eCH-0031 INTERLIS 2-Reference Manual

Name	INTERLIS 2-Reference Manual
eCH-number	eCH-0031
Category	Standard
Quality stage	Implemented
Version	2.0
Status	Approved
Decision on	2016-09-07
Date of issue	2017-10-27
Replaces version	1.0 Major Change
Requirements	-
Annexes	-
Languages	German (Original), French (Translation), English (Translation)
Authors	Technical Unit Geonormen Joseph Dorfschmid, Adasys AG Claude Eisenhut, Eisenhut Informatik AG Michael Germann, infoGrips GmbH Stefan Keller, Hochschlule für Technik Rapperswil Pirmin Kalberer, Sourcepole AG Hugo Thalmann, a/m/t software service AG Rolf Zürcher, COGIS, Federal Office of Topography
Editor / Distribution	eCH registered association, Mainaustrasse 30, *Postfach* [P.O. Box], 8034 Zürich T 044 388 74 64, F 044 388 71 80 www.ech.ch / info@ech.ch

Summary

INTERLIS is a conceptual description language for geodata and according to GeoIV-swisstopo (SR 510.620.1) mandatory for all modelling of any official geodata subject to federal law.

In the present issue we deal with the INTERLIS 2.4 language version.

Table of contents

Introduction **7**

Status **7**
Preface **7**

1 Basic principles **11**

1.1 Overview **11**
1.2 Utilization of models **12**
1.3 A structure of models and topics **13**
1.4 Object concept **14**

1.4.1 Objects and classes 14

1.4.2 Extension of class 15

1.4.3 Meta models and meta objects 15

1.4.4 Relationships between objects 15

1.4.5 Baskets, replication and data transfer 16

1.5 View concept **18**
1.6 Graphic concept **19**
1.7 Services, tool capacities and conformity **20**
1.8 A small example as an introduction **21**
1.9 How this document is structured **21**

2 Description language **23**

2.1 Syntax applied **23**
2.2 Basic symbols of the language **24**

2.2.1 Character codes utilized, blanks and line ends 24

2.2.2 Names 24

2.2.3 Strings 24

2.2.4 Digits 25

2.2.5 Sets of properties 25

2.2.6 Explanations 25

2.2.7 Special symbols and reserved words 26

2.2.8 Comments 26

2.2.8.1 Line comment 27

2.2.8.2 Block comment 27

2.3 Principal rule **27**
2.4 Inheritance **27**
2.5 Models, topics, classes **27**

2.5.1 Models 27

2.5.2 Topics 29

2.5.3 Classes and structures 31

2.5.4 Namespaces 32

2.6 Attributes ... 33
2.6.1 General comments concerning attributes ... 33
2.6.2 Attributes with domain as type ... 34
2.6.3 Reference attributes ... 34
2.6.4 Structure attributes ... 34
2.7 Proper relationships ... 35
2.7.1 Description of relationships ... 35
2.7.2 Force of relationship ... 37
2.7.3 Cardinality ... 37
2.7.4 Ordered relationships ... 38
2.7.5 Relationship access ... 38
2.8 Domains and constants ... 38
2.8.1 Strings ... 40
2.8.2 Enumerations ... 41
2.8.3 Text orientation ... 42
2.8.4 Boolean ... 43
2.8.5 Numeric data types ... 43
2.8.6 Formatted domains ... 45
2.8.7 Date and time ... 45
2.8.8 Coordinates ... 46
2.8.9 Domains of object identifications ... 48
2.8.10 Blackboxes ... 49
2.8.11 Domains of classes and attribute paths ... 49
2.8.12 Line strings ... 50
2.8.12.1 Geometry of the line string ... 50
2.8.12.2 Line strings with straight line segments and circle arcs as predefined curve segments ... 52
2.8.12.2.1 Other forms of curve segments ... 55
2.8.13 Surfaces and tessellations ... 55
2.8.13.1 Geometry of surfaces ... 55
2.8.13.2 Surfaces ... 58
2.8.13.3 Surfaces of a tessellation ... 58
2.8.13.4 Extensibility ... 59
2.9 Units ... 59
2.9.1 Base units ... 59
2.9.2 Derived units ... 60
2.9.3 Combined units ... 60

2.10 Dealing with meta objects ... 61
2.10.1 General comments concerning meta objects ... 61
2.10.2 Parameters ... 62
2.10.2.1 Parameters for reference and coordinate systems ... 62
2.10.2.2 Parameters of symbols ... 62
2.10.3 Reference systems ... 62
2.11 Run time parameters ... 63
2.12 Constraints ... 63
2.13 Expressions ... 65
2.14 Functions ... 69
2.15 Views ... 71
2.16 Graphic descriptions ... 75
3 Sequential transfer ... 80
3.1 Introduction ... 80
3.2 General rules for the sequential transfer ... 80
3.2.1 Derivation from the data model ... 80
3.2.2 Organization of a transfer: Preliminaries ... 80
3.2.3 Transferable objects ... 80
3.2.4 Order of objects within the data domain ... 81
3.2.5 Coding of objects ... 81
3.2.6 Transfer-types ... 81
3.2.7 Normative references ... 82
3.3 XML-coding ... 82
3.3.1 Introduction ... 82
3.3.2 Symbol coding ... 83
3.3.3 General structure of a transfer file ... 83
3.3.4 Header section ... 84
3.3.5 Data section ... 84
3.3.6 Coding of topics ... 85
3.3.7 Coding of classes and relationships ... 85
3.3.8 Coding of views ... 86
3.3.9 Coding of relationships ... 86
3.3.9.1 Embedded relationships ... 87
3.3.9.2 Non-embedded relationships ... 87
3.3.10 Coding of graphic definitions ... 88
3.3.11 Coding of attributes ... 88
3.3.11.1 General rules for the coding of attributes ... 88
3.3.11.2 Coding of strings ... 88

3.3.11.3 Coding of enumerations 88
3.3.11.4 Coding of numeric data types 89
3.3.11.5 Coding of formatted domains 89
3.3.11.6 Coding of date 89
3.3.11.7 Coding of time 89
3.3.11.8 Coding of date and with time 89
3.3.11.9 Coding of blackboxes 89
3.3.11.10 Coding of class types 90
3.3.11.11 Coding of attribute path types 90
3.3.11.12 Coding of structure attributes 90
3.3.11.13 Coding of coordinates 90
3.3.11.14 Coding of line strings 91
3.3.11.15 Coding of surfaces and tessellations 91
3.3.11.16 Coding of references 92
3.3.11.17 Coding of meta objects 92
3.3.11.18 Coding of the OIDType 93
3.4 Derivation of an XML-schema from a data model 93
3.4.1 Introduction 93
3.4.2 Type declarations 93
3.4.3 Simple types (SimpleTypeRestriction) 94
3.4.3.1 Numeric types 94
3.4.3.2 Text 94
3.4.3.3 MText 94
3.4.3.4 OIDType, formatted domains, class types, attribute paths 94
3.4.3.5 XML-blackbox 95
3.4.3.6 Binary blackbox 95
3.4.3.7 Date 95
3.4.3.8 Time 95
3.4.3.9 Date including time 95
3.4.3.10 Enumerations 95
3.4.4 Complex types (ComplexTypeContent) 95
3.4.4.1 Coordinates 95
3.4.4.2 Multi-coordinates 95
3.4.4.3 Polylines 96
3.4.4.4 Multi-polylines 96
3.4.4.5 Surfaces 96

3.4.4.6 Multi-surfaces 96
3.4.4.7 Roles 96
3.4.4.8 Reference attributes 96
3.4.5 Classes, structures and relationships 96
3.4.6 BasketDef 98
4 Disclaimer/Reference to third party rights 99
5 Copyrights 99
Annex A (normative) – The internal INTERLIS-data model 100
Annex B (normative) – The XML-schema for the model INTERLIS 104
Annex C (normative) – The XML-schema for geometry-types 110
Annex D (normative for CH) – Symbol table 113
Annex E (informative) – A small example Roads 120
Annex F (standard extension suggestion) – Organization of object identifiers (OID) 143
Annex G (standard extension suggestion) – Uniqueness of user keys 146
Annex H (standard extension suggestion) – Definition of units 149
Annex I (standard extension suggestion) – Time definitions 151
Annex J (standard extension suggestion) – Color definitions 154
Annex K (standard extension suggestion) – Coordinate systems and coordinate reference systems 161
Annex L (standard extension suggestion) – Symbology models 177
Annex M (informative) – Glossary 184
Anhang N (informativ) – Index 207
Annex O – References & Bibliography 211
Annex P – Cooperation & Verification 211
Annex Q – Changes in comparison to the previous version 211
Annex R – Table of figures 212

Please note:

In the interest of improved legibility and comprehensibility, the masculine form of persons is used in the underlying document only. This wording explicitly includes women in their respective function.

Introduction

Status

Approved: This document was approved by the Experts' Committee. It has normative power for the defined field of application in the determined scope of application.

Preface

This reference manual has been conceived for experts concerned with information systems, especially geo-information-systems or land-information-systems. Above all it ought to be of interest to authorities to whom careful dealing with data is an important issue.

In 1991, "INTERLIS – A Data Exchange Mechanism for Land-Information-Systems" was first published. This mechanism consists of a conceptual description language and a sequential transfer format which in particular takes into account space related data (shortly geodata), thus permitting compatibility among various systems and long-term availability, i.e. depositing in archives and documentation of data. Making use of INTERLIS when deciding, planning or administering processes may yield great profit. Very often – e.g. through multiple application and uniform output of documented and verified data – major economies can be achieved.

Five years after its publication, INTERLIS - in retrospect called version 1, resp. INTERLIS 1 - has come out of its "Sleeping Beauty existence". In the meantime a considerable range of software tools has become available to the user, making it possible to process geodata described and coded in INTERLIS. INTERLIS has been created out of the requirements of Cadastral Surveying, but its range of applications is considerably wider, as shown by more than a hundred data models and projects which work with INTERLIS ten years after its publication. The standard "INTERLIS version 1" in its form of Swiss Norm SN 612030 will remain of use for some time yet – in parallel with its successor versions.

In order to meet increased demands of our users, several extensions to INTERLIS 1 have become necessary, e.g. incremental re-export, structural object orientation or formal description of graphic illustration of objects. 1998 saw the beginning of a process which was to last several years and involved the joint efforts of half a dozen experts in research, administration, counseling and software industry. Its result is a product that may be called an extension to INTERLIS 1 and at the same time a synthesis of all the latest concepts.

In the INTERLIS Version 2-reference manual we have striven to lay down only the absolute necessities; examples and figures only appear where they may complement the concise text. In this way the specification is clearly arranged and easy to implement. If some language elements, such as views and graphic descriptions seem ambitious and demanding, this is in all likelihood not due to INTERLIS itself, but to the complexity of the field. To solve this problem we rely on the following methods: good example, basic and continued education as well as so-called "profiles", i.e. sub-quantities of well-defined INTERLIS tool capacities.

For a general understanding of INTERLIS 2-concepts I should like to advise the reader to peruse at least chapter 1 Basic Principles.

Extensions of INTERLIS 2.4 in comparison with INTERLIS 2.3

In the interest of reliablity and investment protection INTERLIS 2.3 has remained virtually unchanged over a considerably long period. However, in the meantime improvements were suggested by experts working in data modelling and data exchange. Thanks to this present version - INTERLIS 2.4 - requests that are of utmost interest and importance in everyday work-life should be realizable with a minimum of effort.

This will affect above all the entire domain of data transfer. As compared to INTERLIS 2.3, the user will find that transfer-files according to INTERLIS 2.4 have been greatly simplified and adapted to the practices of XML-proceedings (XML Best Practice). Major keywords are:

- Precise definition of symbol coding,
- Rules governing the handling of XML namespaces,
- No more alias-tables in header-sections, as the originally aspired concept of polymorph reading could not establish itself,
- In most cases the class name is sufficent for object tags, thus rendering tags much shorter and therefore transfer-files smaller and clearer,
- Simplified processing of sub-sets (LIST, BAG).

Furthermore, the present norm also includes rules for the derivation of XML-schemas (XSD) from INTERLIS models.

The greater number of adaptions in the modelling language mainly concern details:

- Consistency restraints (constraints) can be named and thus clearly identified.
- Constraints can be reformulated as domains.
- Uniqueness restraints can be limited to one specific transfer.
- Implication to simplify expressions in constraints.
- MULTI-geometries improve compatibility with OGC, resulting in a simplification especially with "exclave"-problems.
- Simplified dealings with time.
- Sub-sets (LIST and BAG) are also possible with primitive types (rather than with structures only).
- Line attributes are eliminated.

There is a new possibility of creating generic coordinate domains which should be of particular significance as it will permit the definition of models without determining their concrete coordinates (notably whether they are NS03 or NS95 coordinates. Their final determination will take place in specific models or even as late as in the transfer data.

A further modification whereby the declaring of models as "CONTRACTED" is no longer of consequence, will hardly have an impact on everyday work-life. As the concept of contracts was never really put into practice, it was now abandoned.

From the language point of view all modifications were carried out in such a way that valid INTERLIS 2.3-models will become valid INTERLIS 2.4-models simply by changing the version number to 2.4. However, in the case of existing models it may prove sensible to adapt them accordingly while making use of these various new features.

Extensions in INTERLIS 2 compared with INTERLIS 1

With some few exceptions the existing description language INTERLIS 1 has only been complemented and not altered. Thus we have extended the possibilities to describe relationships between objects (actual relationships as association class and reference attributes with REFERENCE TO. Note: The "->"-syntax of the INTERLIS 1 relationship attribute has been given a different meaning) while taking into account that the transition from INTERLIS 1 to 2 should be rendered as easy as possible (cf. chapter 2.7 Proper relationships). Henceforth proper relationships and reference attributes can, certain conditions applied, refer to objects in other baskets. Basket is a new term for the well-known organization of objects (i.e. data who describe reality, also called object instances or instances) in a database. We have revised the term TABLE that has become CLASS in accordance with the changeover from relational to object-oriented formalism. Without further specification any attribute is considered optional (OPTIONAL is omitted) and it has to be indicated as MANDATORY. Furthermore, the uniqueness key word IDENT has been renamed UNIQUE. The new object-oriented concepts include transmission amongst others of topics, classes, views, graphic descriptions and attribute domains. Other extensions of great importance are set data types (LIST, BAG), constraints, data views, graphic descriptions, descriptions of units, description of meta objects (coordinate systems and graphic symbols) and incremental re-export. Furthermore, special user-specific extensions, such as functions and line geometries can be defined. However, this will make contracts, i.e. agreements with tool providers, necessary.

From now on the eXtensible Markup Language (XML) will take over coding for our INTERLIS 2 transfer format. We expect XML to become internationally widespread and universally accepted and count on a great number of compatible software products to be obtainable in the near future.

Any user well acquainted with INTERLIS 1 will not have to face many changes as long as he renounces the use of our new concepts, such as object orientation and graphic description: his present knowledge will be applicable in INTERLIS 2. Various tools, such as the commonly available INTERLIS 2 compiler, will facilitate adapting to the new version. Those producers who already had flexible configuration possibilities on their minds when implementing INTERLIS 1, as well as taking into account rules and art of software development (e.g. modularization and abstraction), will find that their past investments will remain of value. Thanks to universally accessible program libraries software manufacturers will be able to concentrate fully on the integration of their systems in INTERLIS 2.

Outlook

INTERLIS 1 appeared at a time when the relational data definition language SQL-92 had not yet been standardized and object orientation had not been talked of. It is thanks to Joseph Dorf-schmied - originator of INTERLS 1- acting with rare foresight, that some of these nowadays well established concepts already then were introduced. Now that INTERLIS has been revised and object-oriented and specific concepts of informatics have been included, it may be said to have achieved a new degree of maturity. Thus INTERLIS 2 may be used already today - and not only tomorrow – as an efficient tool.

Nevertheless we are well aware of the fact that even with INTERLIS Version 2 our quest for a universal data description language is nowhere near its end. However, any hesitation might lead to similar consequences as the shortsighted handling of the resources of our earth. INTERLIS deserves to be accepted not only as a data exchange format but also as an enduring tool: Thanks to INTERLIS the call for an enduring way to deal with technology has been given a name!

Wabern, March 2016

Rolf Zürcher

1 Basic principles

1.1 Overview

INTERLIS allows co-operation between information systems, especially geographic information systems or land information systems. As its name suggests, INTERLIS stands between (inter) **l**and **i**nformation **s**ystems. It is crucial that all systems involved have a very clear notion of these concepts that are of major importance to their co-operation.

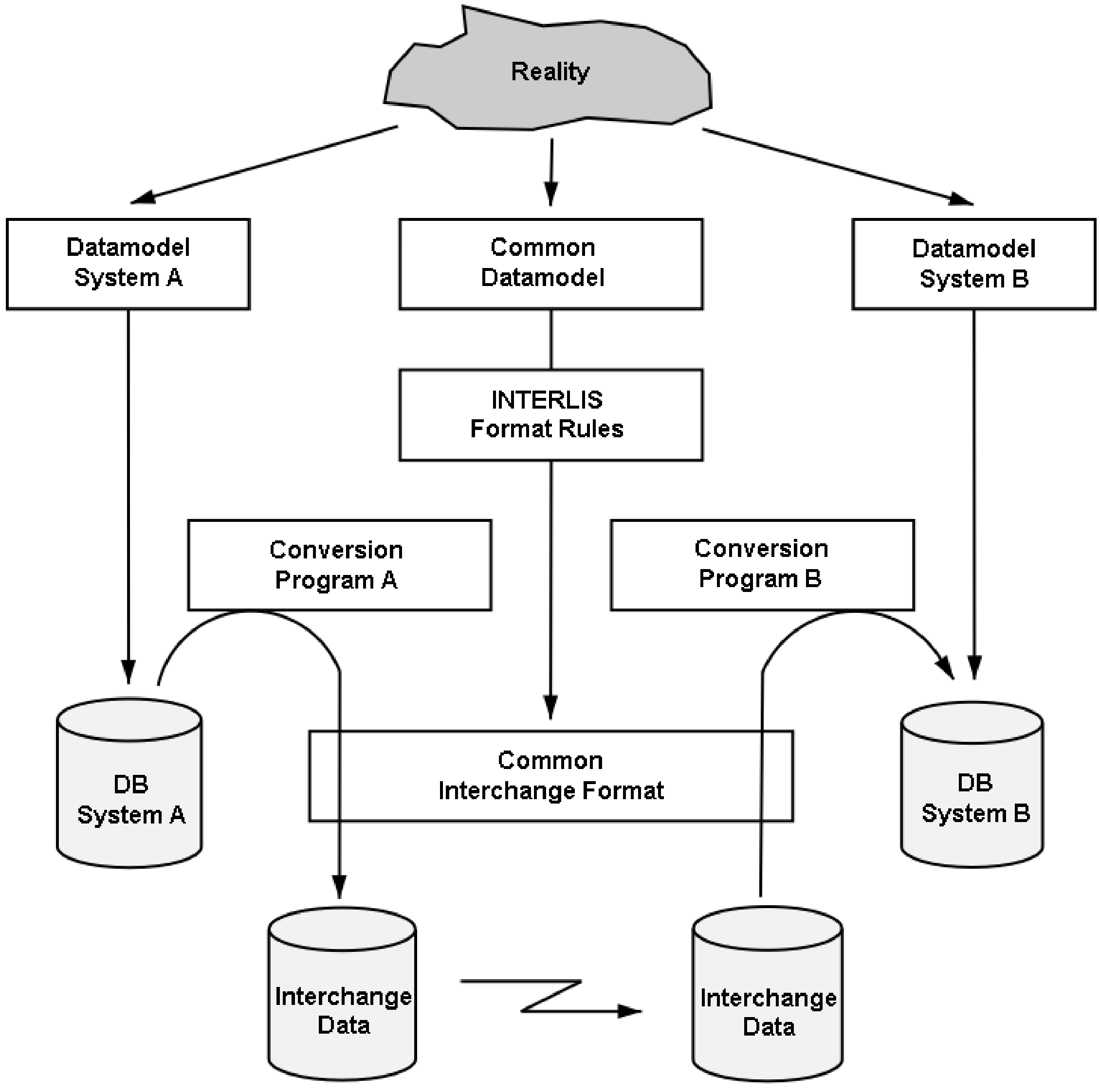


Figure 1: Data transfer between several databases via a common data model (data schema) described in a common data description language.

That is why INTERLIS comprises a conceptual description language. Thanks to this language a detail of reality may be described that is of interest for a certain application. Such a description is called (conceptual) application model or application schema, respectively in short model or schema. Some few concepts with strictly defined meaning will describe (i.e. model) classes of objects with their respective characteristics and relationships. Furthermore this INTERLIS description language permits the introduction of classes of derived objects, hence making it possible to define these as views on other classes of objects. True as well as derived classes of objects may serve as the basis of graphic descriptions, however INTERLIS assures a strict separation of graphic descriptions (representation definition) and description of the under-lying data structure (data definition).

INTERLIS does not aim at any specific application. The draft takes its bearings of common object-oriented principles, nevertheless we have tried to ascertain very good support of such concepts as are of importance to land information systems. Thus coordinates, lines, and surfaces are data types that represent basic constructs of INTERLIS and language elements describing precision of measurements and units are at your disposal. Still it is possible to use INTERLIS in non-geographic applications.

Aspects which are under-lying a field of application can be described in a basic model. Subsequently this model will be specialized according to the specific needs of a country, in further steps according to those of a certain area (county, region or community; see figure 2). INTERLIS 2 offers two object-oriented concepts as possible tools for this specialization: inheritance and polymorphism, thus assuring that already set definitions need not be repeated or accidentally be made doubtful.

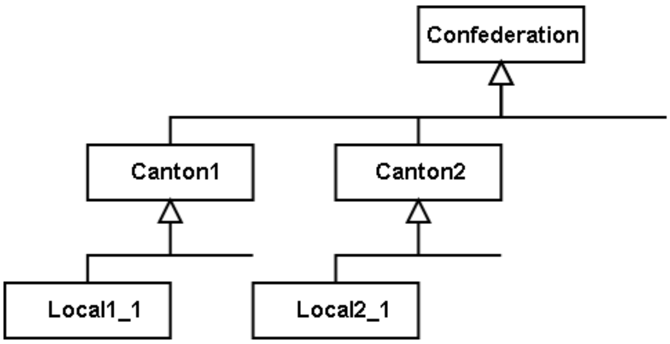


Figure 2: Specializing the modeling of a concept from the federal level, to cantonal (country specific) and local level.

Extensive applications need not be defined in one single description. On the contrary, they can be split up into several description units (models, schemas). A description unit may comprise several topics. In the interest of the readability of such models it is also possible to define a model as a simple translation of other models.

1.2 Utilization of models

In the first place an INTERLIS model (resp. INTERLIS schema) represents a means of communication for users. Its language is designed in such a way as to be readable by humans. Nevertheless INTERLIS models are precise, unequivocal and can be interpreted without any possible misunderstandings. Therefore the textual INTERLIS language offers itself as a necessary complement to the graphic description language Unified Modeling Language (UML, www.omg.org/uml).

But INTERLIS does not stop here: Since a model possesses a formal and clearly defined significance, it permits the implementation of a service in a computer system to be automatically derived from this model. For example INTERLIS comprises an XML-based transfer service, whose definitions are produced from respective models according to their rules. The utilization of data modeling in close connection with system neutral interface services is called *model-based* or *model-driven architecture* (see "Model-Driven Architecture" by OMG, www.omg.org/mda/).

