to be added to a MassPointRelief.

Subclass of: None

Stereotype: «DataType»

ADEOfRasterRelief

Definition: ADEOfRasterRelief acts as a hook to define properties within an ADE that are to be added to a RasterRelief.

Subclass of: None

Stereotype: «DataType»

ADEOfReliefFeature

Definition: ADEOfReliefFeature acts as a hook to define properties within an ADE that are to be added to a ReliefFeature.

Subclass of: None

Stereotype: «DataType»

ADEOfTINRelief

Definition: ADEOfTINRelief acts as a hook to define properties within an ADE that are to be added to a TINRelief.

Subclass of: None

Stereotype: «DataType»

9.10.3. Basic Types

none

9.10.4. Unions

none

9.10.5. Code Lists

none

9.10.6. Enumerations

none

9.11. Transportation

Description: The Transportation module supports representation of the transportation infrastructure. Transportation features include roads, tracks, waterways, railways, and squares. Transportation features may be represented as a network and/or as a collection of spaces or surface elements embedded in a three-dimensional space.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.11.1. Classes

AbstractTransportationSpace

Definition: AbstractTransportationSpace is the abstract superclass of transportation objects such as Roads, Tracks, Railways, Waterways or Squares.

Subclass of: [AbstractUnoccupiedSpace](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
marking	Marking [*]	Relates to the markings that are part of the transportation space.
trafficSpace	TrafficSpace [*]	Relates to the traffic spaces that are part of the transportation space.
auxiliaryTrafficSpace	AuxiliaryTrafficSpace [*]	Relates to the auxiliary traffic spaces that are part of the transportation space.
hole	Hole [*]	Relates to the holes that are part of the transportation space.
Attribute	Value type and multiplicity	Definition
trafficDirection	TrafficDirectionValue [0..1]	Indicates the direction of traffic flow relative to the corresponding linear geometry representation.
occupancy	Occupancy [0..*]	Provides information on the residency of persons, vehicles, or other moving features in the transportation space.
ADEOfAbstractTransportationSpace	ADEOfAbstractTransportationSpace [0..*]	Augments AbstractTransportationSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AuxiliaryTrafficArea

Definition:	An AuxiliaryTrafficArea is the ground surface of an AuxiliaryTrafficSpace.
Subclass of:	AbstractThematicSurface
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
class	AuxiliaryTrafficAre aClassValue [0..1]	Indicates the specific type of the AuxiliaryTrafficArea.
function	AuxiliaryTrafficAre aFunctionValue [0..*]	Specifies the intended purposes of the AuxiliaryTrafficArea.
usage	AuxiliaryTrafficAre aUsageValue [0..*]	Specifies the actual uses of the AuxiliaryTrafficArea.
surfaceMaterial	SurfaceMaterialVal [0..1]	Specifies the type of pavement of the AuxiliaryTrafficArea.
adeOfAuxiliaryTrafficArea	ADEOfAuxiliaryTrafficArea [0..*]	Augments the AuxiliaryTrafficArea with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AuxiliaryTrafficSpace

Definition: An AuxiliaryTrafficSpace is a space within the transportation space not intended for traffic purposes.

Subclass of: [AbstractUnoccupiedSpace](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
boundary	AuxiliaryTrafficAre a [*]	Relates to the auxiliary traffic areas that bound the AuxiliaryTrafficSpace. This relation is inherited from the Core module.

Attribute	Value type and multiplicity	Definition
class	AuxiliaryTrafficSpace <code>ceClassValue [0..1]</code>	Indicates the specific type of the AuxiliaryTrafficSpace.
function	AuxiliaryTrafficSpace <code>ceFunctionValue [0..*]</code>	Specifies the intended purposes of the AuxiliaryTrafficSpace.
usage	AuxiliaryTrafficSpace <code>ceUsageValue [0..*]</code>	Specifies the actual uses of the AuxiliaryTrafficSpace.
granularity	GranularityValue <code>[1..1]</code>	Defines whether auxiliary traffic spaces are represented by individual ways or by individual lanes, depending on the desired level of spatial and semantic decomposition.
adeOfAuxiliar yTrafficSpace	ADEOfAuxiliaryTrafficSpace <code>[0..*]</code>	Augments the AuxiliaryTrafficSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

ClearanceSpace

- Definition: A ClearanceSpace represents the actual free space above a TrafficArea within which a mobile object can move without contacting an obstruction.
- Subclass of: [AbstractUnoccupiedSpace](#)
- Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
class	ClearanceSpaceClass <code>ceValue [0..*]</code>	Indicates the specific type of the ClearanceSpace.
adeOfClearan ceSpace	ADEOfClearanceSpace <code>[0..*]</code>	Augments the ClearanceSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Hole

Definition:	A Hole is an opening in the surface of a Road, Track or Square such as road damages, manholes or drains. Holes can span multiple transportation objects.	
Subclass of:	AbstractUnoccupiedSpace	
Stereotype:	«FeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the Hole. This relation is inherited from the Core module.
<hr/>		
Attribute	Value type and multiplicity	Definition
class	HoleClassValue [0..1]	Indicates the specific type of the Hole.
adeOfHole	ADEOfHole [0..*]	Augments the Hole with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

HoleSurface		
<hr/>		
Definition:	A HoleSurface is a representation of the ground surface of a hole.	
Subclass of:	AbstractThematicSurface	
Stereotype:	«FeatureType»	
<hr/>		
Attribute	Value type and multiplicity	Definition
adeOfHoleSurface	ADEOfHoleSurface [0..*]	Augments the HoleSurface with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

Intersection		
<hr/>		
Definition:	An Intersection is a transportation space that is a shared segment of multiple Road, Track, Railway, or Waterway objects (e.g. a crossing of two roads or a level crossing of a road and a railway).	
Subclass of:	AbstractTransportationSpace	
Stereotype:	«FeatureType»	
<hr/>		

Attribute	Value type and multiplicity	Definition
class	IntersectionClassValue [0..1]	Indicates the specific type of the Intersection.
adeOfIntersection	ADEOfIntersection [0..*]	Augments the Intersection with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

Marking

Definition:	A Marking is a visible pattern on a transportation area relevant to the structuring or restriction of traffic. Examples are road markings and markings related to railway or waterway traffic.
Subclass of:	AbstractThematicSurface
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
class	MarkingClassValue [0..1]	Indicates the specific type of the Marking.
adeOfMarking	ADEOfMarking [0..*]	Augments the Marking with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

Railway

Definition:	A Railway is a transportation space used by wheeled vehicles on rails.
Subclass of:	AbstractTransportationSpace
Stereotype:	«TopLevelFeatureType»

Role name	Target class and multiplicity	Definition
intersection	Intersection [*]	Relates to the intersections that are part of the Railway.
section	Section [*]	Relates to the sections that are part of the Railway.

Attribute	Value type and multiplicity	Definition
class	RailwayClassValue [0..1]	Indicates the specific type of the Railway.
function	RailwayFunctionValue [0..*]	Specifies the intended purposes of the Railway.
usage	RailwayUsageValue [0..*]	Specifies the actual uses of the Railway.
adeOfRailway	ADEOfRailway [0..*]	Augments the Railway with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Road

Definition:	A Road is a transportation space used by vehicles, bicycles and/or pedestrians.
Subclass of:	AbstractTransportationSpace
Stereotype:	«TopLevelFeatureType»

Role name	Target class and multiplicity	Definition
intersection	Intersection [*]	Relates to the intersections that are part of the Road.
section	Section [*]	Relates to the sections that are part of the Road.

Attribute	Value type and multiplicity	Definition
class	RoadClassValue [0..1]	Indicates the specific type of the Road.
function	RoadFunctionValue [0..*]	Specifies the intended purposes of the Road.
usage	RoadUsageValue [0..*]	Specifies the actual uses of the Road.
adeOfRoad	ADEOfRoad [0..*]	Augments the Road with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Section

Definition:	A Section is a transportation space that is a segment of a Road, Railway, Track, or Waterway.										
Subclass of:	AbstractTransportationSpace										
Stereotype:	«FeatureType»										
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>class</td><td>SectionClassValue [0..1]</td><td>Indicates the specific type of the Section.</td></tr> <tr> <td>adeOfSection</td><td>ADEOfSection [0..*]</td><td>Augments the Section with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	class	SectionClassValue [0..1]	Indicates the specific type of the Section.	adeOfSection	ADEOfSection [0..*]	Augments the Section with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition									
class	SectionClassValue [0..1]	Indicates the specific type of the Section.									
adeOfSection	ADEOfSection [0..*]	Augments the Section with properties defined in an ADE.									
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».											

Square

Definition:	A Square is a transportation space for unrestricted movement for vehicles, bicycles and/or pedestrians. This includes plazas as well as large sealed surfaces such as parking lots.																
Subclass of:	AbstractTransportationSpace																
Stereotype:	«TopLevelFeatureType»																
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>class</td><td>SquareClassValue [0..1]</td><td>Indicates the specific type of the Square.</td></tr> <tr> <td>function</td><td>SquareFunctionVal [0..*]</td><td>Specifies the intended purposes of the Square.</td></tr> <tr> <td>usage</td><td>SquareUsageValue [0..*]</td><td>Specifies the actual uses of the Square.</td></tr> <tr> <td>adeOfSquare</td><td>ADEOfSquare [0..*]</td><td>Augments the Square with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	class	SquareClassValue [0..1]	Indicates the specific type of the Square.	function	SquareFunctionVal [0..*]	Specifies the intended purposes of the Square.	usage	SquareUsageValue [0..*]	Specifies the actual uses of the Square.	adeOfSquare	ADEOfSquare [0..*]	Augments the Square with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition															
class	SquareClassValue [0..1]	Indicates the specific type of the Square.															
function	SquareFunctionVal [0..*]	Specifies the intended purposes of the Square.															
usage	SquareUsageValue [0..*]	Specifies the actual uses of the Square.															
adeOfSquare	ADEOfSquare [0..*]	Augments the Square with properties defined in an ADE.															
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».																	

Track

Definition:	A Track is a small path mainly used by pedestrians. Tracks can be segmented into Sections and Intersections.	
Subclass of:	AbstractTransportationSpace	
Stereotype:	«TopLevelFeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
section	Section [*]	Relates to the sections that are part of the Track.
intersection	Intersection [*]	Relates to the intersections that are part of the Track.
<hr/>		
Attribute	Value type and multiplicity	Definition
class	TrackClassValue [0..1]	Indicates the specific type of the Track.
function	TrackFunctionValue e [0..*]	Specifies the intended purposes of the Track.
usage	TrackUsageValue [0..*]	Specifies the actual uses of the Track.
adeOfTrack	ADEOfTrack [0..*]	Augments the Track with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

TrafficArea

Definition:	A TrafficArea is the ground surface of a TrafficSpace. Traffic areas are the surfaces upon which traffic actually takes place.
Subclass of:	AbstractThematicSurface
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
class	TrafficAreaClassVal [0..1]	Indicates the specific type of the TrafficArea.
function	TrafficAreaFunction [0..*]	Specifies the intended purposes of the TrafficArea.
usage	TrafficAreaUsageValue [0..*]	Specifies the actual uses of the TrafficArea.
surfaceMaterial	SurfaceMaterialValue [0..1]	Specifies the type of pavement of the TrafficArea.
adeOfTrafficArea	ADEOfTrafficArea [0..*]	Augments the TrafficArea with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TrafficSpace

Role name	Target class and multiplicity	Definition
successor	TrafficSpace [*]	Indicates the successor(s) of the TrafficSpace.
clearanceSpace	ClearanceSpace [*]	Relates to the clearance spaces that are part of the TrafficSpace.
predecessor	TrafficSpace [*]	Indicates the predecessor(s) of the TrafficSpace.
boundary	TrafficArea [*]	Relates to the traffic areas that bound the TrafficSpace. This relation is inherited from the Core module.

Attribute	Value type and multiplicity	Definition
class	TrafficSpaceClassValue [0..1]	Indicates the specific type of the TrafficSpace.
function	TrafficSpaceFunctionValue [0..*]	Specifies the intended purposes of the TrafficSpace.
usage	TrafficSpaceUsageValue [0..*]	Specifies the actual uses of the TrafficSpace.
granularity	GranularityValue [1..1]	Defines whether traffic spaces are represented by individual ways or by individual lanes, depending on the desired level of spatial and semantic decomposition.
trafficDirection	TrafficDirectionValue [0..1]	Indicates the direction of traffic flow relative to the corresponding linear geometry representation.
occupancy	Occupancy [0..*]	Provides information on the residency of persons, vehicles, or other moving features in the TrafficSpace.
adeOfTrafficSpace	ADEOfTrafficSpace [0..*]	Augments the TrafficSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Waterway

Role name	Target class and multiplicity	Definition
intersection	Intersection [*]	Relates to the intersections that are part of the Waterway.
section	Section [*]	Relates to the sections that are part of the Waterway.

Attribute	Value type and multiplicity	Definition
class	WaterwayClassVal [0..1]	Indicates the specific type of the Waterway.
function	WaterwayFunction Value [0..*]	Specifies the intended purposes of the Waterway.
usage	WaterwayUsageVal [0..*]	Specifies the actual uses of the Waterway.
adeOfWaterway	ADEOfWaterway [0..*]	Augments the Waterway with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.11.2. Data Types

ADEOfAbstractTransportationSpace

- Definition: ADEOfAbstractTransportationSpace acts as a hook to define properties within an ADE that are to be added to AbstractTransportationSpace.
- Subclass of: None
- Stereotype: «DataType»

ADEOfAuxiliaryTrafficArea

- Definition: ADEOfAuxiliaryTrafficArea acts as a hook to define properties within an ADE that are to be added to an AuxiliaryTrafficArea.
- Subclass of: None
- Stereotype: «DataType»

ADEOfAuxiliaryTrafficSpace

- Definition: ADEOfAuxiliaryTrafficSpace acts as a hook to define properties within an ADE that are to be added to an AuxiliaryTrafficSpace.
- Subclass of: None
- Stereotype: «DataType»

ADEOfClearanceSpace

Definition: ADEOfClearanceSpace acts as a hook to define properties within an ADE that are to be added to a ClearanceSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfHole

Definition: ADEOfHole acts as a hook to define properties within an ADE that are to be added to a Hole.

Subclass of: None

Stereotype: «DataType»

ADEOfHoleSurface

Definition: ADEOfHoleSurface acts as a hook to define properties within an ADE that are to be added to a HoleSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfIntersection

Definition: ADEOfIntersection acts as a hook to define properties within an ADE that are to be added to an Intersection.

Subclass of: None

Stereotype: «DataType»

ADEOfMarking

Definition: ADEOfMarking acts as a hook to define properties within an ADE that are to be added to a Marking.

Subclass of: None

Stereotype: «DataType»

ADEOfRailway

Definition: ADEOfRailway acts as a hook to define properties within an ADE that are to be added to a Railway.

Subclass of: None

Stereotype: «DataType»

ADEOfRoad

Definition: ADEOfRoad acts as a hook to define properties within an ADE that are to be added to a Road.

Subclass of: None

Stereotype: «DataType»

ADEOfSection

Definition: ADEOfSection acts as a hook to define properties within an ADE that are to be added to a Section.

Subclass of: None

Stereotype: «DataType»

ADEOfSquare

Definition: ADEOfSquare acts as a hook to define properties within an ADE that are to be added to a Square.

Subclass of: None

Stereotype: «DataType»

ADEOfTrack

Definition: ADEOfTrack acts as a hook to define properties within an ADE that are to be added to a Track.

Subclass of: None

Stereotype: «DataType»

ADEOfTrafficArea

Definition: ADEOfTrafficArea acts as a hook to define properties within an ADE that are to be added to a TrafficArea.