Models may be built upon common basic concepts. Since INTERLIS allows for an explicit description through use of inheritance and polymorphism, it is obvious at any given moment, which concepts are common and which are specific. In view of aspiring to a semantic interoperability this becomes of great importance. For example, it offers the possibility for a transfer file of any community (town, village) to be interpreted by its superior administration unit (county, state), without demanding a common model agreed upon by all participants. It is sufficient that every level builds its own model upon that employed by its superior unit.

It is conceivable and moreover desirable that new services based upon INTERLIS should be developed, a task greatly facilitated by using a model compiler. This program reads and writes INTERLIS models, permits necessary changes within and examines, whether models are in accordance with syntactic and semantic conditions of INTERLIS. Amongst others this compiler can generate automatically – in accordance with the present INTERLIS transfer service with XML – XML schema documents (www.w3org./XML/Schema) derived from INTERLIS models. By introducing adequate general XML-tools it is possible to render the concrete INTERLIS/XML-files available for an even wider range of application. As long as conditions for usage are not violated, this INTERLIS compiler is available for the producing of new tools.

1.3 A structure of models and topics

A model (resp. a schema) describes an image of our world as it may be of significance for a specific application. A model is a self-contained unit, which may also use or extend parts of other models. To some extent an INTERLIS model can be compared to modules or packages of some programming languages.

Primarily we distinguish between models which contain only type definitions (units, value ranges, structures) and others where data may exist. Besides their name models also contain information regarding editor and version. All descriptions with existing data are divided into topics. This division is a result of our conception in what organization units and by whom such data should be controlled or used. If as a typical feature data is controlled and used by several authorities, it should be defined in various topics. Such interdependences ought to be limited to a strict minimum. Relationships between topics whose data are controlled by several authorities ought to be omitted wherever possible, as special efforts to maintain consistence are inevitable. In any case cyclic dependence is excluded. Besides data definitions as such, topics may comprise also definitions for views and graphics.

One topic may enhance another. In this way all concepts defined by the basic topic are transmitted and can be complemented.

For example a model of some country's Cadastral Surveying could comprise the topics "addresses" and "buildings" (see figure 3). These concepts are independent; any relationship is established algorithmically by way of coordinates. Individuals possess addresses; hence the person-model of this country is built upon the national model of Cadastral Surveying, whereby the topic "Persons" depends on the topic "Addresses" of the cadastre model.

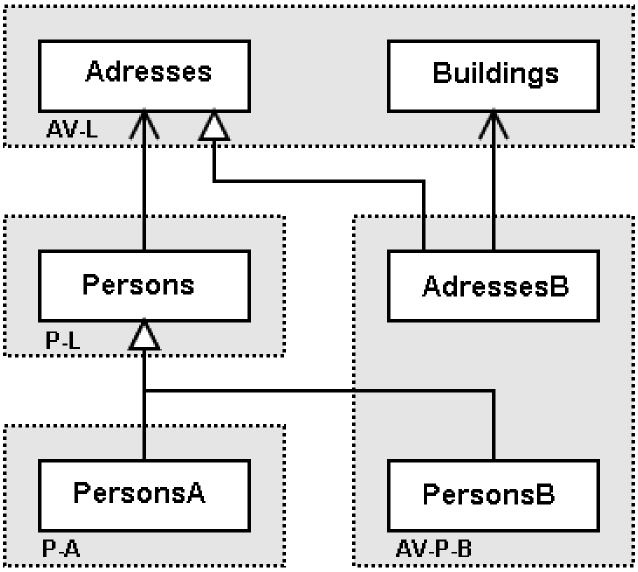


Figure 3: Inheritance hierarchy of addresses, persons and buildings.

Now it is planned to describe the inhabitants of area A more precisely than intended by the national person-model. So area A drafts its own model, transferring the topic "persons" from the nation-wide person-model and extending it.

In area B it is planned to establish an explicit relationship between buildings and addresses on one hand, on the other the national person-model is considered inaccurate. Again the respective topics are transferred and specialized. Both extensions are combined in one single model – the "global view" of area B.

1.4 Object concept

1.4.1 Objects and classes

An object (otherwise called object instance or simply instance) consists of data concerning one real-world item and can be unequivocally identified. Obviously numerous objects possess similar characteristics and hence can be summarized. Such a set of objects (object set) with similar characteristics is called a class. Each characteristic has at least one corresponding attribute. In INTERLIS 1 we used the term table instead of the term class. Other expressions meaning class are: set of entities, type of entities, feature.

When describing a class we record, amongst other information, the qualities and characteristics each object possesses. These are called attributes. Attribute values of objects cannot be chosen at random, but must comply with certain conditions stipulated by the description of an attribute.

With regard to this, INTERLIS offers a series of basic data types (base data types, strings, numeric data type, enumerations, Cartesian and elliptic 2D, resp. 3D coordinates), based upon which new and more complex data structures can be defined. In order to render this statement more precise yet, numeric attributes can be further supplied with a measuring unit and related to a reference system, coordinates can be referred to a coordinate reference system.

In general date and time are applications of formatted value domains. Since indications of time are quite common, an individual structure has been defined for date and time (XMLTime, XMLDate and XMLDateTime).

Besides these basic data types, an attribute may also comprise sub structures. Each of these elements of a sub structure is to be treated as structural element that can only exist in relation to its main object and cannot be found in any other way than by passing by this main object. The organization of structural elements is described in a fashion similar to that of classes.

Besides the fact that all attribute values must correspond to their respective type, further conditions can be defined. INTERLIS distinguishes between the following conditions of constraint:

- Constraints that refer to one single object. These constraints are subdivided into requirements that must be met by each object of a class ("hard" constraints) and regulations, which in rare cases can be violated ("soft" constraints).
- Constraints which demand the clearness of attribute combinations of all objects in one class
- Constraints which demand the existence of an attribute value in an instance of a different class
- More complicated conditions, which refer to an object set and have to be defined by means of views.

1.4.2 Extension of class

Classes are either autonomous or extend (specialize, inherit) a super class, i.e. the description of a class is either independent or contains extensions of another inherited description. An extension of class (also called subclass) can provide further attributes and constraints, as well as heightening already accepted conditions (data types, constraints).

Each single object belongs to exactly one class (in other words it is object instance or instance of a class). At the same time it always meets with all the requirements of its super classes, i.e. its super classes. Therefore there can be found, corresponding to each class, a set of objects, which are instances of the class itself, or one of its extensions. In the case of concrete classes we usually find a smaller subset of instances, which belongs exactly to this class.

Elements of sub-structures are no independent objects, but structural elements and therefore do not form part of the set of instances of any given class.

1.4.3 Meta models and meta objects

As seen from the user's point of view, coordinate systems or coordinate reference systems as well as graphic symbols, are represented as model elements (resp. schema elements) which can be used in application definitions. Since different coordinate systems and coordinate reference systems and above all different graphic symbols can be described in the same way, it makes sense to also describe their characteristics within models by means of classes. Thus every system, resp. every graphic symbol (e.g. a point symbol, a line type) corresponds to an object.

Meta objects must be defined in a meta model (cf. chapter 2.10.1 General comments concerning meta objects). Applicable objects must explicitly be identified as meta objects (extensions of the predefined class meta objects) and consequently can be referenced via their names. To achieve this, it is necessary that meta objects be available to the tool, which treats the application definition by means of baskets (cf. chapter 1.4.5 Baskets, replication and data transfer).

1.4.4 Relationships between objects

INTERLIS 2 (as opposed to INTERLIS 1) distinguishes between two types of relationships between objects: proper relationships and reference attribute.

The term relationship refers to a set of object-pairs (resp. in general object-n-tuples). The first object of each pair belongs to a first class A, the second to a second class B. Since the attribution of objects to pairs is to be predefined, it must only be described, i.e. modeled. As shown further along in chapter 1.5 View concept, it is however also possible to compute this attribution algorithmically, e.g. based upon attribute values.

Proper relationships are described as independent constructs, so-called relationship classes (resp. association classes) that again are extendable. INTERLIS 2 does not only support one-to-one relationships, but also permits multiple relationships and relationships with their own attributes. Thus a relationship class in return is also an object-class.

Important characteristics of such relationships are:

- *Cardinality* – how many objects of class B (resp. class A) can be ascribed to one object of class A (resp. B) within the relationship?
- *Force* – INTERLIS 2 differentiates between associations, aggregations and composition. In all cases the objects in question can be applied to independently. In the case of aggregation and association the objects exist independently of each other. In the case of aggregation and composition we find asymmetry between the classes concerned. The objects of one class (the super class) are described as entities (resp. super-objects), the objects of he other class (subclass) are parts (resp. sub-objects). In associations all objects are equal and only loosely connected. Both aggregation and composition are conceptually directed associations: An entity (super-object of the super class) is ascribed several parts (sub-objects of the subclass). When dealing with an aggregation all ascribed parts are automatically copied when copying the entity, however when deleting the entity the corresponding parts remain untouched. Compared to an aggregation you will find that a composition further implies, that when deleting the entity all parts are deleted at the same time. How copying effects relationships to other objects is described in chapter 2.7.2 Force of relationship. Note: Sub-objects (parts) of compositions are identifiable objects as opposed to structural elements of sub-structures.
- *Role* - What significance have the classes involved see from the viewpoint of the relationship? This is determined for each class involved by means of its role. INTERLIS 2 (as opposed to INTERLIS 1) also admits relationships exceeding the limits of one subject. This however on condition that the relationship attribute is defined within such a class as belongs to a subject depending on the class it is referred to.

By using a *reference attribute* we create a relationship between one object, resp. structural element to another object. However such a relationship is only known to the referring object and not the object referred to. Hence it is onesided.

Without violating the independence of topics it is possible to also define relationships (i.e. proper relationships and reference attributes) by means of a special mark (EXTERNAL), these will create a relationship with objects of *a different basket* of the same or a different topic. However only on condition that the structure within which the relationship is defined belongs to a topic which depends on the topic whose class it is referring to.

In the interest of a clear structure, relationships can only refer to classes already known when defining relationship classes (resp. reference attributes).

1.4.5 Baskets, replication and data transfer

A basket is a compact collection of objects, which form part of a topic or its extension. Compact means that a basket contains all objects related to each other within one topic. A typical example may be a certain area (town, county or even a country as a whole) whose objects in their entirety are contained in a basket. Above all a basket may contain data supplied by various extensions (e.g. different cantons with their own topic extensions), provided that within such a transfer community all labels of models, topics and topic extensions are unequivocal.

Within the scope of constraints we often speak of "all" objects of a class. Basically this includes all objects that have the quality desired and actually exist, i.e. basket-spanning. For various reasons (efficiency, availability, access rights etc.) it is obvious that control is only possible within locally accessible baskets. In the case of baskets with meta objects we state explicitly that constraints only apply within a basket, since we presume that meta data are of descriptive character and have to be at the user's free disposal in a basket (much in the way of a library).

- We distinguish different types of baskets: data-baskets – comprises all instances of classes of one topic
- View-baskets: comprises all instances of views of one topic
- Baskets with basic data for graphics – comprises instances of all data or views necessary for the graphics of one topic. Permits use of graphic application software.
- Baskets with graphic elements – comprises the instances of all graphic objects (= symbols), which are necessary according to the graphics of one topic. Permits use of a graphic representation software (renderer; see figure 5).

INTERLIS does not set rules on how objects must be kept within their systems. Rules only apply to the intersection between systems. At present an intersection for the transfer of baskets as XML-files is defined. Support is not only granted to the complete transfer of the entire basket, but also to the incremental data transfer. We proceed on the assumption that in the case of complete transfer the receiver will create new, independent object copies without any immediate connection with the original object. Within the scope of complete transfer, objects must only be marked with a temporary transfer identification (TID). TID's are used for transferring relationships.

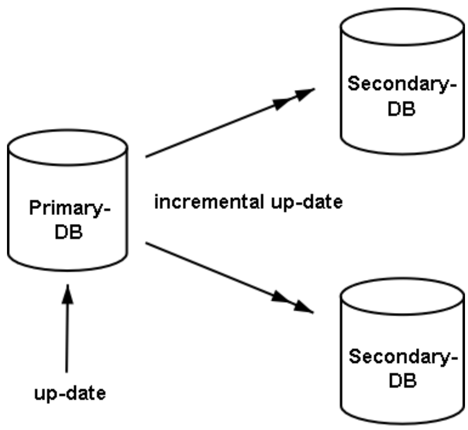


Figure 4: Up-dating of a primary database and subsequent transfer to secondary databases (a double arrow means incremental update).

In the case of incremental transfer we presume that to start with the sender supplies the initial state of a data-basket (all other types of baskets are excluded) and subsequently provides updates, which will allow actualizing the data received (see figure 4). Thus the objects are replicated and keep their connection with the original object, i.e. they cannot be altered independently of the original and will not be given a new identification.

We proceed on the assumption that their exist contracts between the management of primary and secondary databases (extent, frequency of updates etc.) which at present cannot be covered by tools provided by INTERLIS. For the update itself INTERLIS places the necessary tools at your disposal. New objects are displayed in the same way as when first making a transfer and they are assigned an unequivocal object identification (OID) – which must be up-kept at all times. Whenever changes occur, this unequivocal OID is used as a point of reference and all attributes of the object (including all structural elements of structure attributes) are updated. In the same way deleted objects are brought to your notice. It is primarily the sender's responsibility to assure the consistency of objects (e.g. observance of constraints, correctness and cardinality of relationships). To that end he notifies the receiver of any alterations: objects deleted, updated, newly created. In the interest of independence of topics, we do not assume that integrity is guaranteed at all times where several topics are concerned. It is up to the receiver to cope with temporary inconsistencies between base topics and dependent topics, i.e. that an object referred to does not exist.

INTERLIS does not determine the limits within which clearness of OID is guaranteed. Cf. appendix F *Organization of object identifiers (OID)* for an example of how such an OID may be structured – yet other possibilities definitely are conceivable. However it is of primordial importance that all data recorders of one transfer community correctly apply the rules governing the structure of an OID, thus assuring that at all times the OID remains unequivocal within the scope of the transfer community. Depending on the structure of the OID incremental data exchange is possible within a wider (e.g. world-wide) or smaller (e.g. within one organization) circle. Hence the method used to determine an OID also defines the potential transfer community.

Any alteration on an object other than within its original basket strictly depends on the permission of its administration. All other secondary baskets can alter an object only as a result of an update. That is why INTERLIS demands – within the limits of incremental update - that not only objects but also baskets must be identifiable in an unequivocal and permanent manner. Then an OID is also assigned to baskets. Likewise in the case of a complete transfer the basket only needs a transfer identification (TID). Whenever it is important to clearly distinguish between OID and OID of a basket, we speak of BOID (resp. basket identification BID).

We must assume that to start with various objects are registered in baskets, e.g. of one town, then these baskets as a whole are transmitted to the canton and subsequently will be integrated into baskets which contain topic-wise the entire canton. Maybe these baskets will then be further transferred, e.g. to state authorities. In order to have at any moment definite assurance as to the original basket, its BOID is supplied with each replica of an object. This will allow the receiver to create his own basket-administration by stating in which of his own baskets are stored replicated objects supplied by which original basket. (INTERLIS also provides the necessary tools to label such baskets with INTERLIS itself and thus permits their exchange much in the way of normal objects). It is one of the characteristics of INTERLIS 2 that when dealing with relationships concerning several topics not only the OID's of the reference object but also the BOID's of its original baskets are transferred. If the receiver makes full use of this INTERLIS 2-feature which in the case of topic-spanning relationships allows not only to transfer the OID of reference objects but also the BOID of their original baskets, he may determine in an efficient way in which of his baskets the reference object is to be found.

1.5 View concept

INTERLIS 2 enables the user not only to model objects themselves but also views. Views really are virtual classes whose instances are not data originated from a real object, but data derived mathematically from other objects.

A view definition consists of the following parts:

– Base sets – Which classes resp. views supply the objects that are introduced into the calculation of view objects? In the case of classes not only the respective instances are taken into consideration, but in the sense of polymorphism all instances of extensions as well. INTERLIS does not define the baskets that have to be taken into account by a system when calculating a view.

- *Interrelationship rules* – INTERLIS 2 distinguishes joins, unions, aggregations and inspections of sub-structures. In the terms of set theory, joins represent cross over products and unions the combination of base sets. Aggregation allows you to combine elements of the base set in a new view object, provided they fulfill definable criteria. Inspection of sub-structures allows you to view the structural elements of a sub-structure as a set of structural elements. Joins and aggregations may also be conceived as virtual associations.
- *Selection* – Which of the calculated objects should actually form part of the view? INTERLIS allows you to define complex conditions concerning this aspect.

1.6 Graphic concept

Graphic descriptions are based upon classes or views – using a certain projection - and declare – in so-called graphic definitions (see figure 5 and appendix M *Glossary*) which graphic symbols (e.g. point symbol, line, area symbol or text label) are to be assigned to the objects of a view, thus permitting the graphic user to create presentable graphic objects. The graphic symbols themselves have been defined in an extra symbology model – symbol characteristics can be found there as well.

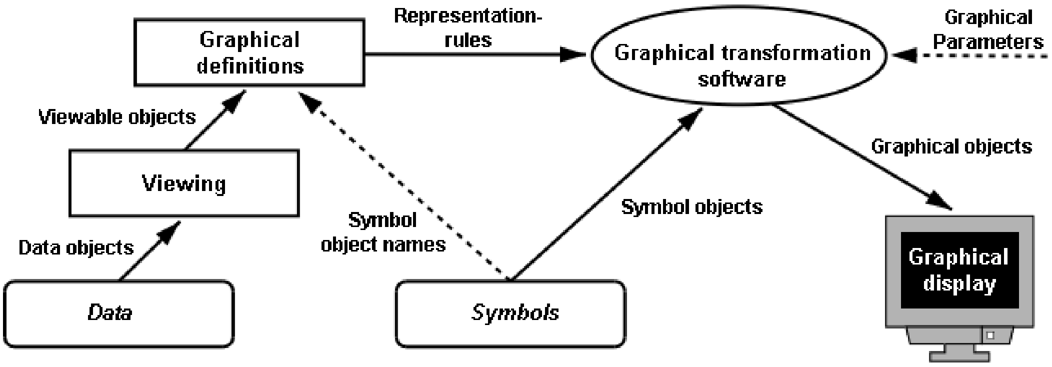


Figure 5: Graphic definitions, on one hand built upon data and views, on the other upon symbols permitting generation of graphics (abstract diagram).

The reference to the graphic symbols in question ensues by using names; see symbol object names in figure 5. The graphic symbols as such (also named symbol objects) are contained like objects (data) in their respective meta objects-baskets. A basket with such symbol objects is also known as a symbol library.

A symbology model determines for every symbol type within its respective symbol class, which additional parameters (e.g. position and orientation of a symbol) are necessary for its representation. Thus the intersection with the graphic-subsystem (graphic application software) of the respective systems is not determined by INTERLIS itself, but by the symbology models. To this extent it is possible at any moment to declare further global parameters (e.g. scale of representation), at the time are known to the graphic sub-system and able to influence the decision, which symbols are to be used in the representation.

A graphic description need not regulate the conversion into symbols in a final way. On the contrary, it can be inherited from a different graphic description. It is then possible to supplement parameters left undefined in base definitions, or existing directions can be replaced.

1.7 Services, tool capacities and conformity

INTERLIS 2 permits a conceptual description of data and defines a system-neutral transfer. INTERLIS 2 explicitly renounces all directions concerning implementation in order to remain system-independent. Thus often in practical operation the question will arise whether a certain tool or a required service is in conformity with INTERLIS or not.

INTERLIS does not stipulate that only the extremities of complete conformance or non-conformance are conceivable. On the contrary one service will conform to INTERLIS in some aspects while not fulfilling other requirements.

In the simplest of cases a certain system meets the INTERLIS specifications in one particular case (for a fixed set of models, only read or only write or both, etc.).

Ideally a set of INTERLIS models can be handed over to an INTERLIS tool, whereupon the tool automatically adapts its services and capacities in accordance with the situation defined by the models. In many cases however this adaptation will demand further manual efforts such as system configuration or even programming. No doubt an essential requirement in such tools is their capacity to read INTERLIS model descriptions correctly. Above all this means that system abilities are offered correctly according to INTERLIS constructs, especially inheritance constructs.

From the standpoint of INTERLIS, the capacities of such tools may be classified as follows (so-called tool capabilities, functionalities or services):

- Read data (including views memorized, no generating of view-objects)
- Read data (including views memorized plus generating of view-objects)
- Examine consistencies
- Write views
- Produce graphics (including the reading of views)
- Treat and write data
- Produce object identifiers (OID)
- Read updates (incremental updates)
- Write updates (incremental updates)

It is quite conceivable that a certain tool or some service will have certain capacities (e.g. incremental update) for one model or topic (data or views), but will not support them in other models or topics.

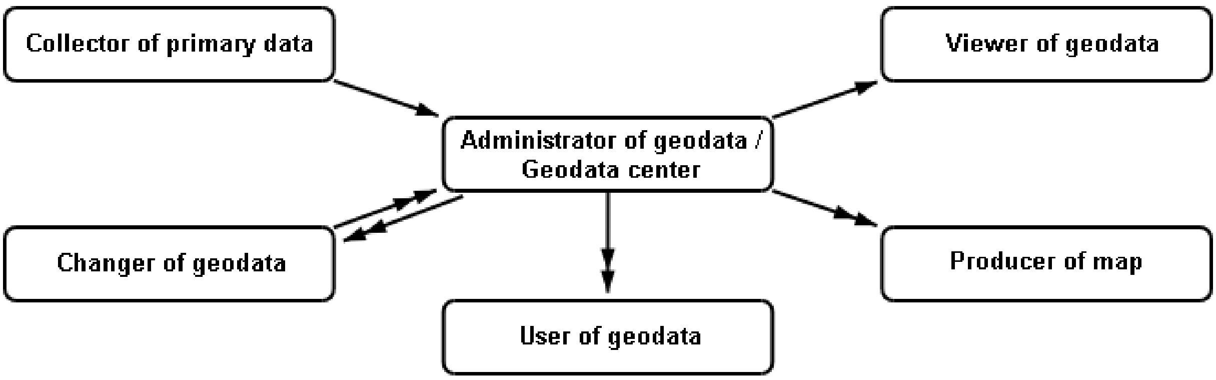


Figure 6: The various ranges of application of INTERLIS. A double arrow means that data can be incrementally transferred.

When observing an example of joint efforts of various parties concerned, it becomes obvious that different INTERLIS 2 tool capabilities or services will be in demand – depending on the operational area and role of the user (see figure 6). Within one application (i.e. INTERLIS model) with different topics (TOPICS) one single user may have various roles:

- *Collector of primary data* – Works with and writes data, produces OID, if the occasion should arise.
- *Changer of geodata (update, supplement)* – Collects data, writes and works with data, produces OID, produces increments, reads increments, examines consistencies (locally).
- *Administrator of geodata, geodata center* – Collects data, writes and works with data, produces OID, reads increments, examines consistencies (globally).
- *User of geodata* – Collects data, reads increments.
- *Viewer of geodata* – Collects data, produces views objects and graphic representations.
- *Producer of map* – Collects data, reads increments; reads views and produces graphic representations.

1.8 A small example as an introduction

Appendix E *A small example Roads* supplies a small example that presents the most important elements of INTERLIS within the limits of a simple application.

Figure 7: Roads - a small example.

1.9 How this document is structured

This reference manual is subdivided as follows: In chapter 2 our description language is formally defined. In chapter 3 the sequential transfer of geodata is specified. This chapter is divided into a more general part concerned with the present and future sequential transfer services of INTERLIS 2, and a more specific part regarding the INTERLIS 2 transfer service with XML.