Subclass of: None

Stereotype: «DataType»

ADEOfTrafficSpace

Definition: ADEOfTrafficSpace acts as a hook to define properties within an ADE that are to be added to a TrafficSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfWaterway

Definition: ADEOfWaterway acts as a hook to define properties within an ADE that are to be added to a Waterway.

Subclass of: None

Stereotype: «DataType»

9.11.3. Basic Types

none

9.11.4. Unions

none

9.11.5. Code Lists

AuxiliaryTrafficAreaClassValue

Definition: AuxiliaryTrafficAreaClassValue is a code list used to further classify an AuxiliaryTrafficArea.

Stereotype: «CodeList»

AuxiliaryTrafficAreaFunctionValue

Definition:	AuxiliaryTrafficAreaFunctionValue is a code list that enumerates the different purposes of an AuxiliaryTrafficArea.
Stereotype:	«CodeList»

AuxiliaryTrafficAreaUsageValue

Definition:	AuxiliaryTrafficAreaUsageValue is a code list that enumerates the different uses of an AuxiliaryTrafficArea.
Stereotype:	«CodeList»

AuxiliaryTrafficSpaceClassValue

Definition:	AuxiliaryTrafficSpaceClassValue is a code list used to further classify an AuxiliaryTrafficSpace.
Stereotype:	«CodeList»

AuxiliaryTrafficSpaceFunctionValue

Definition:	AuxiliaryTrafficSpaceFunctionValue is a code list that enumerates the different purposes of an AuxiliaryTrafficSpace.
Stereotype:	«CodeList»

AuxiliaryTrafficSpaceUsageValue

Definition:	AuxiliaryTrafficSpaceUsageValue is a code list that enumerates the different uses of an AuxiliaryTrafficSpace.
Stereotype:	«CodeList»

ClearanceSpaceClassValue

Definition:	ClearanceSpaceClassValue is a code list used to further classify a ClearanceSpace.
Stereotype:	«CodeList»

HoleClassValue

Definition: HoleClassValue is a code list used to further classify a Hole.

Stereotype: «CodeList»

IntersectionClassValue

Definition: IntersectionClassValue is a code list used to further classify an Intersection.

Stereotype: «CodeList»

MarkingClassValue

Definition: MarkingClassValue is a code list used to further classify a Marking.

Stereotype: «CodeList»

RailwayClassValue

Definition: RailwayClassValue is a code list used to further classify a Railway.

Stereotype: «CodeList»

RailwayFunctionValue

Definition: RailwayFunctionValue is a code list that enumerates the different purposes of a Railway.

Stereotype: «CodeList»

RailwayUsageValue

Definition: RailwayUsageValue is a code list that enumerates the different uses of a Railway.

Stereotype: «CodeList»

RoadClassValue

Definition: RoadClassValue is a code list used to further classify a Road.

Stereotype: «CodeList»

RoadFunctionValue

Definition: RoadFunctionValue is a code list that enumerates the different purposes of a Road.

Stereotype: «CodeList»

RoadUsageValue

Definition: RoadUsageValue is a code list that enumerates the different uses of a Road.

Stereotype: «CodeList»

SectionClassValue

Definition: SectionClassValue is a code list used to further classify a Section.

Stereotype: «CodeList»

SquareClassValue

Definition: SquareClassValue is a code list used to further classify a Square.

Stereotype: «CodeList»

SquareFunctionValue

Definition: SquareFunctionValue is a code list that enumerates the different purposes of a Square.

Stereotype: «CodeList»

SquareUsageValue

Definition: SquareUsageValue is a code list that enumerates the different uses of a Square.

Stereotype: «CodeList»

SurfaceMaterialValue

Definition: SurfaceMaterialValue is a code list that enumerates the different surface materials.

Stereotype: «CodeList»

TrackClassValue

Definition: TrackClassValue is a code list used to further classify a Track.

Stereotype: «CodeList»

TrackFunctionValue

Definition: TrackFunctionValue is a code list that enumerates the different purposes of a Track.

Stereotype: «CodeList»

TrackUsageValue

Definition: TrackUsageValue is a code list that enumerates the different uses of a Track.

Stereotype: «CodeList»

TrafficAreaClassValue

Definition: TrafficAreaClassValue is a code list used to further classify a TrafficArea.

Stereotype: «CodeList»

TrafficAreaFunctionValue

Definition: TrafficAreaFunctionValue is a code list that enumerates the different purposes of a TrafficArea.

Stereotype: «CodeList»

TrafficAreaUsageValue

Definition: TrafficAreaUsageValue is a code list that enumerates the different uses of a TrafficArea.

Stereotype: «CodeList»

TrafficSpaceClassValue

Definition: TrafficSpaceClassValue is a code list used to further classify a TrafficSpace.

Stereotype: «CodeList»

TrafficSpaceFunctionValue

Definition: TrafficSpaceFunctionValue is a code list that enumerates the different purposes of a TrafficSpace.

Stereotype: «CodeList»

TrafficSpaceUsageValue

Definition: TrafficSpaceUsageValue is a code list that enumerates the different uses of a TrafficSpace.

Stereotype: «CodeList»

WaterwayClassValue

Definition: WaterwayClassValue is a code list used to further classify a Waterway.

Stereotype: «CodeList»

WaterwayFunctionValue

Definition: WaterwayFunctionValue is a code list that enumerates the different purposes of a Waterway.

Stereotype: «CodeList»

WaterwayUsageValue

Definition:	WaterwayUsageValue is a code list that enumerates the different uses of a Waterway.
Stereotype:	«CodeList»

9.11.6. Enumerations

GranularityValue

Definition: GranularityValue enumerates the different levels of granularity in which transportation objects are represented.

Stereotype: <<Enumeration>>

Literal value	Definition
lane	Indicates that the individual lanes of the transportation object are represented.
way	Indicates that the individual (carriage)ways of the transportation object are represented.

TrafficDirectionValue

Definition: TrafficDirectionValue enumerates the allowed directions of travel of a mobile object.

Stereotype: <<Enumeration>>

Literal value	Definition
forwards	Indicates that traffic flows in the direction of the corresponding linear geometry.
backwards	Indicates that traffic flows in the opposite direction of the corresponding linear geometry.
both	Indicates that traffic flows in both directions.

9.12. Vegetation

Description: The Vegetation module supports representation of vegetation objects with vegetation-specific thematic classes. CityGML's vegetation model distinguishes between solitary vegetation objects like trees, and vegetation areas which represent biotopes like forests or other plant communities.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.12.1. Classes

AbstractVegetationObject

Definition: AbstractVegetationObject is the abstract superclass for all kinds of vegetation objects.

Subclass of: [AbstractOccupiedSpace](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

adeOfAbstractVegetationObj	ADEOfAbstractVeg [0..*]	Augments AbstractVegetationObject with properties defined in an ADE.
----------------------------	---	--

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

PlantCover

Definition: A PlantCover represents a space covered by vegetation.

Subclass of: [AbstractVegetationObject](#)

Stereotype: «TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

class	PlantCoverClassVal [0..1]	Indicates the specific type of the PlantCover.
-------	---	--

function	PlantCoverFunction [0..*]	Specifies the intended purposes of the PlantCover.
----------	---	--

usage	PlantCoverUsageValue [0..*]	Specifies the actual uses of the PlantCover.
-------	---	--

averageHeight	Length [0..1]	Specifies the average height of the PlantCover.
---------------	-------------------------------	---

minHeight	Length [0..1]	Specifies the minimum height of the PlantCover.
-----------	-------------------------------	---

maxHeight	Length [0..1]	Specifies the maximum height of the PlantCover.
-----------	-------------------------------	---

adeOfPlantCover	ADEOfPlantCover [0..*]	Augments the PlantCover with properties defined in an ADE.
-----------------	--	--

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

SolitaryVegetationObject

Definition: A SolitaryVegetationObject represents individual vegetation objects, e.g. trees or bushes.

Subclass of: [AbstractVegetationObject](#)

Stereotype: «TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
class	SolitaryVegetationObjectClassValue [0..1]	Indicates the specific type of the SolitaryVegetationObject.
function	SolitaryVegetationObjectFunctionValue [0..*]	Specifies the intended purposes of the SolitaryVegetationObject.
usage	SolitaryVegetationObjectUsageValue [0..*]	Specifies the actual uses of the SolitaryVegetationObject.
species	SpeciesValue [0..1]	Indicates the botanical name of the SolitaryVegetationObject.
height	Length [0..1]	Distance between the highest point of the vegetation object and the lowest point of the terrain at the bottom of the object.
trunkDiameter	Length [0..1]	Specifies the diameter of the SolitaryCityObject's trunk.
crownDiameter	Length [0..1]	Specifies the diameter of the SolitaryCityObject's crown.
rootBallDiameter	Length [0..1]	Specifies the diameter of the SolitaryCityObject's root ball.
maxRootBallDepth	Length [0..1]	Specifies the vertical distance between the lowest point of the SolitaryVegetationObject's root ball and the terrain surface.
adeOfSolitaryVegetationObject	ADEOfSolitaryVegetationObject [0..*]	Augments the SolitaryVegetationObject with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.12.2. Data Types

ADEOfAbstractVegetationObject

Definition: ADEOfAbstractVegetationObject acts as a hook to define properties within an ADE that are to be added to AbstractVegetationObject.

Subclass of: None

Stereotype: «DataType»

ADEOfPlantCover

Definition: ADEOfPlantCover acts as a hook to define properties within an ADE that are to be added to a PlantCover.

Subclass of: None

Stereotype: «DataType»

ADEOfSolitaryVegetationObject

Definition: ADEOfSolitaryVegetationObject acts as a hook to define properties within an ADE that are to be added to a SolitaryVegetationObject.

Subclass of: None

Stereotype: «DataType»

9.12.3. Basic Types

none

9.12.4. Unions

none

9.12.5. Code Lists

PlantCoverClassValue

Definition: PlantCoverClassValue is a code list used to further classify a PlantCover.

Stereotype: «CodeList»

PlantCoverFunctionValue

Definition:	PlantCoverFunctionValue is a code list that enumerates the different purposes of a PlantCover.
Stereotype:	«CodeList»

PlantCoverUsageValue

Definition:	PlantCoverUsageValue is a code list that enumerates the different uses of a PlantCover.
Stereotype:	«CodeList»

SolitaryVegetationObjectClassValue

Definition:	SolitaryVegetationObjectClassValue is a code list used to further classify a SolitaryVegetationObject.
Stereotype:	«CodeList»

SolitaryVegetationObjectFunctionValue

Definition:	SolitaryVegetationObjectFunctionValue is a code list that enumerates the different purposes of a SolitaryVegetationObject.
Stereotype:	«CodeList»

SolitaryVegetationObjectUsageValue

Definition:	SolitaryVegetationObjectUsageValue is a code list that enumerates the different uses of a SolitaryVegetationObject.
Stereotype:	«CodeList»

SpeciesValue

Definition:	A SpeciesValue is a code list that enumerates the species of a SolitaryVegetationObject.
Stereotype:	«CodeList»

9.12.6. Enumerations

none

9.13. Versioning

Description: The Versioning module supports representation of multiple versions of CityGML features within a single CityGML model. In addition, also the version transitions and transactions that lead to the different versions can be represented.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.13.1. Classes

Version		
Role name	Target class and multiplicity	Definition
versionMember «Version»	AbstractFeatureWithLifespan [*]	Relates to all city objects that are part of the city model version.
Attribute	Value type and multiplicity	Definition
tag	CharacterString [0..*]	Allows for adding keywords to the city model version.
adeOfVersion	ADEOfVersion [0..*]	Augments the Version with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

VersionTransition

Definition:	VersionTransition describes the change of the state of a city model from one version to another. Version transitions can have names, a description and can be further qualified by a type and a reason.	
Subclass of:	AbstractVersionTransition	
Stereotype:	«FeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
from	Version [0..1]	Relates to the predecessor version of the VersionTransition.
transaction	Transaction [*]	Relates to all transactions that have been applied as part of the VersionTransition.
to	Version [0..1]	Relates to the successor version of the VersionTransition.
<hr/>		
Attribute	Value type and multiplicity	Definition
reason	CharacterString [0..1]	Specifies why the VersionTransition has been carried out.
clonePrecedes	Boolean [1..1]	Indicates whether the set of city object instances belonging to the successor version of the city model is either explicitly enumerated within the successor version object (attribute clonePredecessor=false), or has to be derived from the modifications of the city model provided as a list of transactions on the city object versions contained in the predecessor version (attribute clonePredecessor=true).
sor		
type	TransitionTypeVal [0..1]	Indicates the specific type of the VersionTransition.
adeOfVersion	ADEOfVersionTransition [0..*]	Augments the VersionTransition with properties defined in an ADE.
Transition		
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

9.13.2. Data Types

ADEOfVersion	
Definition:	ADEOfVersion acts as a hook to define properties within an ADE that are to be added to a Version.
Subclass of:	None
Stereotype:	«DataType»

ADEOfVersionTransition

Definition: ADEOfVersionTransition acts as a hook to define properties within an ADE that are to be added to a VersionTransition.

Subclass of: None

Stereotype: «DataType»

Transaction

Definition: Transaction represents a modification of the city model by the creation, termination, or replacement of a specific city object. While the creation of a city object also marks its first object version, the termination marks the end of existence of a real world object and, hence, also terminates the final version of a city object. The replacement of a city object means that a specific version of it is replaced by a new version.

Subclass of: None

Stereotype: «DataType»

Role name	Target class and multiplicity	Definition
newFeature «Version»	AbstractFeatureWithLifespan [0..1]	Relates to the version of the city object subsequent to the Transaction.
oldFeature «Version»	AbstractFeatureWithLifespan [0..1]	Relates to the version of the city object prior to the Transaction.

Attribute	Value type and multiplicity	Definition
type	TransactionTypeValue [1..1]	Indicates the specific type of the Transaction.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.13.3. Basic Types

none

9.13.4. Unions

none

9.13.5. Code Lists

none

9.13.6. Enumerations

TransactionTypeValue

Definition: TransactionTypeValue enumerates the three possible types of transactions: insert, delete, or replace.

Stereotype: <>Enumeration>>

Literal value	Definition
insert	Indicates that the feature referenced from the Transaction via the "newFeature" association has been newly created; the association "oldFeature" is empty in this case.
delete	Indicates that the feature referenced from the Transaction via the "oldFeature" association ceases to exist; the association "newFeature" is empty in this case.
replace	Indicates that the feature referenced from the Transaction via the "oldFeature" association has been replaced by the feature referenced via the "newFeature" association.

TransitionTypeValue

Definition: TransitionTypeValue enumerates the different kinds of version transitions. “planned” and “fork” should be used in cases when from one city model version multiple successor versions are being created. “realized” and “merge” should be used when different city model versions are converging into a common successor version.

Stereotype: <>Enumeration>>

Literal value	Definition
planned	Indicates that the successor version of the city model represents a planning state for a possible future of the city.
realized	Indicates that the predecessor version is the chosen one from a number of possible planning versions.
historicalSuccessor	Indicates that the successor version reflects updates on the city model over time (historical timeline). It shall only be used for at most one version transition outgoing from a city model version.
on	
fork	Indicates other reasons to create alternative city model versions, for example, when different parties are updating parts of the city model or to reflect the results of different simulation runs.
merge	Indicates other reasons to converge multiple versions back into a common city model version.

9.14. WaterBody

Description:	The WaterBody module supports representation of the thematic aspects and 3D geometry of rivers, canals, lakes, and basins. It does, however, not inherit any hydrological or other dynamic aspects of fluid flow.
Parent Package:	CityGML
Stereotype:	«ApplicationSchema»

9.14.1. Classes

AbstractWaterBoundarySurface

Definition:	AbstractWaterBoundarySurface is the abstract superclass for all kinds of thematic surfaces bounding a water body.
Subclass of:	AbstractThematicSurface
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstractWaterBoundarySurface	ADEOfAbstractWaterBoundarySurface [0..*]	Augments AbstractWaterBoundarySurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

WaterBody

Definition: A WaterBody represents significant and permanent or semi-permanent accumulations of surface water, usually covering a part of the Earth.

Subclass of: [AbstractOccupiedSpace](#)

Stereotype: «TopLevelFeatureType»

Role name	Target class and multiplicity	Definition
boundary	AbstractWaterBoundarySurface [*]	
Attribute	Value type and multiplicity	Definition
class	WaterBodyClassVal [0..1]	Indicates the specific type of the WaterBody.
function	WaterBodyFunction [0..*]	Specifies the intended purposes of the WaterBody.
usage	WaterBodyUsageValue [0..*]	Specifies the actual uses of the WaterBody.
adeOfWaterBody	ADEOfWaterBody [0..*]	Augments the WaterBody with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

WaterGroundSurface

Definition: A WaterGroundSurface represents the exterior boundary surface of the submerged bottom of a water body.

Subclass of: [AbstractWaterBoundarySurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfWaterGroundSurface	ADEOfWaterGroundSurface [0..*]	Augments the WaterGroundSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

WaterSurface

Definition: A WaterSurface represents the upper exterior interface between a water body and the atmosphere.

Subclass of: [AbstractWaterBoundarySurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
waterLevel	WaterLevelValue [0..1]	Specifies the level of the WaterSurface.
adeOfWaterSurface	ADEOfWaterSurface [0..*]	Augments the WaterSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.14.2. Data Types

ADEOfAbstractWaterBoundarySurface

Definition: ADEOfAbstractWaterBoundarySurface acts as a hook to define properties within an ADE that are to be added to AbstractWaterBoundarySurface.

Subclass of: None

Stereotype: «DataType»

ADEOfWaterBody

Definition: ADEOfWaterBody acts as a hook to define properties within an ADE that are to be added to a WaterBody.

Subclass of: None

Stereotype: «DataType»

ADEOfWaterGroundSurface

Definition: ADEOfWaterGroundSurface acts as a hook to define properties within an ADE that are to be added to a WaterGroundSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfWaterSurface

Definition: ADEOfWaterSurface acts as a hook to define properties within an ADE that are to be added to a WaterSurface.

Subclass of: None

Stereotype: «DataType»

9.14.3. Basic Types

none

9.14.4. Unions

none

9.14.5. Code Lists

WaterBodyClassValue

Definition: WaterBodyClassValue is a code list used to further classify a WaterBody.

Stereotype: «CodeList»

WaterBodyFunctionValue

Definition: WaterBodyFunctionValue is a code list that enumerates the different purposes of a WaterBody.

Stereotype: «CodeList»

WaterBodyUsageValue

Definition: WaterBodyUsageValue is a code list that enumerates the different uses of a WaterBody.

Stereotype: «CodeList»

WaterLevelValue

Definition:	WaterLevelValue is a code list that enumerates the different levels of a water surface.
Stereotype:	«CodeList»

9.14.6. Enumerations

none

9.15. Construction

Description: The Construction module supports representation of key elements of different types of constructions. These key elements include construction surfaces (e.g floor and ceiling), windows and doors, constructive elements (e.g. beams and slabs), installations, and furniture.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.15.1. Classes

AbstractConstruction

Definition: AbstractConstruction is the abstract superclass for objects that are manufactured by humans from construction materials, are connected to earth, and are intended to be permanent. A connection with the ground also exists when the construction rests by its own weight on the ground or is moveable limited on stationary rails or if the construction is intended to be used mainly stationary.

Subclass of: [AbstractOccupiedSpace](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
boundary	AbstractThematicsSurface [*]	Relates to the surfaces that bound the construction. This relation is inherited from the Core module.

Attribute	Value type and multiplicity	Definition
conditionOfConstruction	ConditionOfConstructionValue [0..1]	Indicates the life-cycle status of the construction. [cf. INSPIRE]
dateOfConstruction	Date [0..1]	Indicates the date at which the construction was completed.
dateOfDemolition	Date [0..1]	Indicates the date at which the construction was demolished.
constructionEvent	ConstructionEvent [0..*]	Describes specific events in the life-time of the construction.
elevation	Elevation [0..*]	Specifies qualified elevations of the construction in relation to a well-defined surface which is commonly taken as origin (e.g. geoid or water level). [cf. INSPIRE]
height	Height [0..*]	Specifies qualified heights of the construction above ground or below ground. [cf. INSPIRE]
occupancy	Occupancy [0..*]	Provides qualified information on the residency of persons, animals, or other moveable objects in the construction.
adeOfAbstractConstruction	ADEOfAbstractConstruction [0..*]	Augments AbstractConstruction with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractConstructionSurface

Definition:	AbstractConstructionSurface is the abstract superclass for different kinds of surfaces that bound a construction.
Subclass of:	AbstractThematicSurface
Stereotype:	«FeatureType»

Role name	Target class and multiplicity	Definition
fillingSurface	AbstractFillingSurface [*]	Relates to the surfaces that seal the openings of the construction surface.
Attribute	Value type and multiplicity	Definition
adeOfAbstractConstructionSurface	ADEOfAbstractConstructionSurface [0..*]	Augments AbstractConstructionSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractConstructiveElement

Definition: AbstractConstructiveElement is the abstract superclass for the representation of volumetric elements of a construction. Examples are walls, beams, slabs.

Subclass of: [AbstractOccupiedSpace](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the constructive element. This relation is inherited from the Core module.
filling	AbstractFillingElement [*]	Relates to the elements that fill the opening of the constructive element.
Attribute	Value type and multiplicity	Definition
isStructuralElement	Boolean [0..1]	Indicates whether the constructive element is essential from a structural point of view. A structural element cannot be omitted without collapsing of the construction. Examples are pylons and anchorages of bridges.
adeOfAbstractConstructiveElement	ADEOfAbstractConstructiveElement [0..*]	Augments AbstractConstructiveElement with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractFillingElement

Definition: AbstractFillingElement is the abstract superclass for different kinds of elements that fill the openings of a construction.

Subclass of: [AbstractOccupiedSpace](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstract FillingElement	ADEOfAbstractFilli ngElement [0..*]	Augments AbstractFillingElement with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractFillingSurface

Definition:	AbstractFillingSurface is the abstract superclass for different kinds of surfaces that seal openings filled by filling elements.	
Subclass of:	AbstractThematicSurface	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
adeOfAbstract FillingSurface	ADEOfAbstractFilli ngSurface [0..*]	Augments AbstractFillingSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractFurniture

Definition:	AbstractFurniture is the abstract superclass for the representation of furniture objects of a construction.	
Subclass of:	AbstractOccupiedSpace	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
adeOfAbstract Furniture	ADEOfAbstractFur niture [0..*]	Augments AbstractFurniture with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractInstallation

Definition:	AbstractInstallation is the abstract superclass for the representation of installation objects of a construction.										
Subclass of:	AbstractOccupiedSpace										
Stereotype:	«FeatureType»										
<table border="1"> <thead> <tr> <th>Role name</th> <th>Target class and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>boundary</td> <td>AbstractThematicSurface [*]</td> <td>Relates to the surfaces that bound the installation. This relation is inherited from the Core module.</td> </tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the installation. This relation is inherited from the Core module.			
Role name	Target class and multiplicity	Definition									
boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the installation. This relation is inherited from the Core module.									
<table border="1"> <thead> <tr> <th>Attribute</th> <th>Value type and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>relationToConstruction</td> <td>RelationToConstruction [0..1]</td> <td>Indicates whether the installation is located inside and/or outside of the construction.</td> </tr> <tr> <td>adeOfAbstractInstallation</td> <td>ADEOfAbstractInstallation [0..*]</td> <td>Augments AbstractInstallation with properties defined in an ADE.</td> </tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	relationToConstruction	RelationToConstruction [0..1]	Indicates whether the installation is located inside and/or outside of the construction.	adeOfAbstractInstallation	ADEOfAbstractInstallation [0..*]	Augments AbstractInstallation with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition									
relationToConstruction	RelationToConstruction [0..1]	Indicates whether the installation is located inside and/or outside of the construction.									
adeOfAbstractInstallation	ADEOfAbstractInstallation [0..*]	Augments AbstractInstallation with properties defined in an ADE.									
<p>Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».</p>											

CeilingSurface								
Definition:	A CeilingSurface is a surface that represents the interior ceiling of a construction. An example is the ceiling of a room.							
Subclass of:	AbstractConstructionSurface							
Stereotype:	«FeatureType»							
<table border="1"> <thead> <tr> <th>Attribute</th> <th>Value type and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>adeOfCeilingSurface</td> <td>ADEOfCeilingSurface [0..*]</td> <td>Augments the CeilingSurface with properties defined in an ADE.</td> </tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	adeOfCeilingSurface	ADEOfCeilingSurface [0..*]	Augments the CeilingSurface with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition						
adeOfCeilingSurface	ADEOfCeilingSurface [0..*]	Augments the CeilingSurface with properties defined in an ADE.						
<p>Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».</p>								

Door

Definition:	A Door is a construction for closing an opening intended primarily for access or egress or both. [cf. ISO 6707-1]	
Subclass of:	AbstractFillingElement	
Stereotype:	«FeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
address	Address [*]	Relates to the addresses that are assigned to the Door.
boundary	DoorSurface [*]	Relates to the door surfaces that bound the Door. This relation is inherited from the Core module.
<hr/>		
Attribute	Value type and multiplicity	Definition
class	DoorClassValue [0..1]	Indicates the specific type of the Door.
function	DoorFunctionValue [0..*]	Specifies the intended purposes of the Door.
usage	DoorUsageValue [0..*]	Specifies the actual uses of the Door.
adeOfDoor	ADEOfDoor [0..*]	Augments the Door with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

DoorSurface		
<hr/>		
Definition: A DoorSurface is either a boundary surface of a Door feature or a surface that seals an opening filled by a door.		
Subclass of: AbstractFillingSurface		
Stereotype: «FeatureType»		
<hr/>		
Role name	Target class and multiplicity	Definition
address	Address [*]	Relates to the addresses that are assigned to the DoorSurface.
<hr/>		
Attribute	Value type and multiplicity	Definition
adeOfDoorSur	ADEOfDoorSurface [0..*]	Augments the DoorSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

FloorSurface

Definition: A FloorSurface is surface that represents the interior floor of a construction.
An example is the floor of a room.

Subclass of: [AbstractConstructionSurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfFloorSurface	ADEOfFloorSurface [0..*]	Augments the FloorSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

GroundSurface

Definition: A GroundSurface is a surface that represents the ground plate of a construction. The polygon defining the ground plate is congruent with the footprint of the construction.

Subclass of: [AbstractConstructionSurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfGroundSurface	ADEOfGroundSurface [0..*]	Augments the GroundSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

InteriorWallSurface

Definition: An InteriorWallSurface is a surface that is visible from inside a construction.
An example is the wall of a room.

Subclass of: [AbstractConstructionSurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfInteriorWallSurface	ADEOfInteriorWallSurface [0..*]	Augments the InteriorWallSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

OtherConstruction

Attribute	Value type and multiplicity	Definition
Definition:		An OtherConstruction is a construction that is not covered by any of the other subclasses of AbstractConstruction.
Subclass of:		AbstractConstruction
Stereotype:		«TopLevelFeatureType»
Attribute	Value type and multiplicity	Definition
class	OtherConstruction ClassValue [0..1]	Indicates the specific type of the OtherConstruction.
function	OtherConstruction FunctionValue [0..*]	Specifies the intended purposes of the OtherConstruction.
usage	OtherConstruction UsageValue [0..*]	Specifies the actual uses of the OtherConstruction.
adeOfOtherConstruction	ADEOfOtherConstruction [0..*]	Augments the OtherConstruction with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

OuterCeilingSurface

Definition:	An OuterCeilingSurface is a surface that belongs to the outer building shell with the orientation pointing downwards. An example is the ceiling of a loggia.
Subclass of:	AbstractConstructionSurface
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfOuterCeilingSurface	ADEOfOuterCeiling Surface [0..*]	Augments the OuterCeilingSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

OuterFloorSurface

Definition:	An OuterFloorSurface is a surface that belongs to the outer construction shell with the orientation pointing upwards. An example is the floor of a loggia.	
Subclass of:	AbstractConstructionSurface	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
adeOfOuterFloorSurface	ADEOfOuterFloor Surface [0..*]	Augments the OuterFloorSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

RoofSurface

Definition:	A RoofSurface is a surface that delimits major roof parts of a construction.	
Subclass of:	AbstractConstructionSurface	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
adeOfRoofSurface	ADEOfRoofSurface [0..*]	Augments the RoofSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

WallSurface

Definition: A WallSurface is a surface that is part of the building facade visible from the outside.

Subclass of: [AbstractConstructionSurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfWallSurface	ADEOfWallSurface [0..*]	Augments the WallSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Window

Definition: A Window is a construction for closing an opening in a wall or roof, primarily intended to admit light and/or provide ventilation. [cf. ISO 6707-1]

Subclass of: [AbstractFillingElement](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
boundary	WindowSurface [*]	Relates to the window surfaces that bound the Window. This relation is inherited from the Core module.

Attribute	Value type and multiplicity	Definition
class	WindowClassName [0..1]	Indicates the specific type of the Window.
function	WindowFunctionValue [0..*]	Specifies the intended purposes of the Window.
usage	WindowUsageValue [0..*]	Specifies the actual uses of the Window.
adeOfWindow	ADEOfWindow [0..*]	Augments the Window with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

WindowSurface

Definition: A WindowSurface is either a boundary surface of a Window feature or a surface that seals an opening filled by a window.

Subclass of: [AbstractFillingSurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

adeOfWindow Surface	ADEOfWindowSurf ace [0..*]	Augments the WindowSurface with properties defined in an ADE.
---------------------	--	---

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.15.2. Data Types

ADEOfAbstractConstruction

Definition: ADEOfAbstractConstruction acts as a hook to define properties within an ADE that are to be added to AbstractConstruction.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractConstructionSurface

Definition: ADEOfAbstractConstructionSurface acts as a hook to define properties within an ADE that are to be added to AbstractConstructionSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractConstructiveElement

Definition: ADEOfAbstractConstructiveElement acts as a hook to define properties within an ADE that are to be added to AbstractConstructiveElement.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractFillingElement

Definition: ADEOfAbstractFillingElement acts as a hook to define properties within an ADE that are to be added to AbstractFillingElement.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractFillingSurface

Definition: ADEOfAbstractFillingSurface acts as a hook to define properties within an ADE that are to be added to AbstractFillingSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractFurniture

Definition: ADEOfAbstractFurniture acts as a hook to define properties within an ADE that are to be added to AbstractFurniture.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractInstallation

Definition: ADEOfAbstractInstallation acts as a hook to define properties within an ADE that are to be added to AbstractInstallation.

Subclass of: None

Stereotype: «DataType»

ADEOfCeilingSurface

Definition: ADEOfCeilingSurface acts as a hook to define properties within an ADE that are to be added to a CeilingSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfDoor

Definition: ADEOfDoor acts as a hook to define properties within an ADE that are to be added to a Door.

Subclass of: None

Stereotype: «DataType»

ADEOfDoorSurface

Definition: ADEOfDoorSurface acts as a hook to define properties within an ADE that are to be added to a DoorSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfFloorSurface

Definition: ADEOfFloorSurface acts as a hook to define properties within an ADE that are to be added to a FloorSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfGroundSurface

Definition: ADEOfGroundSurface acts as a hook to define properties within an ADE that are to be added to a GroundSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfInteriorWallSurface

Definition: ADEOfInteriorWallSurface acts as a hook to define properties within an ADE that are to be added to an InteriorWallSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfOtherConstruction

Definition: ADEOfOtherConstruction acts as a hook to define properties within an ADE that are to be added to an OtherConstruction.