The appendices are arranged in four *normative* appendices A through to D, complemented by an *informative* appendix E, several *standard extension suggestions* F through to L, as well as a glossary (appendix M) and an index (appendix N).

The normative appendices are integrating part of this specification. We strongly recommend the standard extension suggestions (appendices F through to L) for implementation. These are informative (non-normative) appendices, however of autonomous character. Implementations may find different answers to these questions; they will still remain INTERLIS 2 conformant. In such a case it is up to the parties involved in the transfer to reach an agreement concerning the specification.

2 Description language

2.1 Syntax applied

In order to determine a conceptual data model (data schema) and transfer parameters of a data transfer, a formal language will be defined in the following chapters. This language itself has been formally defined. Therein rules of syntax set the admissible sequence of symbols.

Thus this description is analogous to the ones generally employed in modern programming languages. Hence we only give you the briefest of outlines needed for basic understanding and advise you to consult technical literature for further details (e.g. a short introduction can be found in "Programming in Modula-2 by Niklaus Wirth).

In the sense of the extended Backus-Naur-Notation (EBNF) a formula is structured as follows:

```
Formula-Name = Formula-Expression.
```

This formula-expression is a combination of:

– Fixed words (including special characters) of the language. They are enclosed in apostrophes, e.g. 'BEGIN'

– References of other formulas by indicating the formula-name.

Valid combinations are the following

Sequential composition

```
a b c          first a, then b, then c.
```

Grouping

```
( a )          round brackets group formula-expressions.
```

Choice

```
a | b | c      a, b or c.
```

Option

```
[ a ]          a or nothing (void).
```

Optional repetition

```
{ a }          any chosen sequence of a or nothing (void).
```

Obligatory repetition (as supplement to EBNF)

```
(* a *)        any chosen sequence of a, minimum one.
```

Examples:

```
(a|b)(c|d)     ac, ad, bc or bd
a[b]c          abc or ac
a{ba}          a, aba, ababa, abababa, ...
{a|b}c         c, ac, bc, aac, abc, bbc, bac, ...
a(*b*)         ab, abb, abbb, abbbb, ...
(*ab|[c]d*)    ab, d, cd, abd, dab, cdab, ababddd, cdababcddcd, ...
```

Often one would like to use a syntactically identical formula in different contexts, for different purposes. In order to express this correlation, an additional formula would have to be written:

```
Example = 'CLASS' Classname '=' Classdef.
Classname = Name.
```

In order to avoid this detour, we rather use the following shortened notation:

```
Example = 'CLASS' Class-Name '=' Classdef.
```

The formula Class-Name is not defined. Syntactically the rule "Name" is directly applied (cf. chapter 2.2.2 Names). Considering its meaning, name however is a class-name. Thus "class" virtually becomes a comment.

2.2 Basic symbols of the language

Our description language features the following classes of symbols: Names, strings, figures, explanations, special symbols, reserved words and comments.

2.2.1 Character codes utilized, blanks and line ends

The language itself only uses the printable US-ASCII symbols (32 to 126). What other symbols besides the blank space are considered as interspaced, has to be determined by the actual compiler implementation. It is also up to this implementation to determine what symbols or combination of symbols mark the end of a line. Likewise it is the compiler implementation that determines the memorization of symbols (character set). This might differ depending on the platform used.

As far as comments are concerned, it is also admissible to apply further symbols such as umlauts and accent marks, etc.

2.2.2 Names

A name is defined as a sequence of a maximum of 256 letters, digits and underlines, wherein the first symbol has to be a letter. We distinguish between capitals and small letters. Names that coincide with reserved words of this language (cf. chapter 2.2.7 Special symbols and reserved words) are inadmissible.

Syntax rules:

```
Name = Letter { Letter | Digit | '_' }.
Letter = ( 'A' | .. | 'Z' | 'a' | .. | 'z' ).
Digit = ( '0' | '1' | .. | '9' ).
HexDigit = ( Digit | 'A' | .. | 'F' | 'a' | .. | 'f' ).
```

Further information regarding uniqueness and validity domain of names is to be found in chapter 2.5.4 Namespaces.

2.2.3 Strings

Strings appear in connection with constants. They begin and end with quotation marks and may not exceed one line. \" Represents a quotation mark, \\ a backslash within a string.

A sequence of \u, immediately followed by exactly four hexadigits represents any chosen Unicode sign. Symbols beyond U+10000 are to be marked, as in the UTF-16-Coding, with two surrogate codes (see www.unicode.org).

Syntax rule:

```
String = '"' { <any character except '\' or '"'>
```

```
                  | '\"'
                  | '\\'
                  | '\u' HexDigit HexDigit HexDigit HexDigit
                  } '"'.
```

2.2.4 Digits

Digits may appear in different ways: positive whole numbers including 0 (PosNumber), numbers (Number), decimals (Dec) and structured numbers. With decimal numbers the scaling may be shown to the power of ten (e.g. 1E2 is 100, 1E-1 is 0.1). Structured numbers only make sense in relation with corresponding units and value ranges (e.g. time). It is only the value of a number that is of importance and not its representation, e.g. 007 is the same as 7.

Syntax rules:

```
PosNumber = (* Digit *).
Number = [ '+' | '-' ] PosNumber.
Dec = ( Number [ '.' PosNumber ] | Float ).
Float = [ '+' | '-' ] '0.' ( ( '1' | '2' | .. | '9' ) [ PosNumber ]
                            | (* '0' *) ) Scaling.
Scaling = ( 'e' | 'E' ) Number.
```

Examples:

```
PosNumber:    5134523            1          23
Number:           123         -435       +5769
Dec:          123.456    0.123456e4   -0.123e-2
Float:          0.1e7  -0.123456E+4  0.987e-100
```

2.2.5 Sets of properties

For several purposes it is necessary to assign properties to a subject matter. To this aim use the general syntax:

Syntax rule:

```
Properties = [ '(' Property { ',' Property } ')' ].
```

In order to define that at a certain place within a syntax rule such properties ought to be defined, the following construct is inserted into the syntax:

```
'Properties' '<' Property-Keyword { ',' Property-Keyword } '>'
```

Hence you write "Properties" and define in brackets (< and >) the admissible keywords. If you take for example the rule ClassDef (cf. chapter 2.5.3 Classes and structures), with "Properties<ABSTRACT, EXTENDED, FINAL>" the keywords "ABSTRACT", "EXTENDED", "FINAL" are accepted terms for the description of properties. Within an INTERLIS 2-Definition amongst others the following definitions would be possible:

```
CLASS A (ABSTRACT) = .....
CLASS A (EXTENDED, FINAL) = .....
```

2.2.6 Explanations

Explanations are required wherever circumstances have to be described more closely. From the viewpoint of the standard mechanism, this explanation will not be interpreted any further, i.e. it is considered a comment. But it is quite permissible to formalize explanations in more detail, so as to prepare them for further mechanical processing. We have chosen // to mark the beginning and the end of an explanation. Within an explanation two subsequent diagonal strokes may never occur.

Syntax rule:

```
Explanation = '//' any character except // '//'.
```

2.2.7 Special symbols and reserved words

In connection with the syntax rules of the language (however not where a data description is concerned), special symbols as well as reserved words always appear between apostrophes e.g. ',' or 'MODEL'. On principle, all reserved words are written in capitals. In order to avoid conflicts between names and reserved words, we advise you not to compose names in capitals.

The following reserved words have been used (some of them already in INTERLIS 1 or 2.3, these remain reserved for reasons of compatibility; they are not represented in bold and italics):

ABSTRACT	**ACCORDING**	**AGGREGATES**	**AGGREGATION**
ALL	**AND**	**ANY**	**ANYCLASS**
ANYSTRUCTURE	**ARCS**	**AREA**	**AS**
ASSOCIATION	**AT**	**ATTRIBUTE**	ATTRIBUTES
BAG	**BASE**	**BASED**	**BASKET**
BINARY	**BLACKBOX**	*BLANK*	**BOOLEAN**
BY	**CARDINALITY**	**CHARSET**	**CIRCULAR**
CLASS	**CLOCKWISE**	*CODE*	**CONSTRAINT**
CONSTRAINTS	**CONTEXT**	*CONTINUE*	**CONTINUOUS**
CONTOUR	**CONTRACTED**	**COORD**	*COORD2*
COORD3	**COUNTERCLOCKWISE**	**DATE**	**DATETIME**
DEFAULT	**DEFERRED**	**DEFINED**	*DEGREES*
DEPENDS	*DERIVATIVES*	**DERIVED**	*DIM1*
DIM2	**DIRECTED**	**DOMAIN**	**END**
ENUMTREEVAL	**ENUMVAL**	**EQUAL**	**EXISTENCE**
EXTENDED	**EXTENDS**	**EXTERNAL**	**FINAL**
FIRST	*FIX*	*FONT*	**FORM**
FORMAT	*FREE*	**FROM**	**FUNCTION**
GENERIC	**GENERICS**	*GRADS*	**GRAPHIC**
HALIGNMENT	**HIDING**	*I16*	*I32*
IDENT	**IMPORTS**	**IN**	**INHERITANCE**
INSPECTION	**INTERLIS**	**JOIN**	**LAST**
LINE	*LINEATTR*	*LINESIZE*	**LIST**
LNBASE	**LOCAL**	**MANDATORY**	**METAOBJECT**
MODEL	**MTEXT**	**MULTIAREA**	**MULTICOORD**
MULTIPOLYLINE	**MULTISURFACE**	**NAME**	**NO**
NOINCREMENTALTRANSFER	**NOT**	**NULL**	**NUMERIC**
OBJECT	**OBJECTS**	**OF**	**OID**
ON	*OPTIONAL*	**OR**	**ORDERED**
OTHERS	**OVERLAPS**	**PARAMETER**	**PARENT**
PERIPHERY	**PI**	**POLYLINE**	**PROJECTION**
RADIANS	**REFERENCE**	**REFSYS**	**REFSYSTEM**
REQUIRED	**RESTRICTION**	**ROTATION**	**SET**
SIGN	**STRAIGHTS**	**STRUCTURE**	**SUBDIVISION**
SURFACE	**SYMBOLOGY**	*TABLE*	**TEXT**
THATAREA	**THIS**	**THISAREA**	*TID*
TIDSIZE	**TIMEOFDAY**	**TO**	**TOPIC**
TRANSFER	**TRANSIENT**	**TRANSLATION**	**TYPE**
UNDEFINED	**UNION**	**UNIQUE**	**UNIT**
UNQUALIFIED	**URI**	**VALIGNMENT**	**VERSION**
VERTEX	*VERTEXINFO*	**VIEW**	**WHEN**
WHERE	**WITH**	**WITHOUT**	**XMLNS**

Table 1: Reserved words in INTERLIS 2.

2.2.8 Comments

Two forms of comments are at your disposal:

2.2.8.1 Line comment

Two exclamation marks, one immediately after the other, mark the beginning of a line comment. The line end closes the line comment.

Syntax rule:

```
!! Line comment; goes until end of line
```

2.2.8.2 Block comment

A diagonal stroke and a star introduce a block comment; a star and a diagonal stroke mark its end. It can extend over several lines and may contain itself line comments; involved block comments are not permissible.

Syntax rule:

```
/* Block comment,
    additional line comment */
```

2.3 Principal rule

Each description unit starts by indicating the language version. Thus we lay the basis for language supplements at a later date. In this document we describe version **2.4** of INTERLIS.

Subsequently you will find model descriptions.

Syntax rule:

```
INTERLIS2Def = 'INTERLIS' Version-Dec ';'
               { ModelDef }.
```

2.4 Inheritance

Different constructs of INTERLIS may be extended in the sense of the object-oriented way of thinking: Firstly a definition sets a basis that thereafter can be specialized in several steps.

Topics, classes, views, graphic definitions, units and domains can extend their corresponding basic constructs (keyword EXTENDS, resp. EXTENDED) and therewith inherit all their properties. In certain cases it may be possible that indicating EXTENDED extend a primarily defined construction, and yet keep its name.

FINAL prevents the extension of a definition. Some constructions can be defined in a still incomplete form (keyword ABSTRACT); in an extension supplied at a later time they will be complemented and become concrete definitions.

In order to indicate all admissible keywords within a certain context, we use the general property notation (cf. chapter 2.2.5 Sets of properties).

2.5 Models, topics, classes

2.5.1 Models

The term 'model' means a self-contained, complete definition. According to the type of model it may contain several constructions.

A pure type model (TYPE MODEL) may only declare measure units, domains, functions and types of lines.

A reference-system model (REFSYSTEM MODEL) should declare definitions of a type model, as well as topics and classes related to the extensions of the predefined classes AXIS, resp. REFSYSTEM (cf. chapter 2.10.3 Reference systems). The language cannot reinforce the observation of this rule. It is up to the user to abide by it.

A symbology model (SYMBOLOGY MODEL) should declare definitions of a type model, as well as topics and classes related to the extensions of the predefined class SIGN, and furthermore run time parameters (cf. chapter 2.16 Graphic descriptions and chapter 2.11 Run time parameters). It is up to the user to observe this rule.

Where no restrictive model properties are defined, a model may contain any conceivable construct.

Following the model name the language can be indicated (optional). Whenever possible this should be done according to ISO-Norm 639 , i.e. in two letters and using small letters (see www.iso.ch/), e.g. "de" stands for German, "fr" for French, "it" for Italian, "en" for English. According to ISO-Norm 3166 a country code can be added separated by an under line to indicate a variety of language used in a specific country: "de_CH" stands for the written High German used in Switzerland. This indication is of documentary value, in connection with the possibility to declare one model translation of another (TRANSLATION OF). In their structure both models must be exactly identical, hence they can only differ in the names employed. However the declaration 'translation' is not tied to the indication of language. For example in order to support local use of language or particular trade vocabulary it is admissible to add translations in the original language.

When indicating NOINCREMENTALTRANSFER you define that upon data transfer (cf. Chapter 3 Sequential Transfer) no incremental transfer must be supported. In particular the XML-schema generated from the model will not have such possibilties.

Following this, the author of the model will be identified by indicating the corresponding URI (cf. chapter 2.8.1 Strings). We expect the model name to be unequivocal within this context.

Syntax rule:

```
ModelDef = [ 'CONTRACTED' ] [ 'TYPE' | 'REFSYSTEM' | 'SYMBOLOGY' ]
           'MODEL' Model-Name [ '(' Language-Name ')' ]
           [ 'NOINCREMENTALTRANSFER' ]
             'AT' URI-String
             'VERSION' ModelVersion-String [ Explanation ]
           [ 'TRANSLATION' 'OF' Model-Name '[' ModelVersion-String ']' ]
             '='
               [ 'CHARSET' IANA-Name-String ';' ]
               [ 'XMLNS' XMLNS-String ';' ]
               { 'IMPORTS' [ 'UNQUALIFIED' ] Model-Name
               { ',' [ 'UNQUALIFIED' ] Model-Name } ';' }
               { MetaDataBasketDef
               | UnitDef
               | FunctionDef
               | LineFormTypeDef
               | DomainDef
               | ContextDef
               | RunTimeParameterDef
               | ClassDef
               | StructureDef
               | TopicDef }
           'END' Model-Name '.'.
```

By means of this model version we are able to distinguish between different versions (above all different levels of development) of a model. In the comments it is possible to add further information such as remarks concerning compatibility with earlier versions. Nevertheless at a certain moment in time there should only exist one model version. That is why no version will be indicated when importing models. If one model is a translation of another, its version has to be indicated in brackets. Hence such an indication of version within the scope of a translation definition (TRANSLATION OF) will only demonstrate which basic version was used as a base for this translation, i.e. which basic version will have exactly the same structures.

The keyword CONTRACTED is deprived of its function; however, for the sake of compatibility it will retain ist admissibility.

There is an option to define the name of a set of characters – introduced by the keyword CHARSET. Admissible names are determined by IANA[1]. Without specific indication the character set is applied in accordance with appendix D. This definition of a character set defines the amount of symbols a specific software must be able to process in transfers (such as save, search, sort, print) and neither the admissible character set nor the admissible symbole coding (cf. chapter 3.3.2 Symbol coding).

With an indication according to XMLNS it is possible to optionally define the XML-namespace[2] according to chapter 3[3]. Without such an indication the XML-namespace is derived according to a set rule (cf. chaper 3 Sequential transfer) based on the model name.

Whenever an INTERLIS construction refers to a definition stated in another model, this model has to be imported (keyword IMPORTS). Thus topics can be extended and references to classes created. IMPORTS only offers the possibility of use. When using imported definitions they still have to be referred to with a qualified name (model, topic), unless the keyword UNQUALIFIED is applied. Topics will only belong to a model (and thus can be transferred in accordance with chapter 3.3.6 Coding of topics), if they have been defined within this model (rule TopicDef).

A pre-defined base model "INTERLIS" (cf. appendix A *The internal INTERLIS-data model*) is connected to the language. It need not be imported. Nevertheless its elements are only available with unqualified names, if the model is introduced with IMPORTS UNQUALIFIED INTERLIS.

2.5.2 Topics

A topic (keyword TOPIC) contains all definitions necessary to describe a specific part of reality. A topic can also define types such as measure units, domains or structures or it may use those belonging to the enveloping model or an imported model. Put in parenthesis ( ) properties of inheritance can be defined. If a topic contains abstract definitions, that aren't concretised in the same topic, the topic must be declared to be ABSTRACT . Abstract definitions must be concretised in concrete extensions of the topic. For the datatransfer, a concrete topic is necessary. Since an extension of a topic always refers to a topic of a different name, EXTENDED would not make sense and hence is not admissible.

Syntax rules:

```
TopicDef = [ 'VIEW' ] 'TOPIC' Topic-Name
                Properties<ABSTRACT,FINAL>
                [ 'EXTENDS' TopicRef ] '='
                    [ 'BASKET' 'OID' 'AS' OID-DomainRef ';' ]
```

[1] (Internet Assigned Numbers Authority) Official Names for Character Sets, ed. Keld Simonsen et al. http://www.iana.org/assignments/character-sets

[2] Namespaces in XML 1.0 (Third Edition), Tim Bray et al., eds., W3C, 8 December 2009. http://www.w3.org/TR/REC-xml-names

[3] This is not a conceptual definition, however it is possible in the interest of simplification.

```
                 [ 'OID' 'AS' OID-DomainRef ';' ]
                 { 'DEPENDS' 'ON' TopicRef { ',' TopicRef } ';' }
                 [ 'DEFERRED' 'GENERICS'
                      GenericRef { ',' GenericRef } ';' ]
                 Definitions
         'END' Topic-Name ';'.

Definitions =  { MetaDataBasketDef
               | UnitDef
               | FunctionDef
               | DomainDef
               | ClassDef
               | StructureDef
               | AssociationDef
               | ConstraintsDef
               | ViewDef
               | GraphicDef }.

TopicRef = [ Model-Name '.' ] Topic-Name.

GenericRef = GenericCoordType-DomainRef.
```

Concerning a certain topic, which contains concrete classes, there may exist an indefinite number of baskets (databases etc.) They all possess a structure corresponding to the topic but contain different objects.

A data basket only features instances of classes (and their sub-structures). If a topic contains graphic descriptions, three types of baskets can be set up.

- Data baskets.
- Baskets with basic data for graphics. Such baskets contain the instances of classes or views which lay the foundation for graphic descriptions.
- Graphic baskets. These baskets contain graphic objects that have been realized (according to the parameter definition of the symbols employed.

As a rule baskets and objects feature an object identification. Their domains result from the corresponding definition: BASKET OID AS for the object identifications of any basket, OID AS for the object identifications of any object, provided no specific definition was made with the corresponding class. Once an OID-domain has been assigned to a topic, this can no longer be modified in extensions. In many cases it will make sense to use the standard domain STANDARDOID (cf. chapter 2.8.9 Domains of object identifications as well as appendices A and F). If there is no OID definition for a topic or a certain class, no stable object identification will be provided for these baskets resp. objects and consequently no incremental data exchange will be possible.

With exception of specific indications, one topic is, where data are concerned, independent of other topics. Data-related dependencies arise as a consequence of relationships, resp. reference attributes, which refer to a different basket, through special constraints or by means of construction of views or graphic-definitions on classes or other views, but not as a consequence of the use of meta objects (cf. chapter 2.10.1 General comments concerning meta objects). In the interest of clear recognizability of such dependencies these must be explicitly declared already in topic heading (keyword DEPENDS ON). Detailed definitions (e.g. definitions of relationships) may not violate declarations of dependencies. Cyclic dependencies are inadmissible. Without further declaration an extended topic features the same dependencies as its base-topic.

If directly or indirectly a topic refers to generic definitions that only need to be definitely set during transfer, these must explicitly be listed (DEFERRED GENERICS). At present this possibility only exists for coordinate domains (cf. chapter 2.8.8 Coordinates).

2.5.3 Classes and structures

A class definition (keyword CLASS) declares the properties of all its pertinent objects. Within a topic a class is finally defined. Class definitions can be extended whereby an extension primarily inherits all attributes of its basic class. Its domain can be limited and more attributes or additional consistency constraints can be defined.

The domain of the object identifications of all objects of such a class can be determined explicitly (OID AS) or NO OID. Without such a definition we apply the definition of the topic, unless it has been stated explicitly that no object identifications will be demanded (NO OID). If even with the topic there is no such definition, NO OID implies implicitly. NO OID signifies that the object identification is unstable. Consequently, neither an incremental update of objects of this class nor definitions of references to this class (relationships, relationship attributes) are possible. An already established OID-definition cannot be extended, unless by replacing an inherited NO OID by ANY, and an inherited ANY by a concrete definition. However, such an inherited ANY cannot be replaced by NO OID (cf. chapter 2.8.9 Domains of object identifications).

As part of a class definition constraints can be indicated. These conditions represent additional rules all objects have to comply with. Inherited constraints can never be suspended, but always are in force in addition to those declared locally.

Objects of a class are always independent and individually identifiable. Structures (keyword STRUCTURES) formally are defined in the same way as classes; however, their structural elements are dependent and cannot be identified individually. They either occur within substructures of objects (cf. chapter 2.6 Attributes) or they only exist temporarily as a result of functions. Structures may be extended into classes; as other ordinary objects objects of such classes are independent and identifiable. Classes may not be extended into structures.

Specific classes such as those for reference systems; coordinate system axis and graphic symbols (in other words extensions of the predefined class METAOBJECT) will be treated in chapter 2.10 Dealing with meta objects.

In brackets (rule Properties) characteristics of inheritance can be defined. All possibilities are admissible. Whenever a class or structure contains abstract attributes, it has to be declared ABSTRACT. Abstract attributes must be put into concrete terms within the concrete extension of the class. Yet it is also permissible to declare classes as abstract even though their attributes are fully defined. Object instances can only exist for concrete classes that have been defined within a topic, i.e. a data transfer requires concrete classes defined within a concrete topic.

If only an individual class and not the entire class are inherited, no relationships (cf. chapter 2.7 Proper relationships) may be defined.

If one topic extends another, all classes of the inherited topic are transferred. Thus they become classes of the current topic and have the same name as in the inherited topic. Such a class may be extended even while keeping its name (EXTENDED). For example if a topic T2 extends the topic T1 that contains the class C, there is only one class with C (EXTENDED) within T2, and that is C. New classes that have to differ in name from those inherited may also extend inherited classes. Consequently with C2 EXTENDS C there are two such classes in T2 (C and C2). Since INTERLIS, in the interest of simplicity and clarity only supports simple inheritance, EXTENDED is only admissible when neither in the basic topic nor in the current topic the basic class has been extended with EXTENDS. EXTENDED and EXTENDS exclude one another in the same class definition.