Subclass of: None

Stereotype: «DataType»

ADEOfOuterCeilingSurface

Definition: ADEOfOuterCeilingSurface acts as a hook to define properties within an ADE that are to be added to an OuterCeilingSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfOuterFloorSurface

Definition: ADEOfOuterFloorSurface acts as a hook to define properties within an ADE that are to be added to an OuterFloorSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfRoofSurface

Definition: ADEOfRoofSurface acts as a hook to define properties within an ADE that are to be added to a RoofSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfWallSurface

Definition: ADEOfWallSurface acts as a hook to define properties within an ADE that are to be added to a WallSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfWindow

Definition:	ADEOfWindow acts as a hook to define properties within an ADE that are to be added to a Window.
Subclass of:	None
Stereotype:	«DataType»

ADEOfWindowSurface

Definition:	ADEOfWindowSurface acts as a hook to define properties within an ADE that are to be added to a WindowSurface.
Subclass of:	None
Stereotype:	«DataType»

ConstructionEvent

Definition:	A ConstructionEvent is a data type used to describe a specific event that is associated with a construction. Examples are the issuing of a building permit or the renovation of a building.
Subclass of:	None
Stereotype:	«DataType»

Attribute	Value type and multiplicity	Definition
event	EventValue [1..1]	Indicates the specific event type.
dateOfEvent	Date [1..1]	Specifies the date at which the event took or will take place.
description	CharacterString [0..1]	Provides additional information on the event.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Elevation

Definition:	Elevation is a data type that includes the elevation value itself and information on how this elevation was measured. [cf. INSPIRE]
Subclass of:	None
Stereotype:	«DataType»

Attribute	Value type and multiplicity	Definition
elevationReference	ElevationReference Value [1..1]	Specifies the level from which the elevation was measured. [cf. INSPIRE]
elevationValue	DirectPosition [1..1]	Specifies the value of the elevation. [cf. INSPIRE]

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Height

Definition:	Height represents a vertical distance (measured or estimated) between a low reference and a high reference. [cf. INSPIRE]
Subclass of:	None
Stereotype:	«DataType»

Attribute	Value type and multiplicity	Definition
highReference	ElevationReference Value [1..1]	Indicates the high point used to calculate the value of the height. [cf. INSPIRE]
lowReference	ElevationReference Value [1..1]	Indicates the low point used to calculate the value of the height. [cf. INSPIRE]
status	HeightStatusValue [1..1]	Indicates the way the height has been captured. [cf. INSPIRE]
value	Length [1..1]	Specifies the value of the height above or below ground. [cf. INSPIRE]

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.15.3. Basic Types

none

9.15.4. Unions

none

9.15.5. Code Lists

DoorClassValue

Definition: DoorClassValue is a code list used to further classify a Door.

Stereotype: «CodeList»

DoorFunctionValue

Definition: DoorFunctionValue is a code list that enumerates the different purposes of a Door.

Stereotype: «CodeList»

DoorUsageValue

Definition: DoorUsageValue is a code list that enumerates the different uses of a Door.

Stereotype: «CodeList»

ElevationReferenceValue

Definition: ElevationReferenceValue is a code list that enumerates the different elevation reference levels used to measure construction heights.

Stereotype: «CodeList»

EventValue

Definition: EventValue is a code list that enumerates the different events of a construction.

Stereotype: «CodeList»

OtherConstructionClassValue

Definition: OtherConstructionClassValue is a code list used to further classify an OtherConstruction.

Stereotype: «CodeList»

OtherConstructionFunctionValue

Definition: OtherConstructionFunctionValue is a code list that enumerates the different purposes of an OtherConstruction.

Stereotype: «CodeList»

OtherConstructionUsageValue

Definition: OtherConstructionUsageValue is a code list that enumerates the different uses of an OtherConstruction.

Stereotype: «CodeList»

WindowClassValue

Definition: WindowClassValue is a code list used to further classify a Window.

Stereotype: «CodeList»

WindowFunctionValue

Definition: WindowFunctionValue is a code list that enumerates the different purposes of a Window.

Stereotype: «CodeList»

WindowUsageValue

Definition: WindowUsageValue is a code list that enumerates the different uses of a Window.

Stereotype: «CodeList»

9.15.6. Enumerations

ConditionOfConstructionValue

Definition: ConditionOfConstructionValue enumerates different conditions of a construction. [cf. INSPIRE]

Stereotype: <<Enumeration>>

Literal value	Definition
declined	Indicates that the construction cannot be used under normal conditions, though its main elements (walls, roof) are still present. [cf. INSPIRE]
demolished	Indicates that the construction has been demolished. There are no more visible remains. [cf. INSPIRE]
functional	Indicates that the construction is functional. [cf. INSPIRE]
projected	Indicates that the construction is being designed. Construction works have not yet started. [cf. INSPIRE]
ruin	Indicates that the construction has been partly demolished and some main elements (roof, walls) have been destroyed. There are some visible remains of the construction. [cf. INSPIRE]
underConstruction	Indicates that the construction is under construction and not yet functional. This applies only to the initial construction works of the construction and not to maintenance work. [cf. INSPIRE]

HeightStatusValue

Definition: HeightStatusValue enumerates the different methods used to capture a height. [cf. INSPIRE]

Stereotype: <>Enumeration>>

Literal value	Definition
estimated	Indicates that the height has been estimated and not measured. [cf. INSPIRE]
measured	Indicates that the height has been (directly or indirectly) measured. [cf. INSPIRE]

RelationToConstruction

Definition: RelationToConstruction is an enumeration used to describe whether an installation is positioned inside and/or outside of a construction.

Stereotype: <>Enumeration>>

Literal value	Definition
inside	Indicates that the installation is positioned inside of the construction.
outside	Indicates that the installation is positioned outside of the construction.
bothInsideAndOut	Indicates that the installation is positioned inside as well as outside of the construction.

9.16. Bridge

Description: The Bridge module supports representation of thematic and spatial aspects of bridges, bridge parts, bridge installations, and interior bridge structures.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.16.1. Classes

AbstractBridge

Definition: AbstractBridge is an abstract superclass representing the common attributes and associations of the classes Bridge and BridgePart.

Subclass of: [AbstractConstruction](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
bridgeConstructiveElement	BridgeConstructive Element [*]	Relates the constructive elements to the Bridge or BridgePart.
bridgeInstallation	BridgeInstallation [*]	Relates the installation objects to the Bridge or BridgePart.
bridgeFurniture	BridgeFurniture [*]	Relates the furniture objects to the Bridge or BridgePart.
bridgeRoom	BridgeRoom [*]	Relates the rooms to the Bridge or BridgePart.
address	Address [*]	Relates the addresses to the Bridge or BridgePart.

Attribute	Value type and multiplicity	Definition
class	BridgeClassValue [0..1]	Indicates the specific type of the Bridge or BridgePart.
function	BridgeFunctionValue [0..*]	Specifies the intended purposes of the Bridge or BridgePart.
usage	BridgeUsageValue [0..*]	Specifies the actual uses of the Bridge or BridgePart.
isMovable	Boolean [0..1]	Indicates whether the Bridge or BridgePart can be moved to allow for watercraft to pass.
adeOfAbstractBridge	ADEOfAbstractBridge [0..*]	Augments AbstractBridge with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Bridge

Definition:	A Bridge represents a structure that affords the passage of pedestrians, animals, vehicles, and service(s) above obstacles or between two points at a height above ground. [cf. ISO 6707-1]
Subclass of:	AbstractBridge
Stereotype:	«TopLevelFeatureType»

Role name	Target class and multiplicity	Definition
bridgePart	BridgePart [*]	Relates the bridge parts to the Bridge.

Attribute	Value type and multiplicity	Definition
adeOfBridge	ADEOfBridge [0..*]	Augments the Bridge with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BridgeConstructiveElement

Definition:	A BridgeConstructiveElement is an element of a bridge which is essential from a structural point of view. Examples are pylons, anchorages, slabs, beams.
Subclass of:	AbstractConstructiveElement
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
class	BridgeConstructiveElementClassValue [0..1]	Indicates the specific type of the BridgeConstructiveElement.
function	BridgeConstructiveElementFunctionValue [0..*]	Specifies the intended purposes of the BridgeConstructiveElement.
usage	BridgeConstructiveElementUsageValue [0..*]	Specifies the actual uses of the BridgeConstructiveElement.
adeOfBridgeConstructiveElement	ADEOfBridgeConstructiveElement [0..*]	Augments the BridgeConstructiveElement with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

BridgeFurniture

Attribute	Value type and multiplicity	Definition
Definition:	A BridgeFurniture is an equipment for occupant use, usually not fixed to the bridge. [cf. ISO 6707-1]	
Subclass of:	AbstractFurniture	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
class	BridgeFurnitureClassValue [0..1]	Indicates the specific type of the BridgeFurniture.
function	BridgeFurnitureFunctionValue [0..*]	Specifies the intended purposes of the BridgeFurniture.
usage	BridgeFurnitureUsageValue [0..*]	Specifies the actual uses of the BridgeFurniture.
adeOfBridgeFurniture	ADEOfBridgeFurniture [0..*]	Augments the BridgeFurniture with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

BridgeInstallation

Definition:	A BridgeInstallation is a permanent part of a Bridge (inside and/or outside) which does not have the significance of a BridgePart. In contrast to BridgeConstructiveElements, a BridgeInstallation is not essential from a structural point of view. Examples are stairs, antennas or railways.	
Subclass of:	AbstractInstallation	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
class	BridgeInstallationC classValue [0..1]	Indicates the specific type of the BridgeInstallation.
function	BridgeInstallationF unctionValue [0..*]	Specifies the intended purposes of the BridgeInstallation.
usage	BridgeInstallationU sageValue [0..*]	Specifies the actual uses of the BridgeInstallation.
adeOfBridgeI nstallation	ADEOfBridgeInstall ation [0..*]	Augments the BridgeInstallation with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BridgePart

Definition:	A BridgePart is a physical or functional subdivision of a Bridge. It would be considered a Bridge, if it were not part of a collection of other BridgeParts.	
Subclass of:	AbstractBridge	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
adeOfBridgeP art	ADEOfBridgePart [0..*]	Augments the BridgePart with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BridgeRoom

Definition:	A BridgeRoom is a space within a Bridge or BridgePart intended for human occupancy (e.g. a place of work or recreation) and/or containment (storage) of animals or things. A BridgeRoom is bounded physically and/or virtually (e.g. by ClosureSurfaces or GenericSurfaces).	
Subclass of:	AbstractUnoccupiedSpace	
Stereotype:	«FeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
bridgeInstallation	BridgeInstallation [*]	Relates to the installation objects to the BridgeRoom.
boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the BridgeRoom. This relation is inherited from the Core module.
bridgeFurniture	BridgeFurniture [*]	Relates the furniture objects to the BridgeRoom.
<hr/>		
Attribute	Value type and multiplicity	Definition
class	BridgeRoomClassV alue [0..1]	Indicates the specific type of the BridgeRoom.
function	BridgeRoomFunctionV alue [0..*]	Specifies the intended purposes of the BridgeRoom.
usage	BridgeRoomUsageV alue [0..*]	Specifies the actual uses of the BridgeRoom.
adeOfBridgeRoom	ADEOfBridgeRoom [0..*]	Augments the BridgeRoom with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

9.16.2. Data Types

ADEOfAbstractBridge

Definition:	ADEOfAbstractBridge acts as a hook to define properties within an ADE that are to be added to AbstractBridge.
Subclass of:	None
Stereotype:	«DataType»

ADEOfBridge

Definition: ADEOfBridge acts as a hook to define properties within an ADE that are to be added to a Bridge.

Subclass of: None

Stereotype: «DataType»

ADEOfBridgeConstructiveElement

Definition: ADEOfBridgeConstructiveElement acts as a hook to define properties within an ADE that are to be added to a BridgeConstructiveElement.

Subclass of: None

Stereotype: «DataType»

ADEOfBridgeFurniture

Definition: ADEOfBridgeFurniture acts as a hook to define properties within an ADE that are to be added to a BridgeFurniture.

Subclass of: None

Stereotype: «DataType»

ADEOfBridgeInstallation

Definition: ADEOfBridgeInstallation acts as a hook to define properties within an ADE that are to be added to a BridgeInstallation.

Subclass of: None

Stereotype: «DataType»

ADEOfBridgePart

Definition: ADEOfBridgePart acts as a hook to define properties within an ADE that are to be added to a BridgePart.

Subclass of: None

Stereotype: «DataType»

ADEOfBridgeRoom

Definition: ADEOfBridgeRoom acts as a hook to define properties within an ADE that are to be added to a BridgeRoom.

Subclass of: None

Stereotype: «DataType»

9.16.3. Basic Types

none

9.16.4. Unions

none

9.16.5. Code Lists

BridgeClassValue

Definition: BridgeClassValue is a code list used to further classify a Bridge.

Stereotype: «CodeList»

BridgeConstructiveElementClassValue

Definition: BridgeConstructiveElementClassValue is a code list used to further classify a BridgeConstructiveElement.

Stereotype: «CodeList»

BridgeConstructiveElementFunctionValue

Definition: BridgeConstructiveElementFunctionValue is a code list that enumerates the different purposes of a BridgeConstructiveElement.

Stereotype: «CodeList»

BridgeConstructiveElementUsageValue

Definition:	BridgeConstructiveElementUsageValue is a code list that enumerates the different uses of a BridgeConstructiveElement.
Stereotype:	«CodeList»

BridgeFunctionValue

Definition:	BridgeFunctionValue is a code list that enumerates the different purposes of a Bridge.
Stereotype:	«CodeList»

BridgeFurnitureClassValue

Definition:	BridgeFurnitureClassValue is a code list used to further classify a BridgeFurniture.
Stereotype:	«CodeList»

BridgeFurnitureFunctionValue

Definition:	BridgeFurnitureFunctionValue is a code list that enumerates the different purposes of a BridgeFurniture.
Stereotype:	«CodeList»

BridgeFurnitureUsageValue

Definition:	BridgeFurnitureUsageValue is a code list that enumerates the different uses of a BridgeFurniture.
Stereotype:	«CodeList»

BridgeInstallationClassValue

Definition:	BridgeInstallationClassValue is a code list used to further classify a BridgeInstallation.
Stereotype:	«CodeList»

BridgeInstallationFunctionValue

Definition: BridgeInstallationFunctionValue is a code list that enumerates the different purposes of a BridgeInstallation.

Stereotype: «CodeList»

BridgeInstallationUsageValue

Definition: BridgeInstallationUsageValue is a code list that enumerates the different uses of a BridgeInstallation.

Stereotype: «CodeList»

BridgeRoomClassValue

Definition: BridgeRoomClassValue is a code list used to further classify a BridgeRoom.

Stereotype: «CodeList»

BridgeRoomFunctionValue

Definition: BridgeRoomFunctionValue is a code list that enumerates the different purposes of a BridgeRoom.

Stereotype: «CodeList»

BridgeRoomUsageValue

Definition: BridgeRoomUsageValue is a code list that enumerates the different uses of a BridgeRoom.

Stereotype: «CodeList»

BridgeUsageValue

Definition: BridgeUsageValue is a code list that enumerates the different uses of a Bridge.

Stereotype: «CodeList»

9.16.6. Enumerations

none

9.17. Building

Description: The Building module supports representation of thematic and spatial aspects of buildings, building parts, building installations, building subdivisions, and interior building structures.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.17.1. Classes

AbstractBuilding

Definition: AbstractBuilding is an abstract superclass representing the common attributes and associations of the classes Building and BuildingPart.

Subclass of: [AbstractConstruction](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
buildingFurniture	BuildingFurniture [*]	Relates the furniture objects to the Building or BuildingPart.
buildingRoom	BuildingRoom [*]	Relates the rooms to the Building or BuildingPart.
buildingInstallation	BuildingInstallation n [*]	Relates the installation objects to the Building or BuildingPart.
buildingSubdivision	AbstractBuildingSubdivision [*]	Relates the logical subdivisions to the Building or BuildingPart.
buildingConstructiveElement	BuildingConstructiveElement [*]	Relates the constructive elements to the Building or BuildingPart.
address	Address [*]	Relates the addresses to the Building or BuildingPart.

Attribute	Value type and multiplicity	Definition
class	BuildingClassValue [0..1]	Indicates the specific type of the Building or BuildingPart.
function	BuildingFunctionValue [0..*]	Specifies the intended purposes of the Building or BuildingPart.
usage	BuildingUsageValue [0..*]	Specifies the actual uses of the Building or BuildingPart.
roofType	RoofTypeValue [0..1]	Indicates the shape of the roof of the Building or BuildingPart.
storeysAboveGround	Integer [0..1]	Indicates the number of storeys positioned above ground level.
storeysBelowGround	Integer [0..1]	Indicates the number of storeys positioned below ground level.
storeyHeightsAboveGround	MeasureOrNilReasonList [0..1]	Lists the heights of each storey above ground. The first value in the list denotes the height of the storey closest to the ground level, the last value denotes the height furthest away.
storeyHeightsBelowGround	MeasureOrNilReasonList [0..1]	Lists the height of each storey below ground. The first value in the list denotes the height of the storey closest to the ground level, the last value denotes the height furthest away.
adeOfAbstractBuilding	ADEOfAbstractBuilding [0..*]	Augments AbstractBuilding with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractBuildingSubdivision

Definition:	AbstractBuildingSubdivision is the abstract superclass for different kinds of logical building subdivisions.
Subclass of:	AbstractLogicalSpace
Stereotype:	«FeatureType»

Role name	Target class and multiplicity	Definition
buildingRoom	BuildingRoom [*]	Relates the rooms to the building subdivision.
buildingFurniture	BuildingFurniture [*]	Relates the furniture objects to the building subdivision.
buildingConst ructiveEleme nt	BuildingConstructiveElement [*]	Relates the constructive elements to the building subdivision.
buildingInstal lation	BuildingInstallation [*]	Relates the installation objects to the building subdivision.
Attribute	Value type and multiplicity	Definition
class	BuildingSubdivisionClassValue [0..1]	Indicates the specific type of the building subdivision.
function	BuildingSubdivisionFunctionValue [0..*]	Specifies the intended purposes of the building subdivision.
usage	BuildingSubdivisionUsageValue [0..*]	Specifies the actual uses of the building subdivision.
elevation	Elevation [0..*]	Specifies qualified elevations of the building subdivision in relation to a well-defined surface which is commonly taken as origin (e.g. geoid or water level). [cf. INSPIRE]
sortKey	Real [0..1]	Defines an order among the objects that belong to the building subdivision. An example is the sorting of storeys.
adeOfAbstract BuildingSubdi vision	ADEOfAbstractBuildingSubdivision [0..*]	Augments AbstractBuildingSubdivision with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Building

Definition: A Building is a free-standing, self-supporting construction that is roofed, usually walled, and can be entered by humans and is normally designed to stand permanently in one place. It is intended for human occupancy (e.g. a place of work or recreation), habitation and/or shelter of humans, animals or things.

Subclass of: [AbstractBuilding](#)

Stereotype: «TopLevelFeatureType»

Role name	Target class and multiplicity	Definition
------------------	--------------------------------------	-------------------

buildingPart	BuildingPart [*]	Relates the building parts to the Building.
--------------	----------------------------------	---

Attribute	Value type and multiplicity	Definition
------------------	------------------------------------	-------------------

adeOfBuilding	ADEOfBuilding [0..*]	Augments the Building with properties defined in an ADE.
---------------	---	--

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BuildingConstructiveElement

Definition: A BuildingConstructiveElement is an element of a Building which is essential from a structural point of view. Examples are walls, slabs, staircases, beams.

Subclass of: [AbstractConstructiveElement](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
class	BuildingConstructiveElementClassValue [0..1]	Indicates the specific type of the BuildingConstructiveElement.
function	BuildingConstructiveElementFunctionValue [0..*]	Specifies the intended purposes of the BuildingConstructiveElement.
usage	BuildingConstructiveElementUsageValue [0..*]	Specifies the actual uses of the BuildingConstructiveElement.
adeOfBuildingConstructiveElement	ADEOfBuildingConstructiveElement [0..*]	Augments the BuildingConstructiveElement with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BuildingFurniture

Definition:	A BuildingFurniture is an equipment for occupant use, usually not fixed to the building. [cf. ISO 6707-1]
Subclass of:	AbstractFurniture
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
class	BuildingFurnitureClassValue [0..1]	Indicates the specific type of the BuildingFurniture.
function	BuildingFurnitureFunctionValue [0..*]	Specifies the intended purposes of the BuildingFurniture.
usage	BuildingFurnitureUsageValue [0..*]	Specifies the actual uses of the BuildingFurniture.
adeOfBuildingFurniture	ADEOfBuildingFurniture [0..*]	Augments the BuildingFurniture with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BuildingInstallation

Definition:	A BuildingInstallation is a permanent part of a Building (inside and/or outside) which has not the significance of a BuildingPart. Examples are stairs, antennas, balconies or small roofs.	
Subclass of:	AbstractInstallation	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
class	BuildingInstallation nClassValue [0..1]	Indicates the specific type of the BuildingInstallation.
function	BuildingInstallation nFunctionValue [0..*]	Specifies the intended purposes of the BuildingInstallation.
usage	BuildingInstallation nUsageValue [0..*]	Specifies the actual uses of the BuildingInstallation.
adeOfBuilding Installation	ADEOfBuildingInst allation [0..*]	Augments the BuildingInstallation with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BuildingPart

Definition:	A BuildingPart is a physical or functional subdivision of a Building. It would be considered a Building, if it were not part of a collection of other BuildingParts.	
Subclass of:	AbstractBuilding	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
adeOfBuilding Part	ADEOfBuildingPart [0..*]	Augments the BuildingPart with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BuildingRoom

Definition:	A BuildingRoom is a space within a Building or BuildingPart intended for human occupancy (e.g. a place of work or recreation) and/or containment of animals or things. A BuildingRoom is bounded physically and/or virtually (e.g. by ClosureSurfaces or GenericSurfaces).
Subclass of:	AbstractUnoccupiedSpace
Stereotype:	«FeatureType»

Role name	Target class and multiplicity	Definition
buildingInstallation	BuildingInstallation n [*]	Relates the installation objects to the BuildingRoom.
buildingFurniture	BuildingFurniture [*]	Relates the furniture objects to the BuildingRoom.
boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the BuildingRoom. This relation is inherited from the Core module.
Attribute	Value type and multiplicity	Definition
class	BuildingRoomClass Value [0..1]	Indicates the specific type of the BuildingRoom.
function	BuildingRoomFunction value [0..*]	Specifies the intended purposes of the BuildingRoom.
usage	BuildingRoomUsage eValue [0..*]	Specifies the actual uses of the BuildingRoom.
roomHeight	RoomHeight [0..*]	Specifies qualified heights of the BuildingRoom.
adeOfBuildingRoom	ADEOfBuildingRoom [0..*]	Augments the BuildingRoom with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BuildingUnit

Definition:	A BuildingUnit is a logical subdivision of a Building. BuildingUnits are formed according to some homogeneous property like function, ownership, management, or accessibility. They may be separately sold, rented out, inherited, managed, etc.
Subclass of:	AbstractBuildingSubdivision
Stereotype:	«FeatureType»

Role name	Target class and multiplicity	Definition
storey	Storey [*]	Relates to the storeys on which the BuildingUnit is located.
address	Address [*]	Relates to the addresses that are assigned to the BuildingUnit.
Attribute	Value type and multiplicity	Definition
adeOfBuilding Unit	ADEOfBuildingUnit [0..*]	Augments the BuildingUnit with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Storey

Definition:	A Storey is typically a horizontal section of a Building. Storeys are not always defined according to the building structure, but can also be defined according to logical considerations.	
Subclass of:	AbstractBuildingSubdivision	
Stereotype:	«FeatureType»	
Role name	Target class and multiplicity	Definition
boundary	AbstractThematicsSurface [*]	Relates to the surfaces that bound the Storey. This relation is inherited from the Core module.
buildingUnit	BuildingUnit [*]	Relates to the building units that belong to the Storey.
Attribute	Value type and multiplicity	Definition
adeOfStorey	ADEOfStorey [0..*]	Augments the Storey with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.17.2. Data Types

ADEOfAbstractBuilding

Definition: ADEOfAbstractBuilding acts as a hook to define properties within an ADE that are to be added to AbstractBuilding.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractBuildingSubdivision

Definition: ADEOfAbstractBuildingSubdivision acts as a hook to define properties within an ADE that are to be added to AbstractBuildingSubdivision.

Subclass of: None

Stereotype: «DataType»

ADEOfBuilding

Definition: ADEOfBuilding acts as a hook to define properties within an ADE that are to be added to a Building.

Subclass of: None

Stereotype: «DataType»

ADEOfBuildingConstructiveElement

Definition: ADEOfBuildingConstructiveElement acts as a hook to define properties within an ADE that are to be added to a BuildingConstructiveElement.

Subclass of: None

Stereotype: «DataType»

ADEOfBuildingFurniture

Definition: ADEOfBuildingFurniture acts as a hook to define properties within an ADE that are to be added to a BuildingFurniture.

Subclass of: None

Stereotype: «DataType»

ADEOfBuildingInstallation

Definition: ADEOfBuildingInstallation acts as a hook to define properties within an ADE that are to be added to a BuildingInstallation.

Subclass of: None

Stereotype: «DataType»

ADEOfBuildingPart

Definition: ADEOfBuildingPart acts as a hook to define properties within an ADE that are to be added to a BuildingPart.

Subclass of: None

Stereotype: «DataType»

ADEOfBuildingRoom

Definition: ADEOfBuildingRoom acts as a hook to define properties within an ADE that are to be added to a BuildingRoom.

Subclass of: None

Stereotype: «DataType»

ADEOfBuildingUnit

Definition: ADEOfBuildingUnit acts as a hook to define properties within an ADE that are to be added to a BuildingUnit.

Subclass of: None

Stereotype: «DataType»

ADEOfStorey

Definition: ADEOfStorey acts as a hook to define properties within an ADE that are to be added to a Storey.

Subclass of: None

Stereotype: «DataType»

RoomHeight

Definition: The RoomHeight represents a vertical distance (measured or estimated) between a low reference and a high reference. [cf. INSPIRE]

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
highReference	RoomElevationRef [1..1]	Indicates the high point used to calculate the value of the room height.
lowReference	RoomElevationRef [1..1]	Indicates the low point used to calculate the value of the room height.
status	HeightStatusValue [1..1]	Indicates the way the room height has been captured.
value	Length [1..1]	Specifies the value of the room height.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.17.3. Basic Types

none

9.17.4. Unions

none

9.17.5. Code Lists

BuildingClassValue

Definition: BuildingClassValue is a code list used to further classify a Building.

Stereotype: «CodeList»

BuildingConstructiveElementClassValue

Definition: BuildingConstructiveElementClassValue is a code list used to further classify a BuildingConstructiveElement.

Stereotype: «CodeList»

BuildingConstructiveElementFunctionValue

Definition: BuildingConstructiveElementFunctionValue is a code list that enumerates the different purposes of a BuildingConstructiveElement.

Stereotype: «CodeList»

BuildingConstructiveElementUsageValue

Definition: BuildingConstructiveElementUsageValue is a code list that enumerates the different uses of a BuildingConstructiveElement.

Stereotype: «CodeList»

BuildingFunctionValue

Definition: BuildingFunctionValue is a code list that enumerates the different purposes of a Building.

Stereotype: «CodeList»

BuildingFurnitureClassValue

Definition: BuildingFurnitureClassValue is a code list used to further classify a BuildingFurniture.

Stereotype: «CodeList»

BuildingFurnitureFunctionValue

Definition: BuildingFurnitureFunctionValue is a code list that enumerates the different purposes of a BuildingFurniture.

Stereotype: «CodeList»

BuildingFurnitureUsageValue

Definition: BuildingFurnitureUsageValue is a code list that enumerates the different uses of a BuildingFurniture.

Stereotype: «CodeList»

BuildingInstallationClassValue

Definition: BuildingInstallationClassValue is a code list used to further classify a BuildingInstallation.

Stereotype: «CodeList»

BuildingInstallationFunctionValue

Definition: BuildingInstallationFunctionValue is a code list that enumerates the different purposes of a BuildingInstallation.

Stereotype: «CodeList»

BuildingInstallationUsageValue

Definition: BuildingInstallationUsageValue is a code list that enumerates the different uses of a BuildingInstallation.

Stereotype: «CodeList»

BuildingRoomClassValue

Definition: BuildingRoomClassValue is a code list used to further classify a BuildingRoom.

Stereotype: «CodeList»

BuildingRoomFunctionValue

Definition: BuildingRoomFunctionValue is a code list that enumerates the different purposes of a BuildingRoom.

Stereotype: «CodeList»

BuildingRoomUsageValue

Definition: BuildingRoomUsageValue is a code list that enumerates the different uses of a BuildingRoom.

Stereotype: «CodeList»

BuildingSubdivisionClassValue

Definition: BuildingSubdivisionClassValue is a code list used to further classify a BuildingSubdivision.

Stereotype: «CodeList»

BuildingSubdivisionFunctionValue

Definition: BuildingSubdivisionFunctionValue is a code list that enumerates the different purposes of a BuildingSubdivision.

Stereotype: «CodeList»

BuildingSubdivisionUsageValue

Definition: BuildingSubdivisionUsageValue is a code list that enumerates the different uses of a BuildingSubdivision.

Stereotype: «CodeList»

BuildingUsageValue

Definition: BuildingUsageValue is a code list that enumerates the different uses of a Building.

Stereotype: «CodeList»

RoofTypeValue

Definition: RoofTypeValue is a code list that enumerates different roof types.

Stereotype: «CodeList»

RoomElevationReferenceValue

Definition: RoomElevationReferenceValue is a code list that enumerates the different elevation reference levels used to measure room heights.

Stereotype: «CodeList»

9.17.6. Enumerations

none

9.18. Tunnel

Description: The Tunnel module supports representation of thematic and spatial aspects of tunnels, tunnel parts, tunnel installations, and interior tunnel structures.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.18.1. Classes

AbstractTunnel

Definition: AbstractTunnel is an abstract superclass representing the common attributes and associations of the classes Tunnel and TunnelPart.

Subclass of: [AbstractConstruction](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
hollowSpace	HollowSpace [*]	Relates the hollow spaces to the Tunnel or TunnelPart.
tunnelConstructiveElement	TunnelConstructiveElement [*]	Relates the constructive elements to the Tunnel or TunnelPart.
tunnelInstallation	TunnelInstallation [*]	Relates the installation objects to the Tunnel or TunnelPart.
tunnelFurniture	TunnelFurniture [*]	Relates the furniture objects to the Tunnel or TunnelPart.
Attribute	Value type and multiplicity	Definition
class	TunnelClassValue [0..1]	Indicates the specific type of the Tunnel or TunnelPart.
function	TunnelFunctionValue [0..*]	Specifies the intended purposes of the Tunnel or TunnelPart.
usage	TunnelUsageValue [0..*]	Specifies the actual uses of the Tunnel or TunnelPart.
adeOfAbstractTunnel	ADEOfAbstractTunnel [0..*]	Augments AbstractTunnel with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

HollowSpace

Definition: A HollowSpace is a space within a Tunnel or TunnelPart intended for certain functions (e.g. transport or passage ways, service rooms, emergency shelters). A HollowSpace is bounded physically and/or virtually (e.g. by ClosureSurfaces or GenericSurfaces).