Classes and structures that are not built upon a class or structure already defined need no EXTENDS-part.

Syntax rules:

```
ClassDef = 'CLASS' Class-Name
             Properties<ABSTRACT,EXTENDED,FINAL>
               [ 'EXTENDS' ClassOrStructureRef ] '='
             [ ( 'OID' 'AS' OID-DomainRef | 'NO' 'OID' ) ';' ]
             ClassOrStructureDef
           'END' Class-Name ';'.

StructureDef = 'STRUCTURE' Structure-Name
                 Properties<ABSTRACT,EXTENDED,FINAL>
                   [ 'EXTENDS' StructureRef ] '='
                 ClassOrStructureDef
               'END' Structure-Name ';'.

ClassOrStructureDef = [ 'ATTRIBUTE' ] { AttributeDef }
                      { ConstraintDef }
                      [ 'PARAMETER' { ParameterDef } ].

ClassRef = [ Model-Name '.' [ Topic-Name '.' ] ] Class-Name.

StructureRef = [ Model-Name '.' [ Topic-Name '.' ] ] Structure-Name.

ClassOrStructureRef = ( ClassRef | StructureRef ).
```

Which names have to be qualified (by model-name, resp. model-name.topic-name) will be explained at the end of the following paragraph (cf. chapter 2.5.4 Namespaces). Classes and structures that are not built upon an already defined class or structure do not need an EXTENDS-part.

2.5.4 Namespaces

The term namespace signifies a set of (unequivocal) names. Each modeling element (data model, topic, class element), as well as all meta data-baskets provide their respective namespace for their name categories (type name, part name, meta object name).

Modeling elements exist on three hierarchy levels:

- Model (MODEL is sole modeling element at top level)
- Topic (TOPIC is sole modeling element on this level)
- Class elements are CLASS, STRUCTURE, ASSOCIATION, VIEW and GRAPHIC

Meta data-basket names give access to meta objects (cf. chapter 2.10 Dealing with meta objects).

There are three name categories that contain the following names:

- Type names are abbreviations (names) of units and the names of functions, line types, domains, structures, topics, classes, associations, views, graphics, baskets.
- Part names are the names of run time parameters, attributes, rules for drawing, parameters, roles and basic views.
- Meta object-names are the names of meta objects. They only exist within meta data-baskets.

Whenever a modeling element extends another, all names of the base modeling element will be added to its namespaces. In order to avoid misunderstandings modeling elements take over the name of the superior modeling elements in accordance with the name category. A name locally defined within the modeling element may not collide with a name that has been transmitted unless it is specifically termed as an extension (EXTENDED).

If you want to reference description elements of the data model, their name is usually to be qualified, i.e. it has to be indicated with preceding model and topic name. Unqualified it is possible to use the names of the namespaces of the respective modeling element.

2.6 Attributes

2.6.1 General comments concerning attributes

Its name and its type define each attribute. In brackets (rule Properties) characteristics of inheritance can be defined. Whenever an attribute is an extension of an inherited attribute, this must be explicitly indicated with EXTENDED. If the domain of this attribute is abstract, the attribute must be declared ABSTRACT. A numeric attribute (cf. chapter 2.8.5 Numeric data types) can be defined as a subdivision (keyword SUBDIVISION) of its also numeric predecessor-attribute (e.g. minutes as a subdivision of hours). Such a predecessor-attribute must be a whole number and the domain of the subdivision must be positive. If the subdivision is continuous (keyword CONTINUOUS), then the difference of the doomain limits must be in keeping with the factor between the unit of the attribute and the unit of the predecessor-attribute. If a reference system has been defined with regard to a subdivision it must tally with the system of a direct or indirect predecessor-attribute. However no INTERLIS-Compiler or runtime system will have to check this fact.

Syntax rule:

```
AttributeDef = [ [ 'CONTINUOUS' ] 'SUBDIVISION' ]
               Attribute-Name Properties<ABSTRACT,EXTENDED,FINAL,TRANSIENT>
                 ':' AttrTypeDef
                     [ ':=' Factor { ',' Factor } ] ';'.
```

If the attribute value has been determined by means of a factor (cf. chapter 2.13 Expressions), its result type must be compatible with the defined attribute, i.e. it must either feature the same domain or an extended, i.e. specialized domain. Within the scope of views – especially in the case of unions and view extensions (cf. chapter 2.15 Views) – it is possible to determine several factors and in additional view extensions further factors can be added yet. It is the last factor (base first, extension following) with defined value that is valid. Attributes, that have been determined by means of a factor and that are only of significance within other factors, should be excluded from any data transfer and should be signalized as transient.

In extensions it is possible to override an attribute by the following means:

- Limited domain.
- A constant out of the required domain. Such a definition is implicitly final, i.e. it can no longer be overridden.
- A factor, if the type of result would be admissible as extension of an attribute. Could still be overridden.

Syntax rules:

```
AttrTypeDef = ( 'MANDATORY' [ AttrType ]
              | AttrType
              | ( ( 'BAG' | 'LIST' ) [ Cardinality ]
                  'OF' AttrType ) ).

AttrType = ( Type
           | DomainRef
           | ReferenceAttr
```

```
              | RestrictedStructureRef ).

ReferenceAttr = 'REFERENCE' 'TO'
                  Properties<EXTERNAL> RestrictedClassOrAssRef.

RestrictedClassOrAssRef = ( ClassOrAssociationRef | 'ANYCLASS' )
                              [ 'RESTRICTION' '(' ClassOrAssociationRef
                                  { ';' ClassOrAssociationRef } ')' ].

ClassOrAssociationRef = ( ClassRef | AssociationRef ).

RestrictedStructureRef = ( StructureRef | 'ANYSTRUCTURE' )
                             [ 'RESTRICTION' '(' StructureRef
                                 { ';' StructureRef } ')' ].

RestrictedClassOrStructureRef = ( ClassOrStructureRef | 'ANYSTRUCTURE' )
                                    [ 'RESTRICTION' '(' ClassOrStructureRef
                                      { ';' ClassOrStructureRef } ')' ].
```

Within the scope of extensions it is permissible to indicate only MANDATORY. In this case the already defined attribute type is valid. However it is strictly required that the value be defined.

2.6.2 Attributes with domain as type

Direct type definition (rule Type) and the use of already defined domains (rule DomainRef) are conceivable as types of attribute. Various possibilities are listed in chapter 2.8 Domains and constants.

2.6.3 Reference attributes

By using a reference attribute we create a reference to another object. Links between independent objects have to be defined by means of proper relationships (cf. chapter 2.7 Proper relationships).

Classes, whose elements are in consideration for reference, may be concrete or abstract object or relationship classes, but not structures (since these are dependent objects). All concrete classes some in question, provided they correspond to the listed primary class, resp. one of the listed restricting (RESTRICTION TO) classes (class itself or one of its subclasses). On all restriction levels (initial definition or steps of extension) all classes that are still admissible must be listed. Each class defined as a restriction must be subclass of a class hitherto admissible. However such a class is only admissible if it belongs to the same topic as the reference attribute or to a topic the referenced topic is depending on (DEPENDS ON). If it is required that the reference may refer to the object of a different basket of either the same or a different topic (prerequisite: DEPENDS ON), the property EXTERNAL must be indicated. In extensions it is possible to omit and thus exclude this property, however it cannot be added. Unless the reference attribute has been declared abstract, there must be a minimum of one admissible concrete subclass.

2.6.4 Structure attributes

Values of structure attributes are composed of one or several structural elements (instances of structures; cf. Chapter 2.5.3 Classes and structures). Structural elements have no OID, exist only in connection with their main object and can only be found by passing via the former. The admissible number of structure elements is derived from the cardinality indicated respectively the key word MANDATORY, either present or absent. By means of LIST a certain sequence may be obtained for these structure elements. In a case of a transfer this sequence must remain intact. By means of BAG these structure elements are in random order.

Structure attributes may be defined with concrete or abstract types (structures, domains, reference attributes; but no classes).

With structures their order is a result oft he indicated structure (rule RestrictedStructureRef). Structure attributes can be defined with concrete or abstract structures. On principle all structures (but not all classes) are suitable for concrete structure elements, provided they correspond to or extend the structures listed as primary or restricting (RESTRICTION TO). No further information is necessary for the structures of a current topic. Structures of all other topics will only be taken into consideration if they, resp. a basic class also defined within this other topic, are explicitly listed in the definition of the structure attribute as a primary or restricting structure. At each restriction level (primary definition or extensions) all structures still admissible have to be listed. Each structure defined as an extension must be an extension of a hitherto admissible structure.

If the structure of a structure attribute is arbitrary (ANYSTRUCTURE) or if no structure can be found that complies with the definition, then the structure attribute must be declared as abstract, provided it is mandatory or if its minimal cardinality is greater than zero. If structures are defined as formal function arguments (cf. chapter 2.14 Functions), then paths to structure elements or to objects come into question as current arguments. Moreover through ANYSTRUCTURE all structure elements and all objects are compatible.

An ordered sub-structure (LIST) may not be extended by a disordered sub-structure (BAG). The same rules as with relationships apply to cardinality (cf. chapter 2.7.3 Cardinality).

2.7 Proper relationships

2.7.1 Description of relationships

Proper relationships (as opposed to reference attributes (cf. chapter 2.6 Attributes) are described as independent constructs. However they largely have the same properties as classes. For instance they themselves can feature local attributes and consistency constraints. The association name may be omitted. In this case it will be implicitly composed with the role names (starting with the first, then second, etc.). Nevertheless the most important property of a relationship consists in the listing of at least two roles with their assigned classes or relationships (same rules apply as with reference attributes, (cf. chapter 2.6.3 Reference attributes) and details such as force of relationship and cardinality. Role names should typically be nouns. They may but do not have to tally with the names of the assigned classes or relationships. However the relationship to be defined may not be an extension of a relationship thus assigned. It is also possible to alternatively assign different classes or relationships to one role. Such an alternative class or relationship may not be an extension of another alternative class or relationship of the same role.

Example of a relationship between the class K on the one side, and the classes K or L on the other:

```
ASSOCIATION A =
  K -- K;
  KL -- K OR L;
END A;
```

On principle relationships can be extended in the same way as classes. In order to ensure that the significance of the relationship is clear and constant, no additional roles may appear in extensions. However it is possible to restrict the classes or relationships assigned as well as the cardinality. Roles unchanged must not be listed.

Example as to how the relationship A to A1 can be specialized, when only on one hand references to K1 (a subclass of K) and to K, L1, L2 (subclasses of L) on the other are admissible:

```
ASSOCIATION A1 EXTENDS A =
  K (EXTENDED) -- K1;
  KL (EXTENDED) -- K OR L RESTRICTION (L1, L2);
END A1;
```

Hence an object of class K1 can either be related via a relationship A with objects of the classes K or L (admissible, since K1 is a subclass of K) or via a relationship A1 with objects of the classes K, L1 or L2. If furthermore we would like to make sure that an object K1 within the role K can only enter into a specialized relationship A1(as opposed to the general relationship A), then the role K has to be signalized as hiding (HIDING).

```
ASSOCIATION A1 EXTENDS A =
  K (EXTENDED, HIDING) -- K1;
  KL (EXTENDED) -- K OR L RESTRICTION (L1, L2);
END A1;
```

However this is only admissible if for K1 no other relationships as extensions from A have been defined where the role K has not been signalized as hiding.

Syntax rules:

```
AssociationDef = 'ASSOCIATION' [ Association-Name ]
                   Properties<ABSTRACT,EXTENDED,FINAL,OID>
                   [ 'EXTENDS' AssociationRef ]
                   [ 'DERIVED' 'FROM' RenamedViewableRef ] '='
                   [ ( 'OID' 'AS' OID-DomainRef | 'NO' 'OID' ) ';' ]
                   { RoleDef }
                   [ 'ATTRIBUTE' ] { AttributeDef }
                   [ 'CARDINALITY' '=' Cardinality ';' ]
                   { ConstraintDef }
                 'END' [ Association-Name ] ';'.

AssociationRef = [ Model-Name '.' [ Topic-Name '.' ] ] Association-Name.

RoleDef = Role-Name Properties<ABSTRACT,EXTENDED,FINAL,HIDING,
                              ORDERED,EXTERNAL>
            ( '--' | '-<>' | '-<#>' ) [ Cardinality ]
            RestrictedClassOrAssRef { 'OR' RestrictedClassOrAssRef }
            [ ':=' Role-Factor ] ';'.

Cardinality = '{' ( '*' | PosNumber [ '..' ( PosNumber | '*' ) ] ) '}'.
```

The association reference between objects may be considered as an independent object (association reference). Primarily all roles for the references to reference objects will be stated on this association reference (they all have to be defined!). Without further information this association reference will be identified through the references to the objects connected by the association reference. Between these objects only such an association reference will be admissible. Multiple association reference among the same object combination is only admissible if a cardinality with upper limit greater than one is explicitly required for the relationship (CARDINALITY). In this case an object identification (by means of Property OID) is also required. If an object identification of its own is required, then the relationship itself can again be used as a reference in roles.

If you strive to achieve compatibility with INTERLIS 1, only define relationships with two roles and do not exceed maximal number of cardinality of a role greater than 1.

Normally concrete relationships between objects must be established explicitly by means of an application and subsequently be fixed as instances by the processing system. A relationship can also be derived from a view without being instanced (DERIVED FROM). Such a relationship can be an extension of an abstract relationship, but cannot be abstract itself. If it is extended, the extension must build upon the same view or upon one of its extensions. One object path has to be indicated (cf. chapter 2.13 Expressions) which designates a class or association corresponding to the role. The cardinality must be in accordance with the performance of the view. This however can only be checked during run time.

A typical case of an application might be the derivation of a relationship from its geometrical conditions: within a view (union) which is referred to in the association and based upon the geometry of the real estates they are situated on, e.g. buildings are brought in relation with these estates (cf. chapter 2.15 Views).

2.7.2 Force of relationship

In accordance with UML we distinguish different forces of relationship. In order to explain these we mainly describe the influence the force of relationsphip possesses when copying or deleting objects. For INTERLIS 2 however only the deleting of objects (due to incremental update) is of consequence. In addition there are other considerations that influence the force of relationship. Above all it is up to the individual software to set more refined forces of relationship or even other criteria for the processing of certain operations.

- Association: The objects concerned are loosely connected. If one of the objects is copied, the copy is connected to the same objects as the original. If an object is deleted, the relationship is deleted at the same time, however the object itself remains. Syntactically we indicate '--' with all roles.
- Aggregation: There is only a feeble relationship between the entirety and its parts. Aggregations are only permitted in relationships with two roles. Syntactically the role that leads to the entirety must be indicated with a rhombus '-<>'. The role that leads to the part is defined with '--'. An object class may appear in several aggregations in the role of a part. Hence to a certain object of a part there can be assigned different objects of the entirety. As opposed to associations, when making a copy of the entirety corresponding copies of the parts are established. Since within the scope of INTERLIS 2 the copying of objects is without significance, INTERLIS 2 treats aggregations much as associations.
- Composition: A strong relationship exists between the entirety and its parts. An object class may appear in the part-role in more than one composition. However only one entirety may be assigned to a certain part-object. When deleting the entirety its parts are likewise deleted. The role that leads to the entirety is indicated with a filled rhombus '-< #>'.

Associations can be extended to aggregations, these can be extended to compositions, but the reverse is not permissible.

2.7.3 Cardinality

Cardinality defines the minimum and maximum number of permissible objects, where only one value is indicated minimum and maximum are the same. If for the maximum an asterisk replaces the number, there is no upper limit for the number of sub-objects. The indication of cardinality with {*} is the equivalent of {0..*}. If there is no indication of cardinality then normally {0..*} is in force. With composition roles only {0..1} or {1} are admissible (one part can only belong to one entirety). Where there is no indication {0..1} is in force.

In Extensions cardinality can only be restricted but not extended. Hence if in the first place a cardinality of {2..4} has been indicated, an extension cannot declare {2..5}, {7} or {*}. If in extended attributes the indication of cardinality is omitted, it means that the inherited value has been taken over.

Depending on the form of application, cardinality has the following significance:

- With sub-structures: number of admissible elements.
- With roles of relationships: Number of objects corresponding to one role which via the relationship can be assigned to an arbitrary combination of objects that correspond to the other roles.
- With the relationship as a whole: Number of association references for an arbitrary combination of objects in accordance with all roles of the relationship.

2.7.4 Ordered relationships

If you want to achieve that a relationship, from the viewpoint of a certain reference class, is set up in a certain order, this property (ORDERED) has to be required within the role. This order is defined when establishing the relationship and has to be maintained throughout all transfers.

2.7.5 Relationship access

Relationship access (AssociationAccess) means the possibility, from the viewpoint of one object, to navigate to the association references and further on to the reference objects in accordance with a relationship. Relationship accesses need not be defined; they originate in the definition of a relationship for all classes assigned via roles, which have been defined in the same topic as the relationship. If a class involved in a relationship has been defined in a different topic (topic-spanning relationship), or if it ought to be permitted that a reference object corresponding to this role can be found in a basket other than the one of the association reference, this fact has to be stated specifically with the role (EXTERNAL, cf. chapter 2.7.1 Description of relationships and chapter 2.6.3 Reference attributes). Then this class will not be supplied with relationship accesses. This property will be defined within the basic definition of a relationship and cannot be modified within an extension. If a role refers to a class inherited from the inherited topic, then relationship accesses out of this class are only permitted if this class has been extended within the current topic of the same name (EXTENDED). This restriction prevents from being modified subsequently (i.e. out of the scope within which it had originally been defined).

Relationship access will be transmitted to subclasses, unless this is ruled out for a role of an extended relationship by means of the requirement (HIDING).

Relationship accesses are of significance for all path descriptions (cf. chapter 2.13 Expressions).

2.8 Domains and constants

Different aspects are linked to the idea of a domain. Primarily a data type has to be defined. INTERLIS data types are independent of their implementation. That is why we do not speak of integer or real for example, but simply of numeric data types (cf. chapter 2.8.5 Numeric data types).

Once the data type has been determined, further specifications can be necessary or possible, depending on the data type in question. As long as a domain definition is incomplete (e.g. missing concrete domain definition with a numeric domain), it must be declared abstract (keword ABSTRACT, rule Properties). To simplify a transition, especially from NS03-coordinates to NS95-coordinates, the domain of coordinates can be defined as generic (keyword GENERIC, rule Properties). The domains employed (especially lines), attributes, classes and topics need not as a consequence be defined as abstract, even though the coordinate domain will only be determined in specific models or even as late as in transfer data (cf. chapter 2.8.8 Coordinates).

Domains can be – as other constructs – inherited and thereafter extended, provided they have not been defined FINAL. In such a case the basic principle that extended definitions must always be compatible with their base definition, is of great importance. Thus in domains, extensions (keyword EXTENDS) really are specifications, resp. restrictions. The keyword EXTENDED (rule Properties) is not admissible. In the interest of readability we suggest that parts of definitions in base domains (such as measure units) be repeated in the extension, even if they are unchanged. Example:

```
DOMAIN
  Value (ABSTRACT) = NUMERIC .. NUMERIC;      !! abstract domain
  GenValue EXTENDS Value = 10.0 .. 100.0;     !! concrete extension
  SpecValue EXTENDS GenValue = 20.0 .. 90.0; !! ok
  SpecValue EXTENDS GenValue = 0.0 .. 110.0; !! false, since incompatible
```

An important issue in the definition of domains is whether the value "undefined" belongs to the domain or not. Without any specific indication it does form a part, yet it is possible to demand its exclusion, i.e. an attribute with this domain must always be defined (keyword MANDATORY). MANDATORY alone is only admissible in extensions.

When defining the attribute of a class or structure (and only then), MANDATORY may occur if the domain has been declared FINAL and consequently ought not to be restricted any further.

By means of using the keyword CONSTRAINTS the domain definition can be extended by one or several constraints, e.g. in order to admit only certain patterns of character sequences in text domains. Where several constraints are stated, they all apply. Within the domain definition each constraint will have its own unequivocal name. Syntax rules:

```
DomainDef = 'DOMAIN'
                 { Domain-Name Properties<ABSTRACT,GENERIC,FINAL>
                     [ 'EXTENDS' DomainRef ] '='
                 ( 'MANDATORY' [ Type ] | Type ) [ 'CONSTRAINTS'
                   Constraints-Name ':' Logical-Expression
                   { ',' Constraints-Name ':' Logical-Expression } ] ';' }.

Type = ( BaseType | LineType ).

DomainRef = [ Model-Name '.' [ Topic-Name '.' ] ] Domain-Name.

BaseType = ( TextType
             | EnumerationType
             | EnumTreeValueType
             | AlignmentType
             | BooleanType
             | NumericType
             | FormattedType
             | DateTimeType
             | CoordinateType
             | OIDType
             | BlackboxType
             | ClassType
             | AttributePathType ).
```

In comparison operations (cf. chapter 2.13 Expressions), attribute values can also be compared with constants. These are defined as follows:

```
Constant = ( 'UNDEFINED'
             | NumericConst
             | TextConst
```

```
               | FormattedConst
               | EnumerationConst
               | ClassConst
               | AttributePathConst ).
```

Every data type defines its own specific constants.

2.8.1 Strings

The term ‚string' depicts a series of symbols of maximum length. All the symbols supplied must be clearly defined (cf. appendix D *Symbol table*).

Within the data type MTEXT you will find the symbols «Carriage Return» (#xD), «Line Feed» (#xA) and «Tabulator» (#x9), as opposed to the domain of the data type TEXT.

With the data type string (TEXT), it is primarily the length of the string that is of interest. Depending on the form of definition, it is indicated explicitly or implicitly. In its explicit form (TEXT*…), the maximum length is determined in number of symbols (exceeding 0). If only the keywords TEXT or MTEXT are indicated, the number of symbols is unlimited. Within the scope of an extension its length can be shortened (lengthening would lead to a domain no longer compatible with the basic domain).

An INTERLIS string length features the number of symbols as perceived by the user, but not the maximum number of memory storage spaces a system would need for the representation of such a string. Strings whose length is zero are considered to be undefined.

Note: In connection with INTERLIS the length of a string is defined as the number of those symbols which, according to Unicode-Standard, possess the canonical combination class n° 0, after the string has been put into its canonically decomposed form of Unicode (see www.unicode/org./unicode/reports/tr15/). Thus a string consisting of <LATIN CAPITAL LETTER C WITH CIRCUMFLEX><COMBINING CEDILLA> possesses the length 1, the same as the equivalent string <LATIN CAPITAL LETTER C>< COMBINING CIRCUMFLEX ACCENT><COMBINING CEDILLA>. According to the definition above, ligatures for "fl" or "ffi" possess the length 1. However we advise you against using such representation forms for string attributes.

The name string type (keyword NAME) defines a domain that corresponds exactly to the one of INTERLIS-names (cf. chapter 2.2.2 Names). It is applied in the predefined class METAOBJECT and above all in the classes for reference systems, as well as symbols (cf. chapter 2.10.3 Reference systems and chapter 2.16 Graphic descriptions), because that is where data attributes have to coincide with description elements of models.