Subclass of: [AbstractUnoccupiedSpace](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
tunnelInstallation	TunnelInstallation [*]	Relates the installation objects to the HollowSpace.
tunnelFurniture	TunnelFurniture [*]	Relates the furniture objects to the HollowSpace.
boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the HollowSpace. This relation is inherited from the Core module.
Attribute	Value type and multiplicity	Definition
class	HollowSpaceClassValue [0..1]	Indicates the specific type of the HollowSpace.
function	HollowSpaceFunctionValue [0..*]	Specifies the intended purposes of the HollowSpace.
usage	HollowSpaceUsageValue [0..*]	Specifies the actual uses of the HollowSpace.
adeOfHollowSpace	ADEOfHollowSpace [0..*]	Augments the HollowSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Tunnel

Definition:	A Tunnel represents a horizontal or sloping enclosed passage way of a certain length, mainly underground or underwater. [cf. ISO 6707-1]	
Subclass of:	AbstractTunnel	
Stereotype:	«TopLevelFeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
tunnelPart	TunnelPart [*]	Relates the tunnel parts to the Tunnel.
<hr/>		
Attribute	Value type and multiplicity	Definition
adeOfTunnel	ADEOfTunnel [0..*]	Augments the Tunnel with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

TunnelConstructiveElement		
<hr/>		
Definition:	A TunnelConstructiveElement is an element of a Tunnel which is essential from a structural point of view. Examples are walls, slabs, beams.	
Subclass of:	AbstractConstructiveElement	
Stereotype:	«FeatureType»	
<hr/>		
Attribute	Value type and multiplicity	Definition
class	TunnelConstructive ElementClassName [0..1]	Indicates the specific type of the TunnelConstructiveElement.
function	TunnelConstructive ElementFunctionValue [0..*]	Specifies the intended purposes of the TunnelConstructiveElement.
usage	TunnelConstructive ElementUsageValue [0..*]	Specifies the actual uses of the TunnelConstructiveElement.
adeOfTunnelC	ADEOfTunnelConst [0..*]	Augments the TunnelConstructiveElement with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

TunnelFurniture

Definition: A TunnelFurniture is an equipment for occupant use, usually not fixed to the tunnel. [cf. ISO 6707-1]

Subclass of: [AbstractFurniture](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
class	TunnelFurnitureCl assValue [0..1]	Indicates the specific type of the TunnelFurniture.
function	TunnelFurnitureFu nctionValue [0..*]	Specifies the intended purposes of the TunnelFurniture.
usage	TunnelFurnitureUs ageValue [0..*]	Specifies the actual uses of the TunnelFurniture.
adeOfTunnelFurniture	ADEOfTunnelFurniture [0..*]	Augments the TunnelFurniture with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TunnelInstallation

Definition: A TunnelInstallation is a permanent part of a Tunnel (inside and/or outside) which does not have the significance of a TunnelPart. In contrast to TunnelConstructiveElement, a TunnelInstallation is not essential from a structural point of view. Examples are stairs, antennas or railings.

Subclass of: [AbstractInstallation](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
class	TunnelInstallation ClassValue [0..1]	Indicates the specific type of the TunnelInstallation.
function	TunnelInstallation FunctionValue [0..*]	Specifies the intended purposes of the TunnelInstallation.
usage	TunnelInstallation UsageValue [0..*]	Specifies the actual uses of the TunnelInstallation.
adeOfTunnelInstallation	ADEOfTunnelInstallation [0..*]	Augments the TunnelInstallation with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TunnelPart

Definition:	A TunnelPart is a physical or functional subdivision of a Tunnel. It would be considered a Tunnel, if it were not part of a collection of other TunnelParts.	
Subclass of:	AbstractTunnel	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
adeOfTunnelPart	ADEOfTunnelPart [0..*]	Augments the TunnelPart with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.18.2. Data Types

ADEOfAbstractTunnel

Definition:	ADEOfAbstractTunnel acts as a hook to define properties within an ADE that are to be added to AbstractTunnel.
Subclass of:	None
Stereotype:	«DataType»

ADEOfHollowSpace

Definition: ADEOfHollowSpace acts as a hook to define properties within an ADE that are to be added to a HollowSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfTunnel

Definition: ADEOfTunnel acts as a hook to define properties within an ADE that are to be added to a Tunnel.

Subclass of: None

Stereotype: «DataType»

ADEOfTunnelConstructiveElement

Definition: ADEOfTunnelConstructiveElement acts as a hook to define properties within an ADE that are to be added to a TunnelConstructiveElement.

Subclass of: None

Stereotype: «DataType»

ADEOfTunnelFurniture

Definition: ADEOfTunnelFurniture acts as a hook to define properties within an ADE that are to be added to a TunnelFurniture.

Subclass of: None

Stereotype: «DataType»

ADEOfTunnelInstallation

Definition: ADEOfTunnelInstallation acts as a hook to define properties within an ADE that are to be added to a TunnelInstallation.

Subclass of: None

Stereotype: «DataType»

ADEOfTunnelPart

Definition: ADEOfTunnelPart acts as a hook to define properties within an ADE that are to be added to a TunnelPart.

Subclass of: None

Stereotype: «DataType»

9.18.3. Basic Types

none

9.18.4. Unions

none

9.18.5. Code Lists

HollowSpaceClassValue

Definition: HollowSpaceClassValue is a code list used to further classify a HollowSpace.

Stereotype: «CodeList»

HollowSpaceFunctionValue

Definition: HollowSpaceFunctionValue is a code list that enumerates the different purposes of a HollowSpace.

Stereotype: «CodeList»

HollowSpaceUsageValue

Definition: HollowSpaceUsageValue is a code list that enumerates the different uses of a HollowSpace.

Stereotype: «CodeList»

TunnelClassValue

Definition: TunnelClassValue is a code list used to further classify a Tunnel.

Stereotype: «CodeList»

TunnelConstructiveElementClassValue

Definition: TunnelConstructiveElementClassValue is a code list used to further classify a TunnelConstructiveElement.

Stereotype: «CodeList»

TunnelConstructiveElementFunctionValue

Definition: TunnelConstructiveElementFunctionValue is a code list that enumerates the different purposes of a TunnelConstructiveElement.

Stereotype: «CodeList»

TunnelConstructiveElementUsageValue

Definition: TunnelConstructiveElementUsageValue is a code list that enumerates the different uses of a TunnelConstructiveElement.

Stereotype: «CodeList»

TunnelFunctionValue

Definition: TunnelFunctionValue is a code list that enumerates the different purposes of a Tunnel.

Stereotype: «CodeList»

TunnelFurnitureClassValue

Definition: TunnelFurnitureClassValue is a code list used to further classify a TunnelFurniture.

Stereotype: «CodeList»

TunnelFurnitureFunctionValue

Definition: TunnelFurnitureFunctionValue is a code list that enumerates the different purposes of a TunnelFurniture.

Stereotype: «CodeList»

TunnelFurnitureUsageValue

Definition: TunnelFurnitureUsageValue is a code list that enumerates the different uses of a TunnelFurniture.

Stereotype: «CodeList»

TunnelInstallationClassValue

Definition: TunnelInstallationClassValue is a code list used to further classify a TunnelInstallation.

Stereotype: «CodeList»

TunnelInstallationFunctionValue

Definition: TunnelInstallationFunctionValue is a code list that enumerates the different purposes of a TunnelInstallation.

Stereotype: «CodeList»

TunnelInstallationUsageValue

Definition: TunnelInstallationUsageValue is a code list that enumerates the different uses of a TunnelInstallation.

Stereotype: «CodeList»

TunnelUsageValue

Definition: TunnelUsageValue is a code list that enumerates the different uses of a Tunnel.

Stereotype: «CodeList»

9.18.6. Enumerations

none

Chapter 10. Application Domain Extension (ADE)

An *Application Domain Extension* (ADE) is a formal and systematic extension of CityGML for a specific application or domain in the form of a conceptual UML model. The application data is mapped to a set of additional classes, attributes, and relations. ADEs may use elements from CityGML, for instance, to derive application-specific subclasses, to inject additional properties, to associate application data with predefined CityGML content, or to define value domains for attributes.

The ADE mechanism allows application-specific information to be aligned with the conceptual model of CityGML in a well-structured and systematic way. By this means, CityGML can be extended to meet the information needs of an application while at the same time preserving its concepts and semantic structures. Moreover, and in contrast to generic city objects and attributes, application data can be validated against the formal definition of an ADE to ensure semantic interoperability.

Previous versions of CityGML defined the ADE mechanism solely on the level of the XML Schema encoding. With CityGML 3.0, ADEs become platform-independent models on a conceptual level that can be mapped to multiple and different target encodings.

ADEs have successfully been implemented in practice and enable a wide range of applications and use cases based on CityGML. An overview and discussion of existing ADEs is provided in [Biljecki et al. 2018].

10.1. General Rules for ADEs

An ADE shall be defined as conceptual model in UML in accordance with the conceptual modelling framework of the ISO 19100 series of International Standards and by adhering to the General Feature Model and the rules and constraints for application schemas as specified in ISO 19109 and ISO/TS 19103. The [UML notations and stereotypes](#) used in the CityGML conceptual model should also be applied to corresponding model elements in an ADE.

Every ADE shall be organized into one or more UML packages having globally unique namespaces and containing all UML model elements defined by the ADE. An ADE may additionally import and use predefined classes from external conceptual UML models such as the CityGML modules or the standardized schemas of the ISO 19100 series of International Standards.

10.2. Defining New ADE Model Elements

Following ISO 19109, features are the primary view of geospatial information and the core elements of application schemas. ADEs therefore typically extend CityGML by defining new feature types appropriate to the application area together with additional content such as object types, data types, code lists, and enumerations.

Every feature type in an ADE shall be derived either directly or indirectly from the CityGML root feature type *Core::AbstractFeature* or, depending on its type and characteristics, from a more appropriate subclass thereof. According to the general space concept of CityGML, features

representing spaces or space boundaries shall be derived either directly or indirectly from *Core::AbstractSpace* or *Core::AbstractSpaceBoundary* respectively. UML classes representing top-level feature types shall use the «*TopLevelFeatureType*» stereotype.

In contrast to feature types, object types and data types are not required to be derived from a predefined CityGML class unless explicitly stated otherwise.

ADE classes may have an unlimited number of attributes and associations in addition to those inherited from their parents. Attributes can be modelled with either simple or complex data types. To ensure semantic interoperability, the predefined types from CityGML or the standardized schemas of the ISO 19100 series of International Standards should be used wherever appropriate. This includes, amongst others, basic types from ISO/TS 19103, geometry and topology objects from ISO 10107, and temporal geometry and topology objects from ISO 19108.

If a predefined type is not available, ADEs can either define their own data types or import data types from external conceptual models. This explicitly includes the possibility to define new geometry types not offered by ISO 19107. Designers of an ADE should however note that software might not be able to properly identify and consume such geometry types.

A feature type capturing a real-world feature with geometry should be derived either directly or indirectly from *Core::AbstractSpace* or *Core::AbstractSpaceBoundary*. By this means, the predefined spatial properties and the associated LOD concept of CityGML are inherited and available for the feature type. If, however, these superclasses are either inappropriate or lack a spatial property required to represent the feature, an ADE may define new and additional spatial properties. If such a spatial property should belong to one of the predefined LODs, then the property name shall start with the prefix “lodX”, where X is to be replaced by an integer value between 0 and 3 indicating the target LOD. This enables software to derive the LOD of the geometry.

Constraints on model elements should be expressed using a formal language such as the Object Constraint Language (OCL). The ADE specifies the manner of application of constraints. However, following the CityGML conceptual model, constraints should at least be expressed on ADE subclasses of *Core::AbstractSpace* to limit the types of space boundaries (i.e., instances of *Core::AbstractSpaceBoundary*) that may be used to model the boundary of a space object.

Illustrative examples for ADEs can be found in the [CityGML 3.0 User Guide](#).

10.3. Augmenting CityGML Feature Types with Additional ADE Properties

If a predefined CityGML feature type lacks one or more properties required for a specific application, a feasible solution in CityGML 2.0 was to derive a new ADE feature type as subclass of the CityGML class and to add the properties to this subclass. While conceptually clean, this approach also faces drawbacks. If multiple ADEs require additional properties for the same CityGML feature type, this will lead to many subclasses of this feature type in different ADE namespaces. Information about the same real-world feature might therefore be spread over various instances of the different feature classes in an encoding making it difficult for software to consume the feature data.

For this reason, CityGML 3.0 provides a way to augment the predefined CityGML feature types with additional properties from the ADE domain without the need for subclassing. Each CityGML feature type has an extension attribute of name “*adeOfFeatureTypeName*” and type “*ADEOfFeatureTypeName*”, where *FeatureTypeName* is replaced by the class name in which the attribute is defined. For example, the *Building::Building* class offers the attribute *adeOfBuilding* of type *Building::ADEOfBuilding*. Each of these extension attributes can occur zero to unlimited times, and the attribute types are defined as abstract and empty data types.

If an ADE augments a specific CityGML feature type with additional ADE properties, the ADE shall create a subclass of the corresponding abstract data type associated with the feature class. This subclass shall also be defined as data type using the stereotype «*DataType*». The additional application-specific attributes and associations are then modelled as properties of the ADE subclass. This may include, amongst others, attributes with simple or complex data type, spatial properties or associations to other object and feature types from the ADE or external models such as CityGML.

The predefined “*ADEOfFeatureTypeName*” data types are called “hooks” because they are used as the head of a hierarchy of ADE subclasses attaching application-specific properties. When subclassing the “hook” of a specific CityGML feature type in an ADE, the properties defined in the subclass can be used for that feature type as well as for all directly or indirectly derived feature types, including feature types defined in the same or another ADE.

Multiple distinct ADEs can use the “hook” mechanism to define additional ADE properties for the same CityGML feature type. Since the “*adeOfFeatureTypeName*” attribute may occur multiple times, the various ADE properties can be exchanged as part of the same CityGML feature instance in an encoding. Software can therefore easily consume the default CityGML feature data plus the additional properties from the different ADEs.

Content from unknown or unsupported ADEs may be ignored by an application or service consuming an encoded CityGML model.

Designers of an ADE should favor using this “hook” mechanism over subclassing a CityGML feature type when possible. If an ADE must enable other ADEs to augment its own feature types (so-called ADE of an ADE), then it shall implement “hooks” for its feature types following the same schema and naming concept as in the CityGML conceptual model.

The UML fragment in [Figure 66](#) shows an example for using the “hook” mechanism. For more details on this and other example ADEs, please see the [CityGML 3.0 User Guide](#) for an example ADE.

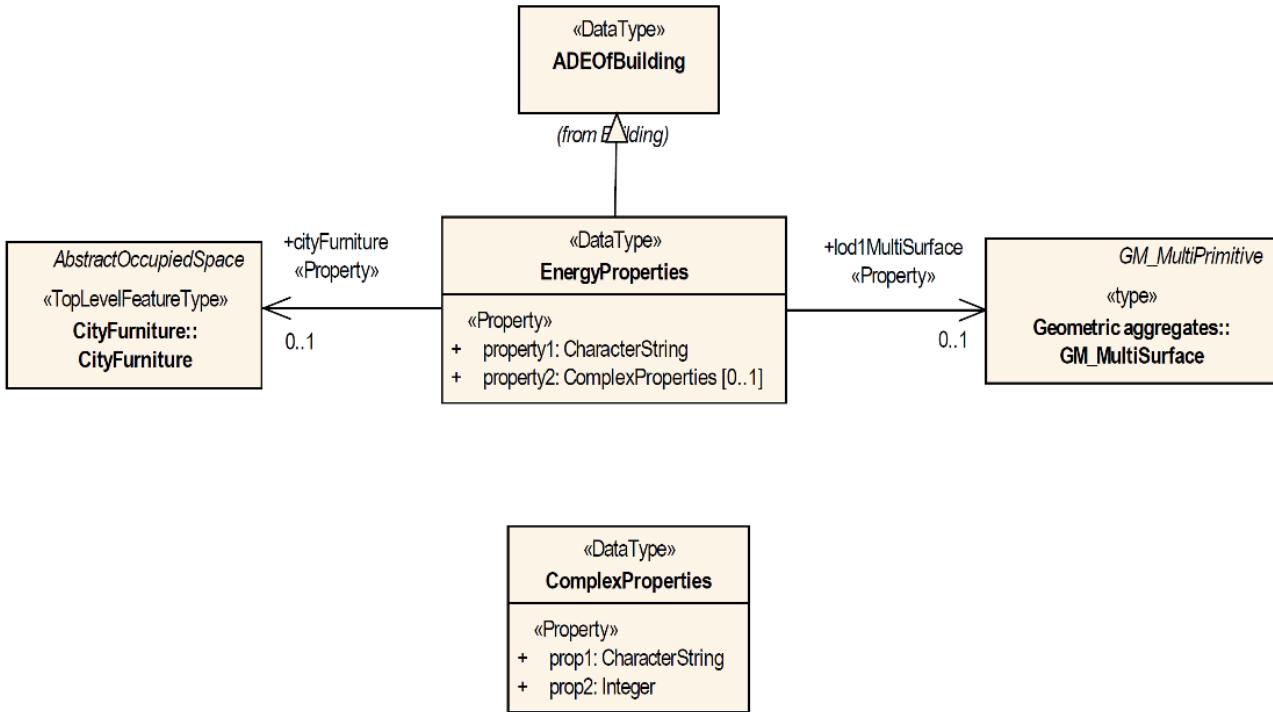


Figure 66. The CityGML feature type `Building` is augmented with additional ADE properties by defining the data type `EnergyProperties` as a subclass of the ADE data type `ADEOfBuilding`.

10.4. Encoding of ADEs

This document only addresses the conceptual modelling of ADEs. Rules and constraints for mapping a conceptual ADE model to a target encoding are expected to be defined in a corresponding CityGML Encoding Standard. If supported, an ADE may provide additional mapping rules and constraints in conformance with a corresponding CityGML Encoding Standard.

10.5. Requirements and Recommendations

The following requirements and recommendations specify how ADEs shall be used as an extension capability to the CityGML Conceptual Model.

Requirements Class	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-ade	
Target type	Conceptual Model
Dependency	/req/req-class-core

10.5.1. UML

Any extension to the CityGML Conceptual Model should be a faithful continuation of the styles and techniques used in that model. The following Requirements and Recommendations define a "faithful continuation".

Requirement 48	/req/ade/uml
An ADE SHALL be defined as conceptual model in UML in accordance with the conceptual modelling framework of the ISO 19100 series of International Standards	
A	The UML model SHALL adhere to the General Feature Model as specified in ISO 19109.
B	The UML model SHALL adhere to rules and constraints for application schemas as specified in ISO/TS 19103.
C	Every ADE SHALL be organized into one or more UML packages having globally unique namespaces and containing all UML model elements defined by the ADE.

Recommendation 1	/rec/ade/uml
In addition to meeting the requirements for a CityGML ADE, an ADE should:	
A	The UML notations and stereotypes used in the CityGML conceptual model SHOULD be applied to corresponding model elements in an ADE.
B	An ADE SHOULD import and use predefined classes from external conceptual UML models such as the CityGML modules or the standardized schemas of the ISO 19100 series of International Standards.

10.5.2. Classes

The following Requirements and Recommendations define how CityGML classes should be extended by an ADE.

Requirement 49	/req/ade/elements
ADEs typically extend CityGML by defining new Feature Types together with additional content such as Object Types, Data Types, Code Lists, and Enumerations.	
A	Every Feature Type in an ADE SHALL be derived either directly or indirectly from the CityGML root Feature Type <i>core:AbstractFeature</i> or a subclass thereof.
B	UML classes representing Top-Level Feature Types SHALL use the « <i>TopLevelFeatureType</i> » stereotype.
C	Features representing spaces or space boundaries SHALL be derived either directly or indirectly from <i>core:AbstractSpace</i> or <i>core:AbstractSpaceBoundary</i> respectively.
D	An ADE may define new and additional spatial properties. If such a spatial property should belong to a predefined LOD, then the property name SHALL start with the prefix “ <i>lodX</i> ”, where <i>X</i> is an integer value indicating the target LOD.

Recommendation	/rec/ade/elements
50	
ADEs typically extend CityGML by defining new feature types together with additional content such as object types, data types, code lists, and enumerations.	
A	ADEs SHOULD use the predefined types from CityGML or the standardized schemas of the ISO 19100 series of International Standards.
B	Constraints on model elements SHOULD be expressed using a formal language such as the Object Constraint Language (OCL).
C	ADE subclasses of <i>core:AbstractSpace</i> SHOULD include constraints to limit the boundaries of the space object.

10.5.3. Properties

The following Requirements define how to use the CityGML extension properties to add attributes to an existing CityGML Feature Type.

Requirement 51	/req/ade/properties
Every Feature Type includes an extension property (hook) of type “ADEOf<FeatureTypeName>” where <FeatureTypeName> is the name of that Feature Type. To add an extension property to a Feature Type:	
A	The ADE SHALL create a subclass of the abstract data type associated with the hook.
B	This subclass SHALL be defined as a data type using the stereotype « <i>DataType</i> ».
C	Application-specific attributes and associations SHALL be modeled as properties of the ADE subclass.

Annex A: Abstract Test Suite (Normative)

A.1. Introduction

CityGML 3.0 is a Conceptual Model. Since it is agnostic to implementing technologies, an Executable Test Script is not feasible. It becomes the responsibility of the Implementation Specifications to provide evidence of conformance. This evidence should be provided as an annex to the Implementation Specification document.

The test method specified in this ATS is manual inspection. Automated methods may be used where they exist.

A.2. Conformance Class Core

Abstract Test 1	/ats/Core/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/Core/classes
Test Method	Manual Inspection

For each UML class defined or referenced in the Core Package:

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.
C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

Abstract Test 2	/ats/Core/ade/uml
Test Purpose	To validate that Application Domain Extensions (ADE) to the CityGML Conceptual Model is modeled correctly in UML.
Requirement	/req/Core/ade/uml
Test Method	Manual Inspection

An ADE is defined as conceptual model in UML in accordance with the conceptual modelling framework of the ISO 19100 series of International Standards

A	Validate that the ADE UML model adheres to the General Feature Model as specified in ISO 19109.
B	Validate that the ADE UML model adheres to rules and constraints for application schemas as specified in ISO/TS 19103.
C	Validate that the ADE UML model is organized into one or more UML packages having globally unique namespaces and containing all UML model elements defined by the ADE.

Abstract Test 3	/ats/Core/ade/elements
Test Purpose	To validate that Application Domain Extensions (ADE) to the CityGML Conceptual Model are implemented correctly.
Requirement	/req/Core/ade/elements
Test Method	Manual Inspection

For each new UML class defined by an ADE:

A	Validate that every Feature Type class in an ADE is derived either directly or indirectly from the CityGML root Feature Type <i>core:AbstractFeature</i> or a subclass thereof.
B	Validate that every UML class in an ADE which represents a top-level Feature Type is assigned the « <i>TopLevelFeatureType</i> » stereotype.
C	Validate that every UML class in an ADE which represents spaces or space boundaries is derived either directly or indirectly from <i>core:AbstractSpace</i> or <i>core:AbstractSpaceBoundary</i> respectively.

D	<p>Validate that any new or additional spatial properties defined by an ADE:</p> <ol style="list-style-type: none"> 1. belongs to a predefined LOD, 2. has a property name which starts with the prefix “lodX”, where X is an integer value indicating the target LOD.
---	--

Abstract Test 4	/ats/Core/ade/properties
Test Purpose	To validate that Application Domain Extension s (ADE) to the CityGML Conceptual Model implement extension properties correctly.
Requirement	/req/Core/ade/proerties
Test Method	Manual Inspection

Every Feature Type in the CityGML Conceptual Model includes an extension property whose purpose is to allow an ADE to add properties to that existing Feature Type. In every case where an extension property has been used:

A	Validate that the ADE creates a subclass of the abstract data type associated with the extension property.
B	Validate that this subclass is defined as a data type using the stereotype « <i>«DataType»</i> ».
C	Validate that all application-specific attributes and associations for that Feature Type are modeled as properties of the ADE subclass.

Abstract Test 5	/ats/Core/generics
Test Purpose	To validate the correct use of Generics objects and attributes.
Requirement	/req/Core/generics
Test Method	Manual Inspection

For each Generic object and attribute defined in an Implementation Specification, validate that:

A	The object or attribute does not duplicate one that has already been defined by the Conceptual Model. (is unique)
---	---

B	That other extension methods provided by the CityGML Conceptual Model do not provide a more specific representation. (is appropriate)
---	---

Abstract Test 6	/ats/Core/isorestrictions
Test Purpose	To validate that none of the restrictions which the CityGML Conceptual Model imposes on ISO classes are violated by an Implementation Specification.
Requirement	/req/Core/isorestrictions
Test Method	Manual Inspection
A	For each instance of the GM_Solid class, validate that there are no interior boundaries associated with that instance.
B	For each instance of a class descended from the GM_Solid class, validate that there are no interior boundaries associated with that instance.

A.3. Conformance Class Appearance

Abstract Test 7	/ats/Appearance/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/Appearance/classes
Test Method	Manual Inspection

For each UML class defined or referenced in the Appearance Package:

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.

C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.4. Conformance Class CityFurniture

Abstract Test 8	/ats/CityFurniture/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/CityFurniture/classes
Test Method	Manual Inspection

For each UML class defined or referenced in the CityFurniture Package:

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.
C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.5. Conformance Class CityObjectGroup

Abstract Test 9	/ats/CityObjectGroup/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/CityObjectGroup/classes
Test Method	Manual Inspection
For each UML class defined or referenced in the CityObjectGroup Package:	
A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.
C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.6. Conformance Class Dynamizer

Abstract Test 10	/ats/Dynamizer/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/Dynamizer/classes
Test Method	Manual Inspection
For each UML class defined or referenced in the Dynamizer Package:	

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.
C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.7. Conformance Class Generics

Abstract Test 11	/ats/Generics/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/Generics/classes
Test Method	Manual Inspection

For each UML class defined or referenced in the Generics Package:

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.

C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.8. Conformance Class LandUse

Abstract Test 12	/ats/LandUse/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/LandUse/classes
Test Method	Manual Inspection

For each UML class defined or referenced in the LandUse Package:

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.
C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.9. Conformance Class PointCloud

Abstract Test 13	/ats/PointCloud/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/PointCloud/classes
Test Method	Manual Inspection
For each UML class defined or referenced in the PointCloud Package:	
A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.
C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.10. Conformance Class Relief

Abstract Test 14	/ats/Relief/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/Relief/classes
Test Method	Manual Inspection
For each UML class defined or referenced in the Relief Package:	

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.
C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.11. Conformance Class Transportation

Abstract Test 15	/ats/Transportation/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/Transportation/classes
Test Method	Manual Inspection

For each UML class defined or referenced in the Transportation Package:

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.

C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.12. Conformance Class Vegetation

Abstract Test 16	/ats/Vegetation/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/Vegetation/classes
Test Method	Manual Inspection

For each UML class defined or referenced in the Vegetation Package:

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.
C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.13. Conformance Class Versioning

Abstract Test 17	/ats/Versioning/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/Versioning/classes
Test Method	Manual Inspection

For each UML class defined or referenced in the Versioning Package:

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.
C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.14. Conformance Class WaterBody

Abstract Test 18	/ats/Waterbody/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/Waterbody/classes
Test Method	Manual Inspection

For each UML class defined or referenced in the Waterbody Package:

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.
C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.15. Conformance Class Construction

Abstract Test 19	/ats/Construction/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/Construction/classes
Test Method	Manual Inspection

For each UML class defined or referenced in the Construction Package:

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.

C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.16. Conformance Class Bridge

Abstract Test 20	/ats/Bridge/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/Bridge/classes
Test Method	Manual Inspection

For each UML class defined or referenced in the Bridge Package:

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.
C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.17. Conformance Class Building

Abstract Test 21	/ats/Building/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/Building/classes
Test Method	Manual Inspection

For each UML class defined or referenced in the Building Package:

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.
C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

A.18. Conformance Class Tunnel

Abstract Test 22	/ats/Tunnel/classes
Test Purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Requirement	/req/Tunnel/classes
Test Method	Manual Inspection

For each UML class defined or referenced in the Tunnel Package:

A	Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class.
B	Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and cardinalities as those documented in the Conceptual Model.
C	Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model.
D	Validate that the properties of the data element include those of all parent classes of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and cardinality of those documented in the Conceptual Model

Annex B: Revision History

Date	Release	Editor	Primary clauses modified	Description
2020-06-04	0.9.0	C. Heazel	all	Draft for review
2020-06-07	0.9.1	T. H. Kolbe	Chapter 10	Bibliography was added
2020-06-08	0.9.2	C. Nagel	Chapter 10	Chapter on ADE mechanism was added
2020-06-11	0.9.3	T. H. Kolbe	Chapter 7	Overview chapter on CityGML was added
2020-06-11	0.9.4	T. Kutzner	Chapter 0	List of participants and submitters was added
2020-08-05	0.9.5	T. Kutzner	Chapter 8	Boundary constraints were added
2020-08-05	0.9.6	C. Heazel, T. Kutzner	Chapter 8	UML update

Annex C: Changelog for CityGML 3.0

The following table lists all feature types, properties, and data types which have been added or changed for CityGML 3.0.

Feature Class / Data Type	Property	New	Changed	Deleted	Description of Change

Annex D: Glossary

conformance test class

set of conformance test modules that must be applied to receive a single certificate of conformance
[OGC 08-131r3, definition 4.4]

feature

abstraction of real world phenomena
[ISO 19101-1:2014, definition 4.1.11]

feature attribute

characteristic of a feature
[ISO 19101-1:2014, definition 4.1.12]

feature type

class of features having common characteristics
[ISO 19156:2011, definition 4.7]

measurement

set of operations having the object of determining the value of a quantity
[ISO 19101-2:2018, definition 3.21] / [VIM:1993, 2.1]

model

abstraction of some aspects of reality
[ISO 19109:2015, definition 4.15]

observation

act of measuring or otherwise determining the value of a property
[ISO 19156:2011, definition 4.11]

observation procedure

method, algorithm or instrument, or system of these, which may be used in making an observation
[ISO 19156:2011, 4.12]

observation result

estimate of the value of a property determined through a known observation procedure
[ISO 19156:2011, 4.14]

property

facet or attribute of an object referenced by a name.
[ISO 19143:2010, definition 4.21]

requirements class

aggregate of all requirement modules that must all be satisfied to satisfy a conformance test class
[OGC 08-131r3, definition 4.19]

schema

formal description of a model
[ISO 19101-1:2014, definition 4.1.34]

sensor

type of observation procedure that provides the estimated value of an observed property at its output

[OGC 08-094r1, definition 4.5]

Standardization Target

TBD

timeseries

sequence of data values which are ordered in time

[OGC 15-043r3]

universe of discourse

view of the real or hypothetical world that includes everything of interest

[ISO 19101-1:2014, definition 4.1.38]

version

Particular variation of a spatial object

[INSPIRE Glossary]

Chapter 11. ISO Concepts

The following concepts from the ISO TC211 Harmonized UML model are referenced by the CityGML Conceptual UML model but do not play a major role in its' definition. They are provided here to support a more complete understanding of the model.

Area

The measure of the physical extent of any topologically 2-D geometric object. Usually measured in "square" units of length.

[ISO 19103:2015]

Boolean

boolean is the mathematical datatype associated with two-valued logic

[ISO 19103:2015]

CC_CoordinateOperation

mathematical operation on coordinates that transforms or converts coordinates to another coordinate reference system.