URI (Uniform Resource Identifier) represents a further string type, e.g. http-, ftp- and mailto-addresses (see paragraph 1.2 in internet standard IETF RFC 2396 at www.w3.org). The length of a URI in INTERLIS is limited to 1023 symbols and hence corresponds to the following definition:

```
DOMAIN
  URI (FINAL) = TEXT*1023;  !! ATTENTION: according to IETF RFC 2396
  NAME (FINAL) = TEXT*255;  !! ATTENTION: according to chapter 2.2.2 Names
```

Syntax rules:

```
TextType = ( 'MTEXT' [ '*' MaxLength-PosNumber ]
           | 'TEXT' [ '*' MaxLength-PosNumber ]
           | 'NAME'
           | 'URI' ).

TextConst = String.
```

2.8.2 Enumerations

By means of an enumeration, all values admissible for this type are determined. However an enumeration is not simply linear, but features a tree-like structure. The leaves of this tree (but not its knots) form the set of admissible values. Example:

```
DOMAIN
  Colors = (red (dark_red, orange, crimson),
            yellow,
            green (light_green, dark_green));
```

Produces the following admissible values – described by means of constants:

```
#red.dark_red #red.orange #red.crimson #yellow #green.light_green #green.dark_green
```

A subdivision level is indicated in brackets (). The element names of each subdivision level must be unequivocal. You are at liberty to choose whatever tree depth seems convenient.

In an ordered enumeration (keyword ORDERED), the sequence of elements is clearly defined. If the enumeration is circular (keyword CIRCULAR), the sequence of elements is defined as if the enumeration were ordered. Moreover it is stated that the last element will again be followed by the first.

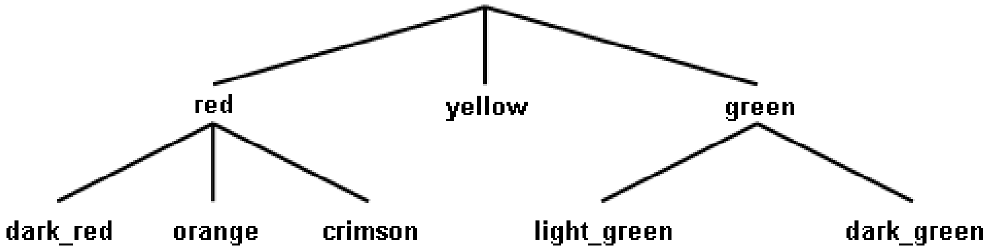


Figure 8: Example of an enumeration.

Besides the enumeration definition in its strictest sense it is also possible to define a domain that will comprise as admissible values all leaves and nodes of an enumeration definition (ALL). Hence it is possible to assign to such an attribute the value of an attribute of its fundamental enumeration definition.

Examples

```
DOMAIN
  Position = (bottom, center, top) ORDERED;
  Weekdays = (Working days (Monday, Tuesday, Wednesday,
                            Thursday, Friday, Saturday),
              Sunday) CIRCULAR;
  ValuesOfWeekdays = = ALL OF Weekdays;
```

Syntax rules:

```
EnumerationType = Enumeration [ 'ORDERED' | 'CIRCULAR' ].

EnumTreeValueType = 'ALL' 'OF' Enumeration-DomainRef.

Enumeration = '(' EnumElement { ',' EnumElement } [ ':' 'FINAL' ]
                | 'FINAL' ')'.

EnumElement = EnumElement-Name { '.' EnumElement-Name } [Sub-Enumeration].

EnumerationConst = '#' ( EnumElement-Name { '.' EnumElement-Name }
                                            [ '.' 'OTHERS' ]
                       | 'OTHERS' ).
```

Within the scope of new definitions of enumerations (primary definitions, supplementary elements of extensions), the EnumElement may only consist of one name. Several names are only admissible in order to identify a prevailing enumeration element for an extension.

On the one hand enumerations can be extended by defining sub-enumerations for leafs of hitherto existing enumerations (in other words enumeration elements which do not feature sub-enumerations). In the extended definition former leaves now become knots for which no values may be defined.

On the other hand each individual part-enumeration in extensions can be supplemented with further elements (knots or leaves). Thus the basic enumeration comprises besides the elements mentioned more potential elements that will only be defined in extensions. At basic level such potential values can be approached via the value OTHERS in expressions, function arguments and symbol assignations (cf. chapter 2.13 Expressions, chapter 2.14 Functions and chapter 2.16 Graphic descriptions). However OTHERS is no admissible value within the scope of the class that the object belongs to. The possibility to add supplementary elements of enumeration in extensions can be restricted by declaring the partial enumeration as final (FINAL). This is done either after the last element listed or within the scope of an extension even without adding new elements.

Circular enumerations (keyword CIRCULAR) cannot be extended.

```
DOMAIN
  Color = (red,
           yellow,
           green);
  ColorPlus EXTENDS Color = (red (dark_red, orange, crimson),
                             green (light_green, dark_green: FINAL),
                             blue);
  ColorPlusPlus EXTENDS ColorPlus = (red (FINAL),
                                     blue (light_blue, dark_blue));
```

for Color Plus this results in the following admissible values – described via constants:

```
#red.dark_red #red.orange #red.crimson #yellow #green.light_green #green.dark_green
#blue
```

and for ColorPlusPlus:

```
#blue.light_blue #blue.dark_blue instead of #blue
```

By indicating FINAL with the shades of green in ColorPlus it is not admissible to define further shades of green in ColorPlusPlus. FINAL in the subdivision of red in ColorPlusPlus prevents the addition of more varieties of red in possible extensions of ColourPlusPlus.

2.8.3 Text orientation

For the processing of plans and maps text positions must be determined. It has to be stated which point of the text the position corresponds to. The horizontal alignment determines whether the position is situated on the left hand or right hand margin or in the center of the text. The vertical alignment determines the position in the direction of the text size.

The distance between cap and base corresponds to the size of capital letters. Descenders are placed in the area base-bottom.

Horizontal and vertical alignment can be understood to signify the following, predefined enumerations:

```
DOMAIN
  HALIGNMENT (FINAL) = (Left, Center, Right) ORDERED;
  VALIGNMENT (FINAL) = (Top, Cap, Half, Base, Bottom) ORDERED;
```

Syntax rule:

```
AlignmentType = ( 'HALIGNMENT' | 'VALIGNMENT' ).
```

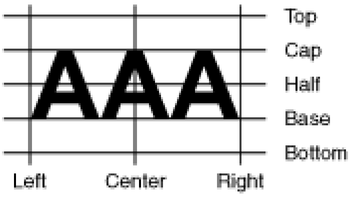


Figure 9: Text orientation horizontally (HALIGNMENT) and vertically (VALIGNMENT).

2.8.4 Boolean

The type Boolean features the values true and false. It can be interpreted as the following pre-defined enumeration:

```
DOMAIN
  BOOLEAN (FINAL) = (false, true) ORDERED;
```

Syntax rule:

```
BooleanType = 'BOOLEAN'.
```

2.8.5 Numeric data types

The most important information in connection with numeric data types are the minimum and maximum value including number of decimals, as well as the scaling factor. Additionally it can be indicated that the type is circular (keyword CIRCULAR), i.e. that in the last significant digit the maximum value increased by 1 and the minimum value technically have the same meaning (e.g. with angles 0 .. 359 degrees). If the attribute has been defined as a continuous subdivision of the predecessor attribute (cf. chapter 2.6.1 General comments concerning attributes), the type has to be defined as circular. If the indication of minimum and maximum value should be missing (keyword NUMERIC), the domain is regarded as abstract.

```
DOMAIN
  Angle1 = 0.00 .. 359.99 CIRCULAR [degree];  !! correct
  Angle2 = 0.00 .. 360.00 CIRCULAR [degree];  !! syntactically correct, but
                                              !! technically false, since 360.01
                                              !! subsequently corresponds to the
                                              !! minimal value 0.00
```

The number of digits must coincide in minimal and maximum value. By means of scaling it is possible to describe float-numbers, but then the minimum as well as the maximum value have to be indicated in mantissa type number representation, i.e. starting with zero (0) and followed by a decimal point (.) the first digit after the decimal point must differ from zero (0). The scaling of the minimum value must be inferior to the scaling of the maximum value. However the notation of minimum and maximum value in no way qualifies as an indication on how to transfer these values (if the domain is defined with 000 .. 999 this does not mean that the value 7 is transferred as 007). Float-numbers are the exception to this rule. These are to be transferred in mantissa type number representation and with scaling.

With extensions the minimum, resp. maximum values can only be restricted. Thus the numeric range becomes smaller. Observe the following situation:

```
DOMAIN
  Normal = 0.00 .. 7.99;
```

```
Exact EXTENDS Normal = 0.0000 .. 7.9949;    !! correct, since Normal
                                            !! is also representable
Exact EXTENDS Normal = 0.0000 .. 7.9999;    !! false, since rounded
                                          !! outside of Normal
```

In order to explain more clearly the significance of a value, a measure unit can be indicated (cf. chapter 2.9 Units). Abstract measure units are only admissible as long as the domain itself remains undefined (keyword NUMERIC).

The following rules apply for extensions:

– If a concrete base domain features no measure units, then none may appear in extensions.

– Where the base domain employs an abstract measure unit, only such measure units may be used in extensions, as are extensions of the aforesaid measure unit.

– Where the base domain employs concrete measure units, they cannot be overridden in extensions.

Examples:

```
UNIT
  foot [ft] = 0.3048 [m];

DOMAIN
  Distance (ABSTRACT) = NUMERIC [Length];
  MeterDist (ABSTRACT) EXTENDS Distance = NUMERIC [m];
  FootDist (ABSTRACT) EXTENDS Distance = NUMERIC [ft];
  ShortMeters EXTENDS MeterDist = 0.00 .. 100.00 [m];
  ShortFeet EXTENDS FussDist = 0.00 .. 100.00 [ft];
  ShortFeet2 (ABSTRACT) EXTENDS ShortMeters = NUMERIC [ft]; !! false: m vs. ft
```

A scalar system (cf. chapter 2.10.3 Reference systems) can be attributed to a numeric domain. Thereafter the values refer to the zero point determined by the scalar system. Consequently, they are absolute values in this scalar system. If in the class of the scalar system the unit is not ANYUNIT, such a unit has to be indicated with the numeric data type as will be compatible with the reference system. With regard to a reference system you may indicate the axis referred to by the values. The unit indicated must be compatible the unit of the corresponding axis. Where this information is omitted, the reference is not specified but ensues from the subject (e.g. if you refer to an ellipse when stating a height, elliptic heights are meant). If you refer to a different domain, the same reference system should be applicable as in the former. In this case the indication of the axis may only be omitted where numeric domains are concerned. With a coordinate domain, the indication of the axis is obligatory. The indication of reference systems cannot be changed in extensions.

If the numeric value represents an angle, its orientation can be determined. In the case of directions it can be specified which coordinate system they refer to (defined by its coordinate domain). Thus both zero direction (azimuth) and sense of rotation are known (cf. chapter 2.8.8 Coordinates). This indication cannot be changed in extensions.

Besides decimal numbers two more numbers are defined as numeric constants: pi (keyword PI) and e – base of the natural logarithm (keyword LNBASE).

Syntax rules:

```
NumericType = ( Min-Dec '..' Max-Dec | 'NUMERIC' ) [ 'CIRCULAR' ]
              [ '[' UnitRef ']' ]
              [ 'CLOCKWISE' | 'COUNTERCLOCKWISE' | RefSys ].

RefSys = ( '{' RefSys-MetaObjectRef [ '[' Axis-PosNumber ']' ] '}'
         | '<' Coord-DomainRef [ '[' Axis-PosNumber ']' ] '>' ).

DecConst = ( Dec | 'PI' | 'LNBASE' ).
```

```
NumericConst = DecConst [ '[' UnitRef ']' ].
```

2.8.6 Formatted domains

Formatted domains are based on structures and use their numeric and formatted attributes within one format. On the one hand this format serves the data exchange (cf. chapter 3.3.11.5 Coding of formatted domains), on the other hand the definition of both lower and upper limit of the domain.

Syntax rules:

```
FormattedType = ( 'FORMAT' ( 'BASED' 'ON' StructureRef FormatDef
                             [ Min-String '..' Max-String ]
                           | FormattedType-DomainRef
                             Min-String '..' Max-String
                           ) )
                | Min-String '..' Max-String.

FormatDef = '(' [ 'INHERITANCE' ]
                [ NonNum-String ] { BaseAttrRef NonNum-String }
                                    BaseAttrRef [ NonNum-String ] ')'.

BaseAttrRef = ( NumericAttribute-Name [ '/' IntPos-PosNumber ]
              | StructureAttribute-Name '/' Formatted-DomainRef ).

FormattedConst = String.
```

Primarily, a basic definition of a formatted domain defines the structure it is built upon and the format which is being used. In addition it is possible to define both lower and upper limit of the domain. They may not extend the limits defined by the structure.

Within the scope of an extension it is possible to refer to an extension of the original structure, to supplement the format (the inherited part must figure at the beginning and in the interest of clearness it should be signalized by the keyword INHERITANCE) and to restrict the domain.

On the one hand the format definition may contain constant strings, which do not start with a number (at the start, at the end or in between the individual attribute references), on the other it may contain direct or indirect (via structure attributes) attribute references. The attribute reference must either designate a numeric attribute or a structure attribute. In the case of a numeric attribute it is possible to define the number of digits on the left of the decimal point. As a result leading noughts will be introduced if necessary. The number of decimals will result from the numeric domain. With structure attributes you must define in accordance with which formatted domain they have to be formatted. The structure must either tally with the basic structure of the domain or be an extension of it.

2.8.7 Date and time

Whenever indications of date or time do not only consist of one single value (e.g. year, second), we tend to use formatted domains.

Syntax rule:

```
DateTimeType = ( 'DATE' | 'TIMEOFDAY' | 'DATETIME' ).
```

The domains for DATE, TIMEOFDAY resp. DATETIME correspond to the following definitions of INTERLIS.XMLDate, INTERLIS.XMLTime resp. INTERLIS.XMLDateTime.

In the interest of compatibility with XML corresponding elements are predefined by INTERLIS:

```
UNIT
  Minute [min] = 60 [INTERLIS.s];
  Hour   [h]   = 60 [min];
  Day    [d]   = 24 [h];
  Month [M] EXTENDS INTERLIS.TIME;
  Year [Y] EXTENDS INTERLIS.TIME;

REFSYSTEM BASKET BaseTimeSystems ~ TIMESYSTEMS
  OBJECTS OF CALENDAR: GregorianCalendar
  OBJECTS OF TIMEOFDAYSYS: UTC;

STRUCTURE TimeOfDay (ABSTRACT) =
  Hours: 0 .. 23 CIRCULAR [h];
  CONTINUOUS SUBDIVISION Minutes: 0 .. 59 CIRCULAR [min];
  CONTINUOUS SUBDIVISION Seconds: 0.000 .. 59.999 CIRCULAR [INTERLIS.s];
END TimeOfDay;

STRUCTURE UTC EXTENDS TimeOfDay =
  Hours(EXTENDED): 0 .. 23 {UTC};
END UTC;

DOMAIN
  GregorianYear = 1582 .. 2999 [Y] {GregorianCalendar};

STRUCTURE GregorianDate =
  Year: GregorianYear;
  SUBDIVISION Month: 1 .. 12 [M];
  SUBDIVISION Day: 1 .. 31 [d];
END GregorianDate;

STRUCTURE GregorianDateTime EXTENDS GregorianDate =
  SUBDIVISION Hours: 0 .. 23 CIRCULAR [h] {UTC};
  CONTINUOUS SUBDIVISION Minutes: 0 .. 59 CIRCULAR [min];
  CONTINUOUS SUBDIVISION Seconds: 0.000 .. 59.999 CIRCULAR [INTERLIS.s];
END GregorianDate;

DOMAIN XMLDate = FORMAT BASED ON GregorianDate ( Year/4 "-" Month/2 "-" Day/2 );
DOMAIN XMLTime = FORMAT BASED ON UTC ( Hours/2 ":" Minutes/2 ":" Seconds/2 );
DOMAIN XMLDateTime EXTENDS XMLDate = FORMAT BASED ON GregorianDateTime
                                          ( INHERITANCE "T" Hours/2 ":" Minutes/2
                                                        ":" Seconds/2 );
```

Practical example:

```
CLASS Project =
  Start: FORMAT INTERLIS.XMLDateTime "2000-01-01T00:00:00.000" ..
                                     "2005-12-31T23:59:59.999";
  End: FORMAT INTERLIS.XMLDateTime "2002-01-01T00:00:00.000" ..
                                   "2007-12-31T23:59:59.999";
END Project;
```

2.8.8 Coordinates

Coordinates can be defined one, two or three dimensionally, and hence are either a single digit, double digits or triple digits. It is permissible to add the second or third dimension only in an extension. For every dimension the numeric domain, as well as perhaps a measure unit and a coordinate system (including numbers of axis) must be indicated. Only concrete measure units can be indicated. If no reference system is indicated and if furthermore measure units are not or as length units, the program system that implements the model may presume that it is dealing with Cartesian coordinates.

Whenever a rotation definition occurs (keyword ROTATION), it is possible within the scope of definitions of zero directions (cf. chapter 2.8.5 Numeric data types) to refer to such a coordinate reference system. The rotation definition determines which axis of the coordinate domain corresponds to the zero direction and which to the direction of a positive right angle. It may be missing in a concrete coordinate definition and could be added in an extension.

Any indication concerning definition of axis and rotation cannot be changed in extensions.

```
DOMAIN
  CHCoord = COORD 480000.00 .. 850000.00 [m] {CHLV03[1]},
                  60000.00 .. 320000.00 [m] {CHLV03[2]},
                 ROTATION 2 -> 1 REFSYS "EPSG:21781";
```

In both defined axes the admissible domain is indicated as well as the units and reference system including the number of axis the coordinates refer to. The actual axes are defined within the reference system. The rotation definition determines that the zero direction leads from axis #2 to axis #1, in other words in the Swiss Federal system where value #1 corresponds to east, and value #2 to north, the zero direction shows north and turns clockwise.

```
DOMAIN
  WGS84Coord = COORD -90.00000 ..  90.00000          [Units.Angle_Degree] {WGS84[1]},
                      0.00000 .. 359.99999 CIRCULAR [Units.Angle_Degree] {WGS84[2]},
                    -1000.00 .. 9000.00              [m] {WGS84Alt[1]};
```

Typically geographic coordinates are represented in degrees and refer to an ellipsoid coordinate system (e.g. CH1903). Then again the altitude is described in meters. It refers to a special ellipse height system with one axis.

Syntax rules:

```
CoordinateType = ( 'COORD' | 'MULTICOORD' ) NumericType
                    [ ',' NumericType [ ',' NumericType ]
                        [ ',' RotationDef ] [ 'REFSYS' Name-String ] ].

RotationDef = 'ROTATION' NullAxis-PosNumber '->' PiHalfAxis-PosNumber.
```

By using the optional indication REFSYS it is possible to define the EPSG-Code[4] of the reference system according to the pattern EPSG:PosNumber.

By using the keyword MULTICOORD instead of COORD the domain will be defined as a non-structured set of coordinates. However, in comparison with a formulation with LIST/BAG it is mandatory that the individual values refer to the same reference system.

In the case of incomplete definitions, the value range must either be declared abstract (property ABSTRACT) or generic (property GENERIC).

With abstract coordinate domains even the number of dimensions may remain unspecific. Abstract value rages can only be used in attributes previously declared as abstract. Subsequently there are various ways to concretize abstract coordinate ranges.

In the case of generic coordinate domains it is mandatory to determine the number of dimensions. However, it is sufficient to limit this indication to NUMERIC. Generic domains may also be used in attributes not declared abstract. Generic domains may be concretized in the same way as abstract coordinate domains where in particular their numeric domain of axis and their reference system are defined resp. limited.

```
DOMAIN
  Coord2 (GENERIC) = COORD NUMERIC, NUMERIC;

CLASS Point =
  Pos: Coord2;
END Point;
```

[4] EPSG Geodetic Parameter Registry. http://www.epsg-registry.org/

However, such a definition is part of a topic (TOPIC) which must be declared abstract, unless governed by a context definition.

By means of a context definition it is possible to define concrete coordinate domains for a generic coordinate domain (GenericCoordDef-DomainRef). If for one generic ccordinate domain not only one but a selection of possible concrete coordinate domains has been defined, then –- upon transfer of the individual container – it will be the domain valid for this basket that is retained (cf. Chapter 3.3.6 Coding of topics). If in the case of a generic coordinate domain not one specific but a selection of possible concrete coordinate domains has been defined, it is the domain valid for a certain basket that is retained upon transferring this basket (cf. chapter 3.3.6 Coding of topics). The fact that this definition has been deferred must be indicated with the topic (cf. Chapter 2.5.2 Topics). The final applicable concrete coordinate domain is valid for all uses of the generic coordinate domain and all uses of line domains based upon the generic coordinate domain.

Syntax rule:

```
ContextDef = 'CONTEXT' { Context-Name '=' { GenericCoordDef-DomainRef '='
               Concrete-DomainRef { 'OR' Concrete-DomainRef } ';' } }.
```

The context name is of no significance; however, in accordance with general rules uniqueness is mandatory and serves e.g. for eventual error messages. A context affects the model in which it has been defined and in all models importing directly or indirectly the defining model.

```
CONTEXT default =
  MyModel.Coord2 = GeometryCHLV03.Coord2 OR GeometryCHLV95.Coord2;
```

Whenever a context definition already applies for a certain generic coordinate domain, a new context definition for the same generic coordinate domain will result in a new determination of the concrete domain. However, these new domains must be specializations of existing determinations.

```
DOMAIN
  Coord2Special EXTENDS GeometryCHLV03.Coord2 = COORD … ;

CONTEXT default =
  MyModel.Coord2 = Coord2Special;
```

2.8.9 Domains of object identifications

Identifiable objects are always labeled with an object identification. In order to make it clear to the system what storage has to be supplied and how these object identifications have to be generated, corresponding domains can be defined and assigned to topics resp. classes (cf. chapter 2.5.2 Topics and chapter 2.5.3 Classes and structures). For the administration of object identifications, mainly also of baskets, it makes sense to have ordinary attributes with such domains.

Syntax rule:

```
OIDType = 'OID' ( 'ANY' | NumericType | TextType ).
```

INTERLIS 2 itself defines the following OID-domains (cf. appendix A *The internal INTERLIS-data model*):

```
DOMAIN
  NOOID = OID ANY;                                !! arbitrary, non-stable
                                                  !! identifier
  ANYOID (ABSTRACT) EXTENDS NOOID = OID ANY;
  I32OID EXTENDS ANYOID = OID 0 .. 2147483647;    !! positive, 4 Bytes integer
                                                  !! values
```

```
STANDARDOID EXTENDS ANYOID = OID TEXT*16;     !! according to appendix F
                                              !! (only numbers and letters)
UUIDOID EXTENDS ANYOID = OID TEXT*36;         !! according to ISO 11578
```

It is not possible to extend an OID-definition other than a NOOID being extended by an ANY-OID, and an ANYOID being extended by a concrete definition (not OID ANY).

If ANYOID is used for abstract topics, resp. classes, it is required to expect an object identification whose exact definition however is to remain open. Otherwise ANYOID can only be used as an attribute domain. Consequently, it is not only the OID itself that belongs to the attribute value, but also the concrete OID domain. OID-values of textual OID-domains must fulfill the XML-ID rule in chapter 3.3.1: the first character must be a letter, a number or an under-score, followed by letters, numbers, dots, hyphens, under-scores, no columns (!).

2.8.10 Blackboxes

By using this data type it is possible to model attributes whose contents cannot be specified. The XML-version describes an attribute with XML-content and the BINARY version a binary content. This type cannot be refined in extensions.

Syntax rule:

```
BlackboxType = 'BLACKBOX' ( 'XML' | 'BINARY' ).
```

2.8.11 Domains of classes and attribute paths

It may make sense that data objects contain references to certain classes and attributes:

Syntax rules:

```
ClassType = ( 'CLASS'
                [ 'RESTRICTION' '(' ViewableRef
                                    { ';' ViewableRef } ')' ]
            | 'STRUCTURE'
                [ 'RESTRICTION' '(' ClassOrStructureRef
                                    { ';' ClassOrStructureRef } ')' ] ).

AttributePathType = 'ATTRIBUTE'
                    [ 'OF' ( ClassType-AttributePath
                           | '@' Argument-Name ) ]
                    [ 'RESTRICTION' '(' AttrTypeDef
                                        { ';' AttrTypeDef } ')' ].

ClassConst = '>' ViewableRef.

AttributePathConst = '>>' [ ViewableRef '->' ] Attribute-Name.
```

By indicating structure any structure or class, by indicating class (also permissible as extension of STRUCTURE) any class (but no structures) are admitted. If only certain structures, resp. classes and their extensions are to be admitted, these have to be listed (RESTRICTION). In extensions all admissible structures, resp. classes have to be listed again. They cannot contradict the basic definition. As soon as such restrictions have been defined, STRUCTURE no longer can be extended by CLASS.

By means of indicating ATTRIBUTE a certain attribute path type is admitted. It may be stated that it should belong to a class (not a subclass!) in accordance with another definition (OF). It is possible to refer either to a ClassType-Attribute or as in the case of the definition of a function (cf. chapter 2.14 Functions) to a different argument. Furthermore all possible attribute types can be restricted (RESTRICTION). The following are suitable as constants: names of the attributes of classes, structures, associations and views. The corresponding class name can be stated explicitly or is derived from the context resp. from the reference to another attribute or another argument (OF).

2.8.12 Line strings

2.8.12.1 Geometry of the line string

Practically a *curve segment* is a 1-dimensional structure which has no splits, no corners and no double points of any type (see figures 10 and 11). Curve segments are smooth and unique. Straight line segments, circle arcs, segments of parabolas and clothoides are examples of curve segments. Every curve segment has two *boundary points* (start point and end point, which are not allowed to be identical. The other points of a curve segment are called *inner points*. These form the *interior* of the curve segment.

Exact definition (mathematical terms which are not explained here but whose definition can be found in textbooks are written *"in italics and in quotes"*): *Curve segment* means a subset of the *"3-dimensional" "Euclidian space"* (hereafter *space* for short), which is the *"image set"* of a *"smooth"* and *"injective" "mapping"* of an *"interval"* (of the *"numerical straight line"*). Start point and end point of the curve segment are the images of the end point of the interval. *Planar curve segment* means a curve segment lying in a *plane* ("2-*dimensional "subspace"* of the space).

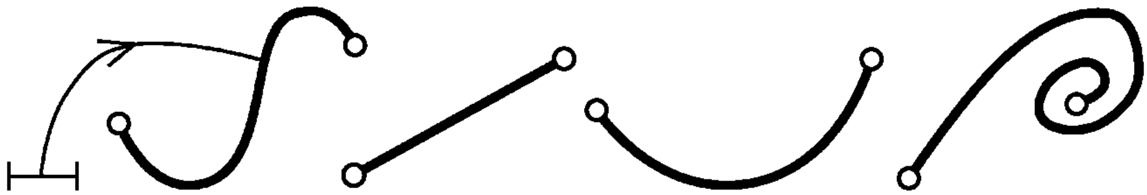
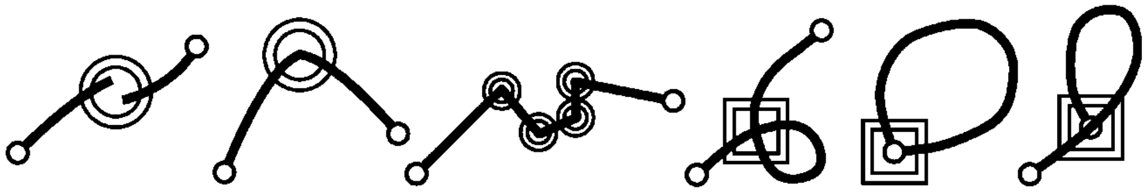


Figure 10: Examples of planar curve segments.



.

Figure 11: Examples of planar sets not being curve segments (a double circle indicates "not smooth" and a double square "not injective").

A line string is a finite sequence of curve segments. Except for the first curve segment, the start point of every curve segment corresponds to the end point of the preceding curve segment. These points are called *control points* of the line string. Practically a line string can have multiply used curve segments, curve segments with common base points, intersecting curve segments and curve segments starting or ending in the interior of other curve segments (see figures 12 and 13). A *simple line string* contains no self-intersection points (see figure 14). In addition, for a *closed simple line* the start point of the first curve segment and the end point of the last curve segment are identical.

Exact definition (mathematical terms which are not explained here but whose definition can be found in textbooks are written *"in italics and in quotes")*: *Line strings* means a subset of the space, which is the *"image set"* of a *"continuous"* and *"partially smooth"* (but not necessarily *"injective") "mapping"* of an *"interval"* (the so-called *characteristic mapping*) and which contains only a finite number of *"non smooth points"*. A *"non smooth point"* is called *vertex*. With a *closed line string* start and end point are identical. A line string, whose characteristic mapping is also *"injective"*, except for its start and end point that are identical, is called a *simple line string*.

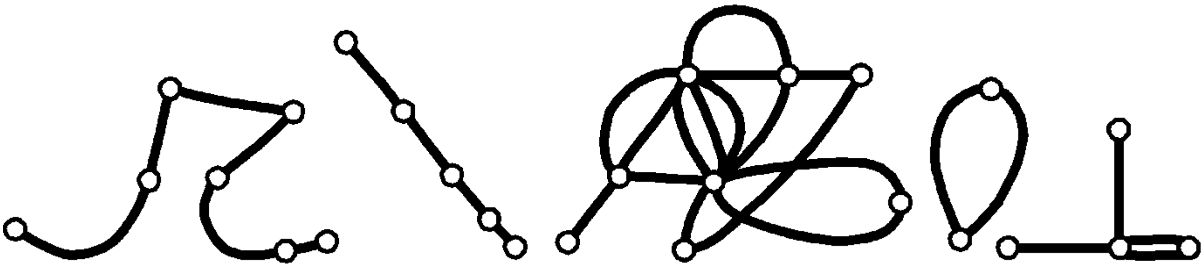


Figure 12: Examples of planar line strings.

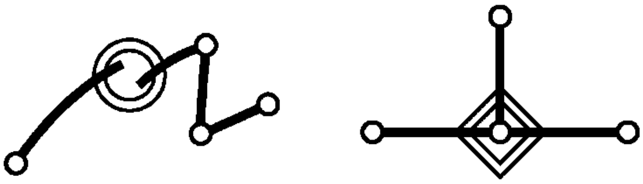


Figure 13: Examples of planar sets that are not line strings (the double circle means "not continuous" and the double rhombus "not image of an interval").

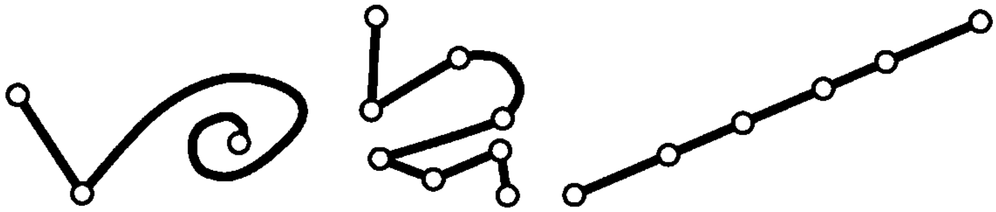


Figure 14: Examples of (planar) simple line strings.

2.8.12.2 Line strings with straight line segments and circle arcs as predefined curve segments

INTERLIS 2 acknowledges the following line strings: directed (DIRECTED POLYLINE), undirected (POLYLINE) or a set of undirected (MULTIPOLYLINE). However, with an individual line string ([DIRECTED] POLYLINE) also MULTIPOLYLINE are admissible when transferring a part only (cf. chapter 3.3.6 Coding of topics). When dealing with a set of line strings (MULTIPOLYLINE) the individual line strings need not be connected. Line strings may also be used within the scope of surfaces and tessellations (cf. chapter 2.8.13 Surfaces and tessellations).

The definition of a concrete domain of a line string always requires the specification of the admissible forms of curve segments by means of enumeration, e.g. straight line segments (keyword STRAIGHTS), circle arcs (keyword ARCS) or other possibilities (cf. chapter 2.8.12.3 Other forms of curve segments) and furthermore the indication of the domain of the vertices. Within an abstract domain of a line string these specifications may be omitted. The following rules apply for domain extensions:

- A line can only be reduced but not completed by new types.
- The coordinate domain indicated within the scope of a line string value domain must be a restriction of the coordinate domain of the original line string value domain, provided the latter has been defined.

Curve segments are always considered an extension of the basic structure 'LineSegment'. The coordinate domain applied therein is the same as the one defined in the definition of the line string.

```
STRUCTURE LineSegment (ABSTRACT) =
  SegmentEndPoint: MANDATORY LineCoord;
END LineSegment;

STRUCTURE StartSegment (FINAL) EXTENDS LineSegment =
END StartSegment;

STRUCTURE StraightSegment (FINAL) EXTENDS LineSegment =
END StraightSegment;

STRUCTURE ArcSegment (FINAL) EXTENDS LineSegment =
  ArcPoint: MANDATORY LineCoord;
  Radius: NUMERIC [LENGTH];
END ArcSegment;
```

The first curve segment of a line string is always a start segment. The start segment only consists of the start point itself, which at the same time is the end point of the start segment. The straight-line segment has an end point and thereby determines a straight from the end point of the predecessor curve segment to its end point. Both start segment and straight line segment do not require any further specifications. Thus the corresponding extensions of the LineSegment are void. Two successive vertices (SegmentEndPoints) may not coincide in the projection.

A circle arc segment describes a curve segment that appears as a true circle arc segment in the projection. In addition to the end point an intermediate point describes the circle arc segment. It is only of significance in connection with the geometry. With 3-dimensional coordinates we use linear interpolation for the height on the circle arc segment. You may imagine this curve as the thread of a cylindrical screw in perpendicular position on the projection plane.The intermediate point is not a vertex of the line string. It should be positioned as exactly as possible in the middle between start and end point. The intermediate point ist to be indicated with at least the same degree of precision (number of decimals) as the vertices. In order to avoid numeric problems (see below) it is recommended to transfer vertices whenever possible with additional precision. Nevertheless the calculated radius may differ considerably from the actual radius. If the actual radius is indicated, it remains decisive for the arc definition. The intermediate point will only determine which one of the four possible arcs will be selected. However, even in this case the intermediate point may only differ a maximum of one precision unit (the value 1 of the last decimal according to domain definition; e.g. 0.001 in the case of coordinates with 3 decimals) from the trace of the arc calculated from the radius. Often from a professional point of view we would like to express the fact that a line string is a simple line string, in other words without self-interjections and, more particularily, that multiple use of the same curve segment is impossible (keyword WITHOUT OVERLAPS). Several concrete systems support this specific requirement whereby a technical problem occurs: In boundary situations (especially in the context of arcs) different systems will come up with different results with regard to overlaps due to differences in precision and calculating methods. Hence we should always expect one system to tolerate or on the other hand to critize small overlaps, whereas another system will show different results. "Small" in this context signifies that the potential overlap is inferior to the value 1 of the last decimal according to domain definition multiplied by half the square root of 2 [e.g. 0.001 * sqrt(2) / 2 in the case of coordinates with 3 decimals].

Especially in the case of data transfer where the data originally was processed graphically, it may make sense from a professional point of view to even tolerate significant overlaps (i.e. several centimeters) in order to avoid an extraordinary effort in updates.

In order to accommodate both requirements the following rule concerning overlap tolerance applies:

If an arc and a line (respectively another arc) being succeeding curve segments of a line string, not only have a common vertex but also a common interior point as in figure b) (not vertex), this is also admitted with a single line string, provided the arc segment dissected from the line (resp.

the double-arc segment dissected from the other arc) does not show a height parameter of arrow inferior or equal to the defined tolerance. Where no tolerance is indicated, a value applies according to technical considerations (see above). With an explicit indication (the decimal following WITHOUT OVERLAPS must be greater than 0) it must be stated with the same numeric precision as the one employed for vertex coordinates and must have a value greater than 0.

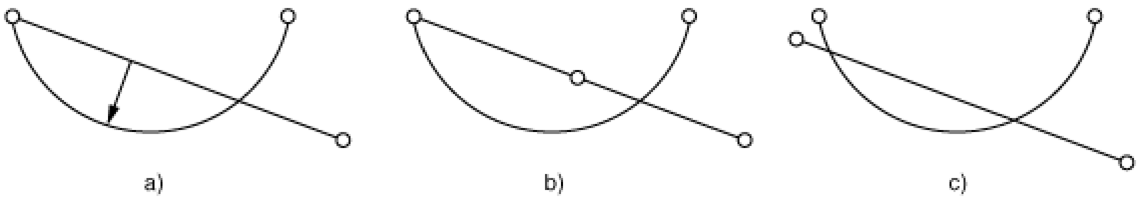


Figure 15: a) Height parameter (of arrow) may not exceed the given tolerance; b) inadmissible overlap of polylines since another vertex is situated between vertex and intersection; c) inadmissible overlap of polylines since there exists no common vertex.

With individual surfaces and area tessellation overlap tolerance is mandatory. Therefore when applying implicit tolerance it is possible to omit WITHOUT OVERLAPS.

No tolerance indication (explicit or implicit) ma be overridden. Within the scope of value domain definitions and attribute extensions, undirected string lines can be extended into directed string lines (cf. chapter 2.8.13.4 Extensibility).

When line strings are directed, their direction always has to be conserved (even when transferring data).

For vertices the value domain of the coordinates is defined. By means of the existence constraint REQUIRED IN (cf. chapter 2.12 Constraints and chapter 2.13 Expressions) it is further possible to demand that coordinates may not be arbitrary but have to correspond to the points of certain classes.

If the coordinate type of the vertices is abstract, then the line string has to be declared abstract as well.

Syntax rules:

```
LineType = ( [ 'DIRECTED' ] 'POLYLINE' | 'SURFACE' | 'AREA' |
             [ 'DIRECTED' ] 'MULTIPOLYLINE' | 'MULTISURFACE' | 'MULTIAREA' )
           [ LineForm ] [ ControlPoints ] [ IntersectionDef ].

LineForm = 'WITH' '(' LineFormType { ',' LineFormType } ')'.

LineFormType = ( 'STRAIGHTS' | 'ARCS'
               | [ Model-Name '.' ] LineFormType-Name ).

ControlPoints = 'VERTEX' CoordType-DomainRef.

IntersectionDef = 'WITHOUT' 'OVERLAPS' [ '>' Dec ].
```

Lines may be based upon generic coordinates without being declared generic themselves. Their concrete definition depends upon the defintion of the coordinate domain (within the scope of a context definition resp. a data transfer). It is not necessary (but definitively admissible) that there are, in the instance of line domains based upon generic coordinate domains, definitions of line domains based upon concrete coordinate domains.

```
DOMAIN
  Coord2 (GENERIC) = COORD NUMERIC, NUMERIC;
  Line = POLYLINE WITH (STRAIGHTS, ARCS) VERTEX Coord2;
```

2.8.12.2.1 Other forms of curve segments

Besides straight line segments and circle arcs it is possible to define other forms of curve segments. Besides their names it also has to be specified according to which structure a curve segment is described.

Syntax rule:

```
LineFormTypeDef = 'LINE' 'FORM'
                  { LineFormType-Name ':' LineStructure-Name ';' }.
```

A line structure must always be an extension of the LineSegment as defined by INTERLIS (cf. chapter 2.8.12.2 Line strings with straight line segments and circle arcs as predefined curve segments).

2.8.13 Surfaces and tessellations

2.8.13.1 Geometry of surfaces

In most cases planar surfaces are sufficient for the modeling of geo-data. In addition INTERLIS also supports planar general surfaces. Practically a planar general surface is limited by one exterior and possibly by one or several interior boundaries (see figure 20). The boundary lines themselves must consist of simple line strings that from a geometrical point of view can be combined into closed simple line strings. Furthermore they must be positioned in such a manner that from any point in the interior of the surface to any other point in the interior of the surface there exists a way that neither intersects a boundary line nor contains vertices of a boundary line (see figure 19). As long as this restriction is not violated, boundaries may meet in vertices. In such situations there are several conceivable possibilities that would allow dividing the boundary of the surface as a whole into individual line strings (see figure 22). INTERLIS does not insist on one specific possibility. If such a surface is transferred several times, a different possibility may occur in each different transfer.

> Exact definitions (mathematical terms which are not explained here but whose definition can be found in textbooks are written *"in italics and in quotes"*):
>
> *Surface element* means a subset of the *space*, which is the *"image set"* of a *"smooth"* and *"injective" "mapping"* of a *"planar" "regular polygon"* (see figures 16 and 17).
>
> *Surface* means the union *F* of a finite number of surface elements, which are *"connected"* and comply with the following condition: For every point *P* of the surface there exists a *"neighborhood",* which is a *deformation* (i.e. a *"homeomorph mapping")* of a planar regular polygon. If point *P* is a deformation of a boundary point of the polygon, it is called *boundary point of F*, otherwise *inner point of F.* It holds: the *"boundary"* (i.e. the set of all boundary points) of a surface is the union of a finite number of curve segments, which meet only in boundary points (start or end points). A *planar surface* is a surface being a subset of a *plane*. It holds: The boundary of a *"simple continuous"* planar surface (graphically speaking: a surface without any holes) is a simple closed line string, a so-called *outer boundary*. The boundary of a *"n-times continuous"* planar surface (graphically speaking: a surface with *n-1* holes) consists of the corresponding outer boundary and *n-1* other simple closed line strings (the so-called *inner boundaries*). The outer boundary and all the inner boundaries have no common points. A surface part cut out by an inner boundary is called an *enclave* (see figures 18, 19 and 20).
>
> A *general surface* is a surface with an additional finite number of *singular points* but with *"connected" interior* (set of inner points). A point is called *singular point* if there exists a

deformation of the point together with a *"neighbourhood"* into a planar *propeller set*, the point itself into the *center*. *Propeller set* means the union of a finite number of triangle surfaces meeting at exactly one point called *center. Planar general surface* means a general surface being subset of a *plane* (see figure 21). It holds: There are different possibilities to compose the boundary of a planar general surface by a finite number of closed planar line strings which have a maximum of a finite number of common points and each a maximum finite number of double points (see figure 22).

Figure 16: Examples of surface elements.

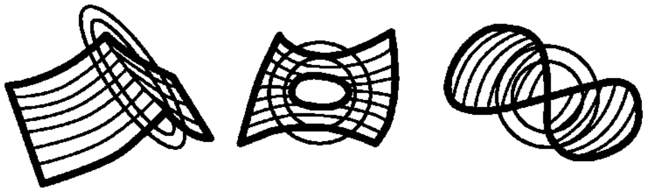


Figure 17: Examples of point sets in space, which are not surface elements (here a double circle means "not smooth").

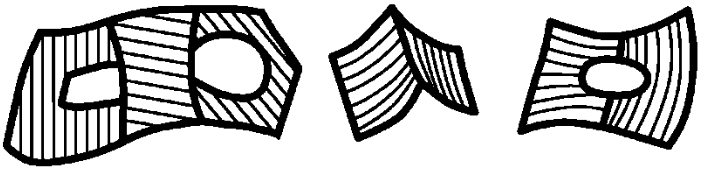


Figure 18: Examples of surfaces in the space.

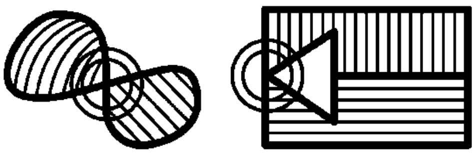


Figure 19: Examples of planar point sets that are not surfaces (a double circle marks a "singular point").

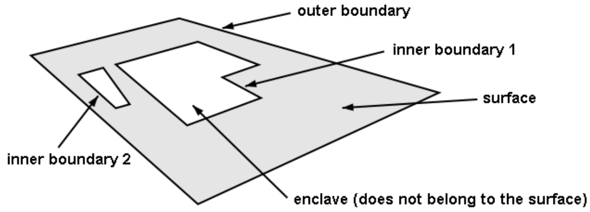


Figure 20: Planar surface with boundaries and enclaves.

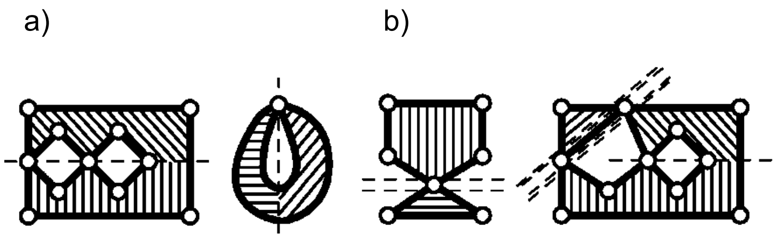


Figure 21: a) Examples of planar general surfaces; b) Examples of planar sets that are not general surfaces, because their interior is not connected. But these planar sets can be subdivided into general surfaces ("---" shows the subdivision into surface elements and "===" the subdivision into general surfaces).

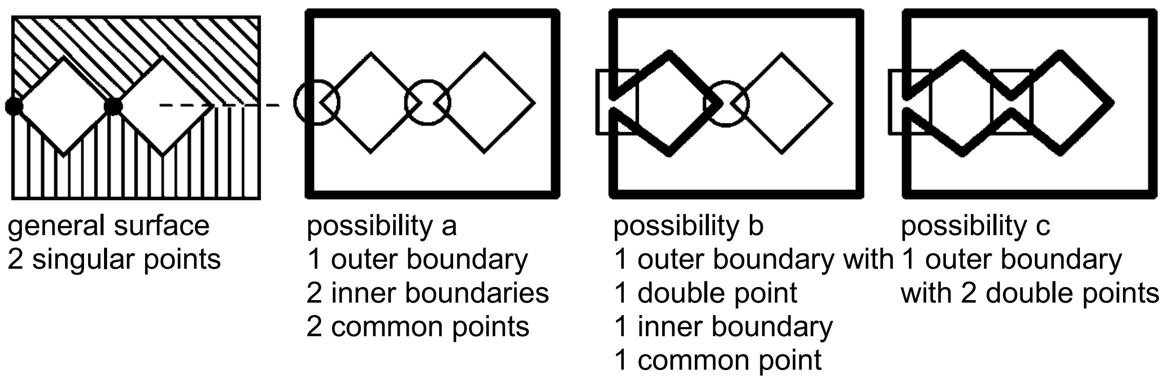


Figure 22: Different possible subdivisions of the boundary of a general surface.

Along with the definition of (general) surfaces, resp. (general) surfaces of a tessellation we determine beyond which tolerance curve segments of the boundary may not overlap (without indication of WITHOUT OVERLAPS it is the implicit resp. inherited tolerance that applies). With surfaces, prohibition of overlaps, resp. intersection does only apply to curve segments of one individual line string but to all curve segments of all line strings of the surface boundary. In the case of surfaces of a tessellation it even applies to all line strings connected with the tessellation. Furthermore and in accordance with the definition of the (general) surface, line strings that are not part of the boundary of a (general) surface are excluded.