[ISO 19111:2019]

Character

symbol from a standard character-set.

[ISO 19103:2015]

CharacterString

Characterstring is a family of datatypes which represent strings of symbols from standard character-sets.

[ISO 19103:2015]

CRS

Coordinate reference system which is usually single but may be compound.

[ISO 19111:2019]

CV_DiscreteCoverage

A subclass of CV_Coverage that returns a single record of values for any direct position within a single geometric object in its spatiotemporal domain.

[ISO 19123:2005]

CV_DomainObject

[ISO 19123:2005]

CV_GridPointValuePair

[ISO 19123:2005]

CV_GridValuesMatrix

The geometry represented by the various offset vectors is in the image plane of the grid.

[ISO 19123:2005]

CV_ReferenceableGrid

[ISO 19123:2005]

Date

[ISO 19103:2015]

DateTime

[ISO 19103:2015]

Distance

Used as a type for returning distances and possibly lengths.

[ISO 19103:2015]

Engineering CRS

A contextually local coordinate reference system which can be divided into two broad categories:

1. earth-fixed systems applied to engineering activities on or near the surface of the earth;
2. CRSSs on moving platforms such as road vehicles, vessels, aircraft or spacecraft.

[ISO 19111:2019]

Generic Name

Generic Name is the abstract class for all names in a NameSpace. Each instance of a GenericName is either a LocalName or a ScopedName.

[ISO 19103:2015]

Geometry

[ISO 19107:2003]

GM_CompositePoint

[ISO 19107:2003]

GM_CompositeSolid

set of geometric solids adjoining one another along common boundary geometric surfaces

[ISO 19107:2003]

GM_GenericSurface

GM_Surface and GM_SurfacePatch both represent sections of surface geometry, and therefore share a number of operation signatures. These are defined in the interface class GM_GenericSurface.

[ISO 19107:2003]

GM_LineString

consists of sequence of line segments, each having a parameterization like the one for GM_LineSegment

[ISO 19107:2003]

GM_MultiPrimitive

[ISO 19107:2003]

GM_OrientableSurface

a surface and an orientation inherited from GM_OrientablePrimitive. If the orientation is "+", then the GM_OrientableSurface is a GM_Surface. If the orientation is "-", then the GM_OrientableSurface is a reference to a GM_Surface with an upNormal that reverses the direction for this GM_OrientableSurface, the sense of "the top of the surface".

[ISO 19107:2003]

GM_PolyhedralSurface

a GM_Surface composed of polygon surfaces (GM_Polygon) connected along their common boundary curves.

[ISO 19107:2003]

GM_Position

a union type consisting of either a DirectPosition or of a reference to a GM_Point from which a DirectPosition shall be obtained.

[ISO 19107:2003]

GM_Primitive

the abstract root class of the geometric primitives. Its main purpose is to define the basic "boundary" operation that ties the primitives in each dimension together.

[ISO 19107:2003]

Integer

An exact integer value, with no fractional part.

[ISO 19103:2015]

IO_IdentifiedObjectBase

[ISO 19103:2015]

Length

The measure of distance as an integral, i.e. the limit of an infinite sum of distances between points on a curve.

[ISO 19103:2015]

Measure

The result from performing the act or process of ascertaining the extent, dimensions, or quantity of some entity.

[ISO 19103:2015]

Number

The base type for all number data, giving the basic algebraic operations.

[ISO 19103:2015]

Point

GM_Point is the basic data type for a geometric object consisting of one and only one point.

[ISO 19107:2003]

Real

The common binary Real finite implementation using base 2.

[ISO 19103:2015]

RS_ReferenceSystem

Description of a spatial and temporal reference system used by a dataset.

[ISO 19111:2019]

Scoped Name

ScopedName is a composite of a LocalName for locating another NameSpace and a GenericName valid in that NameSpace. ScopedName contains a LocalName as head and a GenericName, which might be a LocalName or a ScopedName, as tail.

[ISO 19103:2015]

Solid

GM_Solid, a subclass of GM_Primitive, is the basis for 3-dimensional geometry. The extent of a solid is defined by the boundary surfaces.

[ISO 19107:2003]

Time

[ISO 19103:2015]

TM_Duration

[iso19108]

TM_TemporalPosition

The position of a TM_Instant relative to a TM_ReferenceSystem.

[iso19108]

Unit of Measure

Any of the systems devised to measure some physical quantity such distance or area or a system devised to measure such things as the passage of time.

[ISO 19103:2015]

URI

Uniform Resource Identifier (URI), is a compact string of characters used to identify or name a resource

[ISO 19103:2015]

Volume

[ISO 19103:2015]

Chapter 12. Abbreviated Terms

- 2D Two Dimensional
- 3D Three Dimensional
- AEC Architecture, Engineering, Construction
- ALKIS German National Standard for Cadastral Information
- ATKIS German National Standard for Topographic and Cartographic Information
- BIM Building Information Modeling
- B-Rep Boundary Representation
- bSI buildingSMART International
- CAD Computer Aided Design
- COLLADA Collaborative Design Activity
- CSG Constructive Solid Geometry
- DTM Digital Terrain Model
- DXF Drawing Exchange Format
- EuroSDR European Spatial Data Research Organisation
- ESRI Environmental Systems Research Institute
- FM Facility Management
- GDF Geographic Data Files
- GDI-DE Spatial Data Infrastructure Germany (Geodateninfrastruktur Deutschland)
- GDI NRW Geodata Infrastructure North-Rhine Westphalia
- GML Geography Markup Language
- IAI International Alliance for Interoperability (now buildingSMART International (bSI))
- IETF Internet Engineering Task Force
- IFC Industry Foundation Classes
- ISO International Organization for Standardisation
- ISO/TC211 ISO Technical Committee 211
- LOD Levels of Detail
- NBIMS National Building Information Model Standard
- OASIS Organisation for the Advancement of Structured Information Standards
- OGC Open Geospatial Consortium
- OSCRE Open Standards Consortium for Real Estate
- SIG 3D Special Interest Group 3D of the GDI-DE
- TIC Terrain Intersection Curve
- TIN Triangulated Irregular Network

- UML Unified Modeling Language
- URI Uniform Resource Identifier
- VRML Virtual Reality Modeling Language
- W3C World Wide Web Consortium
- W3DS OGC Web 3D Service
- WFS OGC Web Feature Service
- X3D Open Standards XML-enabled 3D file format of the Web 3D Consortium
- XML Extensible Markup Language
- xAL OASIS extensible Address Language

Annex E: Bibliography

- Open Geospatial Consortium: **The Specification Model—A Standard for Modular specifications**, OGC 08-131
- Agugiaro, G., Benner, J., Cipriano, P., Nouvel, R., 2018: **The Energy Application Domain Extension for CityGML: enhancing interoperability for urban energy simulations**. Open Geospatial Data, Software and Standards, Vol. 3. <https://doi.org/10.1186/s40965-018-0042-y>
- Becker, T., Nagel, C., Kolbe, T. H., 2011: **Integrated 3D Modeling of Multi-utility Networks and their Interdependencies for Critical Infrastructure Analysis**. In: T. H. Kolbe, G. König, C. Nagel (Eds.): Advances in 3D Geoinformation Sciences. LNG&C, Springer, Berlin. https://doi.org/10.1007/978-3-642-12670-3_1
- Beil, C., Kolbe, T. H., 2017: **CityGML and the streets of New York - A proposal for detailed street space modelling**. In: Proceedings of the 12th International 3D GeoInfo Conference 2017, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. IV-4/W5, ISPRS. <http://doi.org/10.5194/isprs-annals-IV-4-W5-9-2017>
- Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., Çöltekin, A., 2015: **Applications of 3D City Models: State of the Art Review**. ISPRS International Journal of Geo-Information, 4(4). <https://doi.org/10.3390/ijgi4042842>
- Biljecki, F., Kumar, K., Nagel, C., 2018: **CityGML Application Domain Extension (ADE): overview of developments**. Open Geospatial Data, Software and Standards, 3(1). <https://doi.org/10.1186/s40965-018-0055-6>
- Billen, R., Zaki, C. E., Servières, M., Moreau, G., Hallot, P., 2012: **Developing an ontology of space: Application to 3D city modeling**. In: Leduc, T., Moreau, G., Billen, R. (eds): Usage, usability, and utility of 3D city models — European COST Action TU0801, EDP Sciences, Nantes, Vol. 02007. <https://hal.archives-ouvertes.fr/hal-01521445>
- Chaturvedi, K., Smyth, C. S., Gesquière, G., Kutzner, T., Kolbe, T. H., 2015: **Managing versions and history within semantic 3D city models for the next generation of CityGML**. In: Selected papers from the 10th International 3DGeoInfo Conference 2015 in Kuala Lumpur, Malaysia, Springer LNG&C, Berlin. https://doi.org/10.1007/978-3-319-25691-7_11
- Chaturvedi, K., Kolbe, T. H., 2016: **Integrating Dynamic Data and Sensors with Semantic 3D City Models in the context of Smart Cities**. In: Proceedings of the 11th International 3D Geoinfo Conference, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. IV-2/W1, ISPRS. <https://doi.org/10.5194/isprs-annals-IV-2-W1-31-2016>
- Chaturvedi, K., Kolbe, T. H., 2017: **Future City Pilot 1 Engineering Report**, Open Geospatial Consortium. [OGC Doc. 19-098](#)
- Chaturvedi, K., Kolbe, T. H., 2019: **A Requirement Analysis on Extending Semantic 3D City Models for Supporting Time-dependent Properties**. In: Proceedings of the 4th International Conference on Smart Data and Smart Cities, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. IV-4/W9, ISPRS. <https://doi.org/10.5194/isprs-annals-IV-4-W9-19-2019>
- Elfes, A., 1989: **Using occupancy grids for mobile robot perception and navigation**. Computer 22(6):46–57. <https://doi.org/10.1109/2.30720>

- Foley, J., van Dam, A., Feiner, S., Hughes, J., 2002: **Computer Graphics: Principles and Practice**. 2nd ed., Addison Wesley
- Gröger, G., Plümer, L., 2012: **CityGML – Interoperable semantic 3D city models**. ISPRS Journal of Photogrammetry and Remote Sensing, Vol. 71, July 2012. <https://dx.doi.org/10.1016/j.isprsjprs.2012.04.004>
- Gröger, G., Kolbe, T. H., Nagel, C., Häfele, K.-H., 2012: **OGC City Geography Markup Language (CityGML) Encoding Standard, Version 2.0.0**, Open Geospatial Consortium. [OGC Doc. 12-019](#)
- Jensen, Christian S. and Dyreson, Curtis E.: **The Consensus Glossary of Temporal Database Concepts**. February 1998 Version. In: Temporal Databases: Research and Practice [online]. Springer Berlin Heidelberg, 1998. p. 367–405. Lecture Notes in Computer Science. Available from: 10.1007/BFb0053710
- Jensen, Christian S. and Snodgrass, Richard T., eds.: **TR-90, Temporal Database Entries for the Springer Encyclopedia of Database Systems**. Technical Report. TimeCenter, 22 May 2008. Available from: <http://timecenter.cs.aau.dk/TimeCenterPublications/TR-90.pdf>
- Johnson, Tom: **Bitemporal Data**. Elsevier, 2014. ISBN 978-0-12-408067-6. Available from: 10.1016/C2012-0-06609-4
- Kaden, R., Clemen, C., 2017: **Applying Geodetic Coordinate Reference Systems within Building Information Modeling (BIM)**. In: Proceedings of the FIG Working Week 2017, Helsinki, Finland. https://www.fig.net/resources/proceedings/fig_proceedings/fig2017/papers/ts06h/TS06H_kaden_clemen_8967.pdf
- Kolbe, T. H., Gröger, G., 2003: **Towards unified 3D city models**. In: Proceedings of the Joint ISPRS Commission IV Workshop on Challenges in Geospatial Analysis, Integration and Visualization II, Stuttgart, Germany. <https://mediatum.ub.tum.de/doc/1145769/>
- Kolbe, T. H., 2009: **Representing and Exchanging 3D City Models with CityGML**. In: J. Lee, S. Zlatanova (Eds.), 3D Geo-Information Sciences, Selected Papers of the 3rd International Workshop on 3D Geo-Information in Seoul, Korea. Springer, Berlin. https://doi.org/10.1007/978-3-540-87395-2_2
- Konde, A., Tauscher, H., Biljecki, F., Crawford, J., 2018: **Floor plans in CityGML**. In: Proceedings of the 13th 3D GeoInfo Conference 2018, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. IV-4/W6, 25–32, ISPRS. <https://doi.org/10.5194/isprs-annals-IV-4-W6-25-2018>
- Kutzner, T., Hijazi, I., Kolbe, T. H., 2018: **Semantic Modelling of 3D Multi-utility Networks for Urban Analyses and Simulations – The CityGML Utility Network ADE**. International Journal of 3-D Information Modeling (IJ3DIM) 7(2), 1-34. <https://dx.doi.org/10.4018/IJ3DIM.2018040101>
- Kutzner, T., Chaturvedi, K. & Kolbe, T. H., 2020: **CityGML 3.0: New Functions Open Up New Applications**. PFG - Journal of Photogrammetry, Remote Sensing and Geoinformation Science, 88, 43–61. <https://doi.org/10.1007/s41064-020-00095-z>
- Labetski, A., van Gerwen, S., Tamminga, G., Ledoux, H., Stoter, J., 2018: **A proposal for an improved transportation model in CityGML**. In: Proceedings of the 13th 3D GeoInfo Conference 2018, ISPRS Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XLII-4/W10, 89–96. <https://doi.org/10.5194/isprs-archives-XLII-4-W10-89-2018>
- Liu, Ling and Özsü, M. Tamer, eds.: **Encyclopedia of Database Systems**. New York, NY :

Springer New York, 2018. ISBN 978-1-4614-8266-6. Available from: 10.1007/978-1-4614-8265-9

- Löwner, M.-O., Gröger, G., Benner, J., Biljecki, F., Nagel, C., 2016: **Proposal for a new LOD and multi-representation concept for CityGML**. In: Proceedings of the 11th 3D Geoinfo Conference 2016, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. IV-2/W1, 3–12. <https://doi.org/10.5194/isprs-annals-IV-2-W1-3-2016>
- Nouvel, R., Bahu, J. M., Kaden, R., Kaempf, J., Cipriano, P., Lauster, M., Haefele, K.-H., Munoz, E., Tournaire, O., Casper, E., 2015: **Development of the CityGML Application Domain Extension Energy for Urban Energy Simulation**. In: Proceedings of Building Simulation 2015 - 14th Conference of the International Building Performance Simulation Association, IBPSA, 559-564. <http://www.ibpsa.org/proceedings/BS2015/p2863.pdf>
- Smith, B., Varzi, A. C., 2000: **Fiat and Bona Fide Boundaries**. Philosophy and Phenomenological Research, Vol. 60, No. 2, 401-420. <https://doi.org/10.2307/2653492>
- Snodgrass, Richard T: **Developing time-oriented database applications in SQL**. San Francisco, California : Morgan Kaufmann Publishers, July 1999. ISBN 1-55860-436-7. Available from: <http://www.cs.arizona.edu/rts/tdbbook.pdf>
- Stadler, A., Kolbe, T. H., 2007: **Spatio-semantic Coherence in the Integration of 3D City Models**. In: Proceedings of the 5th International ISPRS Symposium on Spatial Data Quality ISSDQ 2007 in Enschede. http://www.isprs.org/proceedings/XXXVI/2-C43/Session1/paper_Stadler.pdf
- Vretanos, P. A. 2010: **OpenGIS Web Feature Service 2.0 Interface Standard**, Open Geospatial Consortium. [OGC Doc. 09-025r1](http://www.opengeospatial.org/standards/wfs)