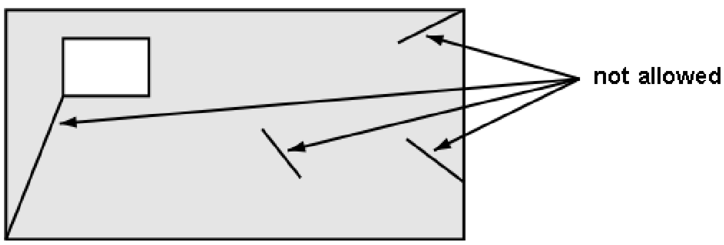


Figure 23: Disallowed boundary configurations for tessellations.

2.8.13.2 Surfaces

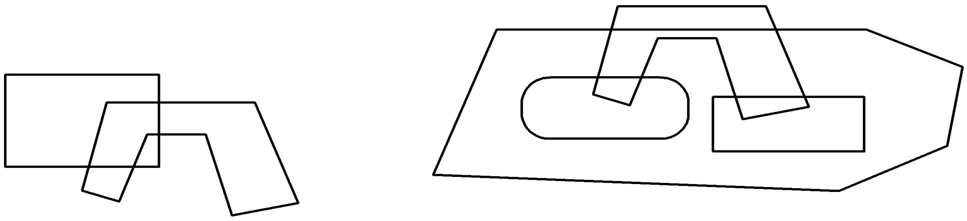


Figure 24: Individual surfaces (SURFACE).

For surfaces that partially or entirely overlap, i.e. which do not only have boundary points in common, the geometrical attribute type SURFACE is available (see figure 24). This type is called surface. A surface consists of one outer and possibly several inner boundaries (around the enclaves). Each boundary consists of at least one line string.

By using the attribute type MULTISURFACE it is possible to define a set of individual surfaces as one single value.The individual surfaces of a MULTISURFACE value may not have common boundaries and may not overlap unless such overlaps are defined.

2.8.13.3 Surfaces of a tessellation

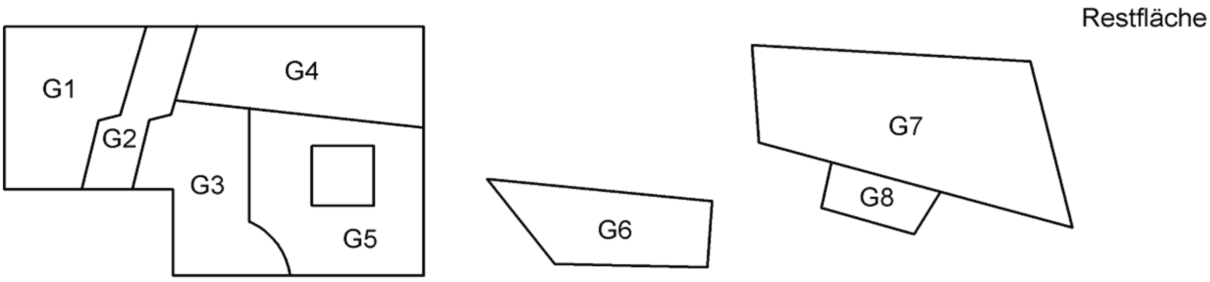


Figure 25: Tessellation (AREA).

(Planar) tessellation signifies a finite set of (general) surfaces and environments that cover a layer without overlaps.

For tessellations the geometrical attribute type AREA is at your disposal.

A maximum of one surface of the tessellation (or exactly one with the additional keyword MANDATORY), but never its environment, is assigned to the area object. It is not admissible that each of two surfaces of the tessellation with a common boundary does not correspond to an area object.

Thus each individual area object corresponds to a surface. As a result the same implicit structure applies to surfaces and to area objects. However additional consistency constraints apply:

- String lines of a tessellation must always be true boundaries. Hence there are no line strings with the same surface on either side (see figure 23). This is also ruled out by the definition of a surface.
- If there are defined area objects on either side of a line string, then each curve segment (join between two vertices) of one area object must in terms of geometry and attributes correspond exactly to the curve segment of the other area object.

Tessellations may not occur within substructures.

In order to be able to refer to line strings of a tessellation as individual objects (and namely as one object even if the line string is the boundary of two area objects), then AREA INSPECTION is at your disposal (cf. chapter 2.15 Views).

2.8.13.4 Extensibility

Surfaces may be extended into areas. The extension of a string line into an area is inadmissible, since with a surface several string lines have to be expected, whereas the definition of the line string only implies one line string.

Independent surfaces and surfaces of a tessellation can be extended in two respects:

- When it is 'SURFACE' that is primarily defined and hence overlaps are allowed, this can be replaced by 'AREA' in extensions, since this will not violate the basic definition.

2.9 Units

Units are always described as a term and (in [ ] brackets) a contraction. Both term and contraction must be names. Where the contraction is omitted it is the same as the term itself. Depending on the type of unit further specifications may follow. In actual use of a unit it is always the contraction that occurs. Hence the term itself is only of documentary character.

2.9.1 Base units

Base units are meters, seconds etc. They need not be specified any further. However base units can also be defined as abstract (keyword ABSTRACT), if the unit itself is yet unknown, but the matter described is clear (e.g. temperature, money). No contraction is assigned to abstract units. Concrete units cannot be extended.

Examples:

```
UNIT
  Length (ABSTRACT);
  Time (ABSTRACT);
  Money (ABSTRACT);
  Temperature (ABSTRACT);
  Meter [m] EXTENDS Length;
  Second [s] EXTENDS Time;
  SwissFranc [CHF] EXTENDS Money;
  Celsius [C] EXTENDS Temperature;
```

INTERLIS itself defines the abstract unit ANYUNIT. All other units inherit this directly or indirectly (cf. chapter 2.10.3 Reference systems). The following units have been defined directly (internally) by INTERLIS:

```
UNIT
  ANYUNIT            (ABSTRACT);
  DIMENSIONLESS      (ABSTRACT);
  LENGTH             (ABSTRACT);
  MASS               (ABSTRACT);
  TIME               (ABSTRACT);
  ELECTRIC_CURRENT   (ABSTRACT);
  TEMPERATURE        (ABSTRACT);
  AMOUNT_OF_MATTER   (ABSTRACT);
  ANGLE              (ABSTRACT);
  SOLID_ANGLE        (ABSTRACT);
  LUMINOUS_INTENSITY (ABSTRACT);
  MONEY              (ABSTRACT);

  METER [m]          EXTENDS LENGTH;
```

```
  KILOGRAM [kg]       EXTENDS QUANTITY;
  SECOND [s]          EXTENDS TIME;
  AMPERE [A]          EXTENDS ELECTRIC_CURRENT;
  DEGREE_KELVIN [K]   EXTENDS TEMPERATURE;
  MOLE [mol]          EXTENDS AMOUNT_OF_MATTER;
  RADIAN [rad]        EXTENDS ANGLE;
  STERADIAN [sr]      EXTENDS SOLID_ANGLE;
  CANDELA [cd]        EXTENDS LUMINOUS_INTENSITY;
```

Remark: In appendix H *Definition of units* the most common units have been assembled in an extended type model.

2.9.2 Derived units

Multiplying or dividing derived units with constants or functions can convert them converted into different units. Example:

```
UNIT
  Kilometer [km] = 1000 [m];
  Centimeter [cm] = 1 / 100 [m];
  Inch [in] = 0.0254 [m];         !! 1 inch equals 2.54 cm
  Fahrenheit [oF] = FUNCTION // (oF + 459.67) / 1.8 // [K];
```

Data in kilometers have to be multiplied by one thousand to become the equivalent in meters. Data in inches have to be multiplied by 2.54 to become the equivalent in centimeters. Add 456.67 to data in degrees Fahrenheit and then divide the result by 1.8 in order to calculate the same temperature in degrees Kelvin.

A derived unit automatically is considered an extension of the same abstract unit it can be converted into.

2.9.3 Combined units

Combined units (e.g. km per hour) are the result of a multiplication or division of other units (basic units, derived or combined units). Combined units can also be defined as abstract units. They then must refer entirely to other abstract units.

The units used in the concrete extension must therefore be a concrete extension of the units appearing in the abstract definition. Example:

```
UNIT
  Velocity (ABSTRACT) = ( Length / Time );
  Kilometer per hour [kmph] EXTENDS Velocity = ( km / h );
```

Syntax rules:

```
UnitDef = 'UNIT'
            { Unit-Name
                [ '(' 'ABSTRACT' ')' | '[' UnitShort-Name ']' ]
                [ 'EXTENDS' Abstract-UnitRef ]
                [ '=' ( DerivedUnit | ComposedUnit ) ] ';' }.

DerivedUnit = [ DecConst { ( '*' | '/' ) DecConst }
              | 'FUNCTION' Explanation ] '[' UnitRef ']'.

ComposedUnit = '(' UnitRef { ( '*' | '/' ) UnitRef } ')'.

UnitRef = [ Model-Name '.' [ Topic-Name '.' ] ] UnitShort-Name.
```

2.10 Dealing with meta objects

2.10.1 General comments concerning meta objects

In the sense of INTERLIS 2 meta objects are objects that are of significance within the scope of descriptions of application models. This is of use in reference systems and graphic symbols.

The construction of reference system objects or symbol objects must be defined with the usual means of classes and structures in either a REFSYSTEM MODEL or SYMBOLOGY MODEL. Reference system classes, axis classes, resp. symbol classes must be extensions of the classes COORDSYSTEM, REFSYSTEM, AXIS, resp. SIGN supplied by INTERLIS. These again are extensions of the class METAOBJECT. This class possesses an attribute name that must be unequivocal within the scope of all meta objects of one basket.

For the description itself of a practical model the meta object as such is not needed. It is enough to know the name as the representative of the meta object. Nevertheless for a running system as a rule the meta objects should be known completely, i.e. with all their attributes. Hence it must be clear for any run time system which basket contains a meta object of a certain name. Before meta objects (resp. their names) can to be used in models, they have to be declared. In order to achieve this, a basket name is introduced, the topic presumed is stated, and then (per class) the names of the expected objects are enumerated (rule MetaDataBasketDef). The basket name can also be defined as the extension of a name already introduced (EXTENDS). Consequently the corresponding topic must be the same as with the basic name or an extension of it.

If a meta object is referred to (rule MetaObjectRef) under the extended basket-name, then the name of the meta object must figure either there or in an inherited definition. Any run time system will first try to find the meta object in its corresponding basket. If no such meta object is found the search continues in accordance with the basket-names that have been directly or indirectly inherited. Thus meta objects can be prepared and refined on several levels. For example to begin with common graphic symbols can be defined, which later on are refined and supplemented at regional level. How to refer to the concrete basket depending on the basket-name is up to the run time system employed.

In a translated application model (cf. chapter 2.5.1 Models) a basket-name can be assigned to a concrete basket that only contains translations (METAOBJECTS_TRANSLATION objects; cf. appendix A *The internal INTERLIS-data model*), thereby translating those meta objects that have been introduced by the corresponding basket in the model taken as a basis. If this model itself is a translation, it is also possible to assign to this basket name a concrete basket that only contains translations. If the model is not a translation, then the basket name can be assigned a concrete basket that contains no translations (i.e. no METAOBJECT_TRANSLATION objects.

Syntax rules:

```
MetaDataBasketDef = ( 'SIGN' | 'REFSYSTEM' ) 'BASKET' Basket-Name
                      Properties<FINAL>
                      [ 'EXTENDS' MetaDataBasketRef ]
                        '~' TopicRef
                      { 'OBJECTS' 'OF' Class-Name ':' MetaObject-Name
                        { ',' MetaObject-Name } } ';'.

MetaDataBasketRef = [ Model-Name '.' [ Topic-Name '.' ] ] Basket-Name.

MetaObjectRef = [ MetaDataBasketRef '.' ] Metaobject-Name.
```

If in the current context only one meta data basket name is necessary, the reference to the meta data basket (MetaDataBasketRef) must not be indicated in the meta object reference.

2.10.2 Parameters

By means of parameters those properties can be designated in the meta model that do not concern the meta object itself but its use within the application. Their definition is introduced by the keyword PARAMETER and it is constructed in a manner similar to the definition of attributes.

2.10.2.1 Parameters for reference and coordinate systems

For reference and coordinate systems, resp. their axis only the predefined parameter Unit is admissible.

If you refer to a reference or coordinate system (rule RefSys) within the definition of a numeric data type (cf. chapter 2.8.5 Numeric data types) or a coordinate type (cf. chapter 2.8.8 Coordinates), then a unit must be indicated which is compatible with the unit of the corresponding axis of the coordinate system, resp. the sole axis of the reference system (cf. chapter 2.10.3 Reference systems).

2.10.2.2 Parameters of symbols

Definitions of parameters of symbols are arbitrary. These parameters correspond to the specifications (e.g. point symbol identification, position, rotation) that have to be supplied to a graphic system in order to enable graphic representation. To start with the symbol has to be selected. This is done by defining a parameter for the reference to the symbol class within which the parameter is defined (METAOBJECT). Such a parameter of a symbol may also be a reference to another metaobject (METAOBJECT OF). In both cases a metaobject-reference will be indicated within the application (cf. chapter 2.16 Graphic descriptions), i.e. both basket reference-name and meta object-name will be indicated. Thus the respective tool can determine the real basket-identification and meta object-identification.

Besides these special cases of parameters, parameters can be defined in the same way as attributes.

Syntax rule:

```
ParameterDef = Parameter-Name Properties<ABSTRACT,EXTENDED,FINAL>
                   ':' ( AttrTypeDef
                       | 'METAOBJECT' [ 'OF' MetaObject-ClassRef ] ) ';'.
```

2.10.3 Reference systems

Without further specifications numeric values and coordinates indicate differences, they do not have a defined absolute reference. To achieve this, a coordinate system, resp. a scalar system must be defined. The model definition will be executed in a REFSYSTEM MODEL. In INTERLIS the following classes are at your disposal:

```
CLASS METAOBJECT (ABSTRACT) =
  Name: MANDATORY NAME;
UNIQUE Name;
END METAOBJECT;

STRUCTURE AXIS =
  PARAMETER
    Unit: NUMERIC [ ANYUNIT ];
END AXIS;

CLASS REFSYSTEM (ABSTRACT) EXTENDS METAOBJECT =
END REFSYSTEM;
```

```
CLASS COORDSYSTEM (ABSTRACT) EXTENDS REFSYSTEM =
  ATTRIBUTE
    Axis: LIST {1..3} OF AXIS;
END COORDSYSTEM;

CLASS SCALSYSTEM (ABSTRACT) EXTENDS REFSYSTEM =
  PARAMETER
    Unit: NUMERIC [ ANYUNIT ];
END SCALSYSTEM;
```

In extensions it is possible to add other typical attributes for this kind of scalar and coordinate systems and even the unit can be replaces by extensions (syntax for parameter definitions cf. chapter 2.10.2 Parameters and chapter 2.5.3 Classes and structures). However this unit will have to be compatible with a unit defined in the value domain (cf. chapter 2.9 Units).

2.11 Run time parameters

Besides the actual data and meta data, individual data elements can be defined that ought to be supplied by a system for processing, evaluating or graphic representation. They are called run time parameters.

Syntax rule:

```
RunTimeParameterDef = 'PARAMETER'
                         { RunTimeParameter-Name ':' AttrTypeDef ';' }.
```

Tessellations (keyword AREA) are not admissible for run time parameters.

Typical run time parameters are for example scale of representation or current date.

2.12 Constraints

Constraints serve to define restrictions the objects have to comply with.

Consistency constraints may have a name which must be unequivocal within the class, resp. the domain the definition refers to.

Constraints referring to one single object are described by a logical expression relating to the attributes of the objects (see next chapter). Such constraints may not use functions which expect a set of objects (OBJECTS OF) on an argument (typically the first). Therein it is possible to define constraints that are obligatory and apply peremptorily to all objects (keyword MANDATORY), and others which only apply as a general rule. In the latter case it is indicated which percentage of the instances of a class must normally comply with the constraint (rule PlausibilityConstraint).

By stating an existence-constraint (rule ExistenceConstraint) we require that the value of an attribute of each object of the constraint-class exists in a certain attribute of an instance of another class. This is only possible if the constraint attribute is compatible with the other attribute and has the following effects:

- If the domain of the constraint attribute equals the domain of the other attribute or one of its extensions, the constraint value must exist in the required attribute of another instance.
- If the domain of an attribute is a structure, all attributes contained therein are compared.
- If the domain of the other attribute is a coordinate or the domain of the constraint attribute a polyline, surface or tessellation with the same or extended coordinate domain, then all coor-

dinates of the vertices of the polyline, surface or tessellation must occur within the required attribute of another instance.

Uniqueness constraints are introduced with the keyword UNIQUE (rule UniquenessConstraint).

Conceptually signifies "globally" that all existing objects in any basket must fulfill these requirements. By indicating (BASKET) the uniqueness constraint no longer applies globally but per basket.

It is possible to demand that within a structure, class or association, a certain combination of attributes of sub-structures defined by BAG OF or LIST OF be locally (LOCAL) unequivocal, i.e. within the scope of all structural elements assigned to the current object or structural element.

Example:

```
CLASS A =
  K: (A, B, C);
  ID: TEXT*10;
UNIQUE K, ID;
END A;
```

If an attribute of undefined value is part of a uniqueness constraint (UNIQUE), the uniqueness constraint is considered fulfilled for the object in question. Structure attributes are not permitted to be used for uniqueness constraints.

If a consistency constraint (MANDATORY CONSTRAINT) cannot be calculated (e.g. because of an attribute of undefined value or because the divisor equals 0), the uniqueness constraint is considered fulfilled for the object in question.

With constraints that refer to the class (SET CONSTRAINT), it is possible to define conditions that will apply to the total amount of objects of the class (resp. the section that complies with the restricting WHERE-constraint). Such constraints must employ a minimum of one function expecting a set of objects (OBJECTS OF) on one argument (typically on the first). In order to hand over the total amount of objects of this class (resp. the section that complies with the WHERE-constraint) to this argument, ALL (without indicating its own class) is introduced. Since such consistency constraints do not refer to the individual object, it is impossible to touch upon the values of attributes. Hence for all further arguments and for comparisons only constants, run time parameters, class types as well as attribute types and other function calls which fulfill this restriction, are considered suitable.

Examples:

```
CLASS B =
  Type: (a, b, c);
  Geometry: SURFACE ...;
SET CONSTRAINT WHERE Type = #a :
  areAreas(ALL, UNDEFINED, >> Geometry);
END B;

STRUCTURE S =
  Geometry: SURFACE ...;
END S;

CLASS C =
  Surfaces: BAG OF S;
SET CONSTRAINT
  areAreas(ALL, >> Surfaces, >> S->Geometry);
END;
```

The objects of class B, whose type is a, should form an AREA since the standard function Areas (cf. chapter 2.14 Functions) is being used. All surfaces (in accordance with the sub-structure Surfaces) of the objects of class C form a tessellation.

More extensive constraints must be defined within views (e.g. a view which connects a certain class with itself) and thus allow to compare any attribute combination with all other objects of the class). It is peremptory that such views be defined within the data model.

Constraints that extend to a multitude of objects (above all uniqueness constraints) are not always entirely controllable, since this control can only be executed with locally available baskets. Nevertheless conceptually they apply globally (except with meta models, cf. appendix G *Uniqueness of user keys*).

Syntax rules:

```
ConstraintDef = ( MandatoryConstraint
                | PlausibilityConstraint
                | ExistenceConstraint
                | UniquenessConstraint
                | SetConstraint ).

MandatoryConstraint = 'MANDATORY' 'CONSTRAINT' [ Constraint-Name ':' ]
                         Logical-Expression ';'.

PlausibilityConstraint = 'CONSTRAINT' [ Constraint-Name ':' ]
                           ( '<=' | '>=' ) Percentage-Dec '%'
                             Logical-Expression ';'.

ExistenceConstraint = 'EXISTENCE' 'CONSTRAINT' [ Constraint-Name ':' ]
                         AttributePath 'REQUIRED' 'IN'
                           ViewableRef ':' AttributePath
                             { 'OR' ViewableRef ':' AttributePath } ';'.

UniquenessConstraint = 'UNIQUE' [ '(' 'BASKET' ')' ]
                         [ Constraint-Name ':' ]
                         [ 'WHERE' Logical-Expression ':' ]
                           ( GlobalUniqueness | LocalUniqueness ) ';'.

GlobalUniqueness = UniqueEl.

UniqueEl = ObjectOrAttributePath { ',' ObjectOrAttributePath }.

LocalUniqueness = '(' 'LOCAL' ')'
                    StructureAttribute-Name
                      { '->' StructureAttribute-Name } ':'
                              Attribute-Name { ',' Attribute-Name }.

SetConstraint = 'SET' 'CONSTRAINT' [ '(' 'BASKET' ')' ]
                  [ Constraint-Name ':' ]
                  [ 'WHERE' Logical-Expression ':' ]
                    Logical-Expression ';'.
```

It is possible to only subsequently define consistency constraints for a certain class or association, provided they be within the TOPIC (typically following the definition of an association).

Syntax rule:

```
ConstraintsDef = 'CONSTRAINTS' 'OF' ViewableRef '='
                   { ConstraintDef }
                 'END' ';'.
```

2.13 Expressions

Generally expressions (Expression) are used for example as arguments of functions or in constraints and selections. With logical expressions (Logical-Expression) the result type must be Boolean. Expressions refer to a context object (i.e. an object that constraints are formulated for. Proceeding from this object it is possible to refer to an attribute, a structure element, a function etc. Such items as well as comparison values such as constants und run time parameters are interlaced as factors in predicates. By means of operators predicates can be made into a expression.

Syntax rules:

```
Expression = Term.

Term = Term0 [ '=>' Term0 ].

Term0 = Term1 { ( 'OR' | '+' | '-' ) Term1 }.

Term1 = Term2 { ( 'AND' | '*' | '/' ) Term2 }.

Term2 = Predicate [ Relation Predicate ].

Predicate = ( Factor
            | [ 'NOT' ] '(' Logical-Expression ')'
            | 'DEFINED' '(' Factor ')' ).

Relation = ( '==' | '!=' | '<>' | '<=' | '>=' | '<' | '>' ).
```

Remarks concerning the significance of these syntax rules:

- In accordance with the syntax rules for terms the most forcible obligation is the comparison (relation), followed by AND resp. * and / and OR resp. + and -, followed by the implication (=>). AND and OR may only be used with boolean predicates, +, -, * and / may only be used with numeric predicates.
- NOT followed by the logical expression in brackets demands the negation of this expression.
- Comparison of factors. Depending on the type of factor, certain comparisons are inadmissible:
  - With string types only Equality (==) and Inequality (!=, <>) are admissible. Further comparisons have to be realized by means of functions. Above all it is of major importance to clearly define how regional particularities such as umlauts or accent marks should be treated.
  - With numeric data type and structured domains comparisons are defined as usual. More and less comparisons are not practical with circular data type.
  - Coordinates as a whole can only be examined with respect to their equality or inequality. All other comparisons are only available for their separate components (rule Factor).
  - With enumerations more and less comparisons are only admissible if the enumeration has been defined as ordered. Equivalence means exact equivalence. If all sub-elements of a node are included as well, then the function isEnumSubVal has to be used.
  - Lines can only be tested as to whether they are undefined (== UNDEFINED).
  - A factor can also designate an object. Then it is possible to examine it in terms of Definition, Equality and Inequality.
- If a factor not only consists of the item itself but also of the path leading to it, the item is always considered undefined provided any attribute of the path is undefined. Thus the built-in function DEFINED (a.b) is equal to (a.b != UNDEFINED).

– Any expression will be examined from left to right but only until its resulting value appears to be definite. In other words terms joined with OR will only be evaluated if the value up to this point is false. On the other hand terms joined with AND will only be evaluated if the value up to this point is true.

The implication (=>) is only false if the first term is true and the second term is false. a => b is identical with the following formulation: NOT(a) OR b. It is typically used when formulating limited constaints. For example: Geometry is only mandatory with a certain state:

Example:

```
CLASS A =
  State: (valid, projected);
  Geometry: SURFACE ...;
MANDATORY CONSTRAINT State== valid => DEFINED(Geometry);
END A;
```

Factors can be formed according to the following syntax rules:

```
Factor = ( ObjectOrAttributePath
         | ( Inspection | 'INSPECTION' Inspection-ViewableRef )
             [ 'OF' ObjectOrAttributePath ]
         | FunctionCall
         | 'PARAMETER' [ Model-Name '.' ] RunTimeParameter-Name
         | Constant ).

ObjectOrAttributePath = PathEl { '->' PathEl }.

AttributePath = ObjectOrAttributePath.

PathEl = ( 'THIS'
         | 'THISAREA' | 'THATAREA'
         | 'PARENT'
         | ReferenceAttribute-Name
         | AssociationPath
         | Role-Name [ '[' Association-Name ']' ]
         | Base-Name
         | AttributeRef ).

AssociationPath = [ '\' ] AssociationAccess-Name.

AttributeRef = ( Attribute-Name ( [ '[' ( 'FIRST'
                                        | 'LAST'
                                        | AxisListIndex-PosNumber ) ']' ] )
               | 'AGGREGATES' ).

FunctionCall = [ Model-Name '.' [ Topic-Name '.' ] ] Function-Name
                   '(' [ Argument { ',' Argument } ] ')'.

Argument = ( Expression
           | 'ALL' [ '(' RestrictedClassOrAssRef | ViewableRef ')' ] ).
```

Factors can refer to objects and their attributes. Step by step it is possible to set up entire object-paths within this procedure. Each construction opens the path from the at the time current object to the next. The first current object results from the context, e.g. an object of the class that a constraint is being defined for.

– THIS: Designates the so-called context-object, i.e. the current object of a class, a view or a graphic definition, which requires an object-path. THIS e.g. is to be indicated when calling a function that features ANYCLASS or ANYSTRUCUTURE as parameter.

- THISAREA and THATAREA: Designates an area object on whose common boundary the current line string object can be found. The application of THISAREA and THATAREA is only possible within the scope of the inspection of a tessellation (cf. chapter 2.15 Views).
- PARENT: Designates super structure-element or super-object of the current structure element or object. The view must be an ordinary inspection (no area inspection) (cf. chapter 2.15 Views).
- Indication of reference attributes: designates the object that is assigned to the current object that is assigned from the current object, resp. the current structure via the indicated reference attribute.
- Indication of role: The indication of role is valid provided one single corresponding role exists. The indication of role may either point to an initial role (according to which the current object is related to the association reference) or to a target role (according to which the association reference is related with the reference object). Whenever the indication of role is supplemented with the reference name, it can only point to initial roles. Depending on the position of the path element within the path the roles are searched for in a different manner. If the indication of role is the first path element within the path, then the role is searched for in all the relationship accesses within the class, provided the path can be used in their context. If the indication of role is a following element of the path, the role is searched for in all the associations available within the topic where the class is defined within whose context the path can be used. Only those associations will be taken into consideration that are related via roles with the class of the predecessor object of the path.
- Indication of basic view: By means of the (local) name of the basic view we designate the corresponding (virtual) object of the basic view in the current view, resp. in the current derived relationship.

When referring to an attribute we mean the value of the attribute of either the context-object or the object designated by the path. In addition all paths that end with an attribute are named attribute paths and can be used in different syntax rules independently of factors.

- Under ordinary circumstances it is sufficient to indicate the attribute name.
- When dealing with a coordinate attribute, we indicate the number of the axis in order to designate the corresponding component of the coordinate. The first component has index 1.
- The implicit attribute AGGREGATES is defined within aggregation views (cf. chapter 2.15 Views) and designates the set (BAG OF) of the aggregated base-objects.

In ordered sub-structures (LIST OF) individual elements can be approached. Admissible indices are:

- FIRST: the first element.
- LAST: the last element.
- Index number: The index indicated must be smaller than or equal the maximum number determined within the cardinality. The first element has index 1. If it is smaller than or equals the minimum number determined within the cardinality, there is always a corresponding element in existence; if it is greater the existence of such an element cannot be guaranteed. Subsequently the factor can be undefined.

A factor may also be an inspection (cf. chapter 2.15 Views). If it is preceded by an object path, then the object class thus given must either coincide with the object class of the inspection or be an extension of it. In order to belong to the set of structure elements supplied by the inspection, they have to belong to the object defined by the object path.

Factors can also be function calls. As their arguments we can consider:

– Expressions: The type of result of an expression must be compatible with the argument type.

– If by means of the expression an indication of role is being made, then the expression designates the set of target objects related via this role. With a formal parameter OBJECT OF or OBJECTS OF (only if based upon the model description it is evident, that only one target object is possible) must be required (cf. chapter 2.14 Functions).

– All objects (ALL) of the class within whose context the function call is being made or all objects of the class indicated. With a formal parameter OBJECTS OF must be required (cf. chapter 2.14 Functions). This always means all objects corresponding to this class or its extensions.

As comparison values the following items come into questions: function calls, run time parameters (cf. chapter 2.16 Graphic descriptions) and constants.

2.14 Functions

By means of its name, formal parameters as well as a short function description, a function is defined as an explanation. The names of the parameters are only of documentary value.

As formal parameters or function results we may consider:

– All those types admissible for attributes, above all also structures. Corresponding factors (rule Factor) come into question as arguments (i.e. current parameters).

– If a structure is indicated it is primarily structure elements that are in consideration as arguments. It is also possible to indicate object paths that lead to objects that are an extension of the structure. Above all with ANYSTRUCTURE it is possible to indicate any kind of object path.

– If OBJECT OF is indicated, then arguments can be all these objects that are attainable via object path and which correspond to the definition. Above all with OBJECT OF ANYCLASS it is possible to indicate any kind of object path. In a similar way as with references to other objects (cf. chapter 2.6.3 Reference attributes) it is possible to define the admissible super class and eventual restrictions and to specialize them in extensions.

– If OBJECTS OF is indicated, arguments may be sets of objects of a class. All objects of the set must comply with the requirements stated with the formal argument on the basis of the model description (in other words the requirement stated with the formal argument does not act as a subsequent filter).

– If ENUMVAL is indicated, arguments may be attributes or constants which designate a leaf of any kind of enumeration (cf. chapter 2.8.2 Enumerations).

– If ENUMTREEVAL is indicated, arguments may be attributes or constants which designate a node or a leaf of any kind of enumeration.

Syntax rules:

```
FunctionDef = 'FUNCTION' Function-Name
                '(' [ Argument-Name ':' ArgumentType
                { ';' Argument-Name ':' ArgumentType } ] ')'
                ':' ArgumentType [ Explanation ] ';'.

ArgumentType = ( AttrTypeDef
               | ( 'OBJECT' | 'OBJECTS' )
                     'OF' ( RestrictedClassOrAssRef | ViewRef )
               | 'ENUMVAL'
               | 'ENUMTREEVAL' ).
```

The following standard functions have been defined:

```
FUNCTION myClass (Object: ANYSTRUCTURE): STRUCTURE;
```

supplies the class of the object.

```
FUNCTION isSubClass (potSubClass: STRUCTURE; potSuperClass: STRUCTURE): BOOLEAN;
```

supplies true, if the class of the first argument of the corresponds to the class or a subclass of the second argument.

```
FUNCTION isOfClass (Object: ANYSTRUCTURE; Class: STRUCTURE): BOOLEAN;
```

supplies true if the object of the first argument belongs to the class or a subclass of the second argument.

```
FUNCTION elementCount (bag: BAG OF ANYSTRUCTURE): NUMERIC;
```

supplies the number of elements contained within the bag (or the list).

```
FUNCTION objectCount (Objects: OBJECTS OF ANYCLASS): NUMERIC;
```

supplies the number of objects appertaining to the given set of objects.

```
FUNCTION len (TextVal: TEXT): NUMERIC;
FUNCTION lenM (TextVal: MTEXT): NUMERIC;
```

supplies the length of text in terms of its number of symbols.

```
FUNCTION trim (TextVal: TEXT): TEXT;
FUNCTION trimM (TextVal: MTEXT): MTEXT;
```

supplies the text without blanks at beginning or end.

```
FUNCTION isEnumSubVal (SubVal: ENUMTREEVAL; NodeVal: ENUMTREEVAL): BOOLEAN;
```

supplies true, if SubVal is a sub-element, i.e. a sub-node or a leaf of the node NodeVal.

```
FUNCTION inEnumRange (Enum: ENUMVAL;
                      MinVal: ENUMTREEVAL;
                      MaxVal: ENUMTREEVAL): BOOLEAN;
```

supplies true, if the enumeration belongs to Enum, is ordered and placed within the range of MinVal to MaxVal. Sub-elements of MinVal or MaxVal are considered to appertain as well.

```
FUNCTION convertUnit (from: NUMERIC): NUMERIC;
```

converts the numeric value of the parameter "from" into the numeric return value and takes into consideration the units that are linked with the parameter and with the application of the result value (typically with the attribute the result is assigned to). This function can only be employed if the arguments of "from" are compatible with these of the return parameter, i.e. if their units have been derived from a common unit.

```
FUNCTION areAreas (Objects: OBJECTS OF ANYCLASS;
```

```
                    SurfaceBag: ATTRIBUTE OF @ Objects
                                          RESTRICTION (BAG OF ANYSTRUCTURE);
                    SurfaceAttr: ATTRIBUTE OF @ SurfaceBag
                                           RESTRICTION (SURFACE)): BOOLEAN;
```

checks whether the surfaces form a tessellation in accordance with object set (first parameter) and attribute (third parameter). If the area attribute is directly part of the object class, then indicate UNDEFINED for SurfaceBag, in all other cases indicate the structure attribute containing the surface attribute.

```
FUNCTION areAreas2 (Objects: OBJECTS OF ANYCLASS;
                    SurfaceBag: TEXT;
                    SurfaceAttr: TEXT): BOOLEAN;
```

checks whether the surfaces form a tessellation in accordance with object set (first parameter) and attribute (third parameter). If the area attribute is directly part of the object set, then indicate UNDEFINED for SurfaceBag, in all other cases indicate the path to the structure attribute whose structure contains the surface attribute. The attribute path is to be indicated as a TEXT constant, however in INTERLIS-syntax, e.g:

```
SET CONSTRAINT areAreas2 (ALL,"Geometry->Surfaces","Surface");
```

2.15 Views

Views are classes and structures whose objects are not original, but virtual, since they have been derived from objects of other views or classes, resp. structures. Amongst others, views are used to formulate the basics for graphics and special constraints. A further application consists of transmitting data in its derived, mostly simplified form to receiving systems.

Views are only transferred if they have been defined in a VIEW TOPIC. In this case their transmission takes place in much the same way as the complete transfer (keyword FULL) of normal classes, subsequently the data-receiver (cf. chapter 3 Sequential transfer) need not be concerned with how the (virtual) objects have been created. Views can also be explicitly excluded from any transfer (TRANSIENT), if they are only of local significance, i.e. if they only serve as base for other views. All incremental transfer of views is explicitly excluded, since no object identification can be assigned to view- objects.

Views can be abstract (ABSTRACT) or concrete. Concrete can also be founded on abstract bases. Therein it is only possible to approach base attributes that are concrete. If this is not the case, the view itself must be declared abstract.

Views can also be extended (EXTENDED or EXTENDS). However it is impossible to alter the formation definition. Thanks to the extension it is possible to conceive extensions of views, classes and structures that serve as foundation to the view in such a way, that further selections, attributes and constraints can be formulated.

Syntax rules:

```
ViewDef = 'VIEW' View-Name
             Properties<ABSTRACT,EXTENDED,FINAL,TRANSIENT>
             [ FormationDef | 'EXTENDS' ViewRef ]
                 { BaseExtensionDef }
                 { Selection }
             '='
             [ ViewAttributes ]
             { ConstraintDef }
           'END' View-Name ';'.
```

```
ViewRef = [ Model-Name '.' [ Topic-Name '.' ] ] View-Name.
```

By means of the formation definition (FormationDef) of a view we define how and on what basis the virtual objects of a view can be formed.

The view-projection (keyword PROJECTION OF) is the simplest form of a view. It allows the viewing of the super class (class, structure or view) in altered form (e.g. attributes only partly or in altered order).

With a *join* (keyword JOIN OF) we produce the Cartesian product (or cross-product) of the super classes (class or view), i.e. there are as many objects of the join-class as there are combinations of objects of the different super classes. It is also possible to define so-called "outer joins", in other words joins of objects of the first super class with (virtually inexistent) void objects of further super classes (indication "(OR NULL)"). Such void objects are added if no object of the desired other class could be found that might correspond to a certain combination of preceding objects. All attributes of the void object are undefined. Thus the corresponding view-attributes may not be compulsory.

The use of *union* (keyword UNION OF) allows merging different super classes into one single class. Typically all attributes of the different super classes are assigned to one attribute of the union class. The attribute type of the super class must be compatible with the attribute type of the union view (same type or one of its extensions).

By means of an *aggregation* (keyword AGGREGATION OF) it is possible to combine in one instance either all instances of a basic set or those whose required attribute combination is identical. Within the aggregation view the implicit attribute AGGREGATES (cf. chapter 2.13 Expressions) renders available the corresponding set of original objects in the form of BAG. This implicit attribute does not belong to the actual attributes of the view and thus will not be transferred even if a transfer is required. For example it can be assigned to the corresponding attribute of the aggregation view or in the form of an argument it can be consigned to a function.

By means of *inspection* (keyword INSPECTION OF) we obtain the set of all structure elements (with BAG OF or LIST OF or defined according to polyline, surface or tessellation) which belongs to a sub-structure attribute (direct or indirect) of an object class.

Any normal inspection of a tessellation respectively the inspection of a surface attribute will supply the boundaries of all areas resp. the surfaces of this class (structure SurfaceBoundary). The line strings of each boundary (structure SurfaceEdge) are formed in such a way as to render minimal their number. Consecutive line strings are thus combined into one line string. Should you attempt a further inspection of the attribute Lines, you would obtain all line strings (structure SurfaceEdge). However in this manner they will appear twice in a tessellation (once for very area object concerned).
By means of the inspection of a tessellation (keyword AREA INSPECTION OF) you will obtain exactly once the line strings of the edges of all areas belonging to the tessellation (in the form of structure SurfaceEdge). The two areas that are bordered by one common line string may be referred to THISAREA resp. THATAREA (cf. chapter 2.13 Expressions). As with a normal inspection line strings (structure SurfaceEdge) will be delivered as condensed as possible.

```
STRUCTURE SurfaceEdge =
  Geometry: DIRECTED POLYLINE;
END SurfaceEdge;

STRUCTURE SurfaceBoundary =
  Lines: LIST OF SurfaceEdge;
END SurfaceBoundary;
```

In the structure as stated below the inspection of a line attribute (POLYLINE) will supply all the segments (LineSegments) that form the line string objects of this class:

```
STRUCTURE LineGeometry =
  Segments: LIST OF LineSegment;
MANDATORY CONSTRAINT isOfClass (Segments[FIRST],INTERLIS.StartSegment);
END LineGeometry;
```

In other words the first curve segment represents a so-called StartSegment with length 0, and all others, so-called LineSegments are either straight lines, arcs or curve segments of some other type, in accordance with the definition stated in the structure.

INTERLIS only aspires to the conceptual description of a view. It is explicitly omitted to support an efficient realization of views. Thus the generating of views is part of a special degree of conformance.

Syntax rules:

```
FormationDef = ( Projection
               | Join
               | Union
               | Aggregation
               | Inspection ) ';'.

Projection = 'PROJECTION' 'OF' RenamedViewableRef.

Join = 'JOIN' 'OF' RenamedViewableRef
         (* ',' RenamedViewableRef
              [ '(' 'OR' 'NULL' ')' ] *).

Union = 'UNION' 'OF' RenamedViewableRef
          (* ',' RenamedViewableRef *).

Aggregation = 'AGGREGATION' 'OF' RenamedViewableRef
                ( 'ALL' | 'EQUAL' '(' UniqueEl ')' ).

Inspection = [ 'AREA' ] 'INSPECTION' 'OF' RenamedViewableRef
                 '->' StructureOrLineAttribute-Name
                      { '->' StructureOrLineAttribute-Name }.
```

All basic views employed in a view receive a name within the view in use; they can be referred to under this name. This name also corresponds to the basic view as long as it is not renamed by means of an explicit (local) base-name definition. Above all renaming becomes necessary where joins are defined which refer repeatedly to the same super class.

Syntax rules:

```
RenamedViewableRef = [ Base-Name '~' ] ViewableRef.

ViewableRef = [ Model-Name '.' [ Topic-Name '.' ] ]
              ( Structure-Name
              | Class-Name
              | Association-Name
              | View-Name ).
```

If within a view, resp. the extension of a view extensions of super classes ought to be taken into account, thus allowing the formulation of further attributes, selections or constraints, a corresponding extension definition (BaseExtensionDef) has to be listed. It proceeds from an already defined basic view and in turn describes its extensions (which must be extensions of former basic views) as basic views. If such a view extension is employed within expressions the value "UNDEFINED" occurs, if the basic object belonging to the virtual object does not match this view extension.

Syntax rule:

```
BaseExtensionDef = 'BASE' Base-Name 'EXTENDED' 'BY'
                      RenamedViewableRef { ',' RenamedViewableRef }.
```

By means of constraints (keyword WHERE) it is possible to apply further restrictions to the set of view objects defined by the formation definition.

Syntax rule:

```
Selection = 'WHERE' Logical-Expression ';'.
```

As far as attributes (and thus receiver views) and constraints are concerned, on principle views are built in the same way as classes and structures. For the purpose of facilitating the writing-process we further offer the possibility to transfer all attributes and roles of a view-base in the same order (ALL OF). However this would not make sense in unions and with AREA-inspections and consequently is inadmissible.

Syntax rule:

```
ViewAttributes = [ 'ATTRIBUTE' ]
                      { 'ALL' 'OF' Base-Name ';'
                      | AttributeDef
                      | Attribute-Name
                          Properties <ABSTRACT,EXTENDED,FINAL,TRANSIENT>
                            ':=' Factor ';' }.
```

In those simple cases where an attribute is transferred from the basic view, it is sufficient to indicate the attribute name and the assignation to the basic attribute. Such definitions are always final, that means they cannot be extended any further.

In unions it is compulsory to indicate for each attribute from which attributes of the basic class it has been derived. However an attribute must not refer to all basic classes as long as the attribute type permits undefined values. It is regarded as undefined for all missing basic objects.

The following example demonstrates how a view can be described which permits the definition of a proper relationship by means of the construct DERIVED FROM (cf. chapter 2.7 Proper relationships).

```
DOMAIN
  CHSurface = ... ;

FUNCTION Intersect (Surface1: CHSurface;
                    Surface2: CHSurface): BOOLEAN;

CLASS A =
  a1: CHSurface;
END A;

CLASS B =
  b1: CHSurface;
END B;

VIEW ABIntersection
  JOIN OF A,B;
  WHERE Intersect (A.a1,B.b1);
  =
END ABIntersection;


ASSOCIATION IntersectedAB
  DERIVED FROM ABIntersection =
  ARole -- A := ABIntersection -> A;
  BRole -- B := ABIntersection -> B;
END IntersectedAB;
```

2.16 Graphic descriptions

A graphic description consists of graphic definitions that are always based upon a view or a class (keyword BASED ON). By means of a graphic definition we conceptionally attempt to assign a graphic symbol (point, line, area symbol, text label) through one or several drawing rules (rule DrawingRule) to each object of this view or class – unless such an object has been ruled out by a specific selection (keyword WHERE) – that refers to the view or the class. Thus one or several graphic objects are created, which in turn will produce the respective representation (see figure 5). To this purpose each drawing rule must select a graphic symbol (with meta object name) and determine arguments for the corresponding parameters.

In brackets (rule Properties) inheritance characteristics can be defined. Whenever a graphic description is abstract, it can produce no symbol objects. The extension of a graphic must be based upon the same class as the base graphic (BASED ON lacks) or upon one of its extensions.

The drawing rule is identified by a name which is unequivocal within the graphic description, in order to be traced in extensions and subsequently refined. (Note: in the sense of specialization it refines also additional parameter values). Where there are extensions to a drawing rule in existence (in extending graphic descriptions), these will not create new graphic objects, but will only influence the symbol parameters of the graphic object determined by the basic definition. It is admissible to define several extensions to one graphic definition. They all are evaluated (in the order of their definition). This is especially of use when planning several extension piles for various aspects (e.g. various drawing rules). Subsequently, the different symbol parameters are determined. This definition may be produced in several steps. It is the value defined last that applies to each respective parameter. Firstly the primary definition is evaluated, and only then possible extensions. Furthermore it is possible to link parameter-assignments to a constraint (rule CondSignParamAssignment), i.e. the assignment is only in force if the constraint is fulfilled. If the selection constraint is not complied with, eventual sub-extensions are no longer taken into consideration. Within assignment rules (rule CondSignParamAssignment) the namespace of the basic class or basic view is valid for all attribute names or role names, for the names of meta objects, of functions and of run time parameters it is the namespace of the graphic definition that is valid.

As soon as drawing rules are concrete, we must define what class the graphic symbols that are to be assigned belong to. In extensions of drawing rules this class of graphic symbols must be replaced by a class which is an extension of the former. Primarily the "responsible" class of graphic symbols is the class to which the assigned graphic symbol object (a meta object) belongs. Concrete values must be assigned to the parameters introduced in the "responsible" class. If the parameters indicated correspond to an extended class of graphic symbols, this becomes the "responsible" class, provided it is in accordance with the class of the graphic symbol of the drawing rule or is one of its extensions.

In the constraints mentioned above object-attributes (see AttributePath in rule SignParamAssignment) can also be compared with run time parameters (cf. chapter 2.11 Run time parameters). Run time parameters which are of significance for the graphics (e.g. scale of the required graphic) typically are defined in symbology models, since they describe - much in the same way as parameter of symbols – graphic competences that are expected of a system. For the parameter of a graphic symbol which requires a meta object, a meta object reference has to be indicated (cf. chapter 2.10 Dealing with meta objects).

The value of the ordinary parameter of a graphic symbol is indicated in terms of a constant or reference to an object-attribute (cf. factor in rule SignParamAssignment). Thereby we always refer to the attribute of an object from the basic class or basic view which has been specified by means of BASED ON.

Since the representation often depends on attributes which have been defined by means of enumerations, a special construct is available to this purpose: the enumeration domain. An enumeration domain is either a single knot of the enumeration-type tree or an interval between knots defined by two knots of the same level. Interval definitions are only admissible when dealing with ordered enumeration-types. If the attribute value lies within the indicated enumeration domain, the corresponding parameter value is set. Concrete symbols are a result of the symbology model. Therein all symbol classes plus the necessary run time parameters (keyword PARAMETER) for their application are defined. It is admissible to define numeric data-types only in an abstract way.

Syntax rules:

```
GraphicDef = 'GRAPHIC' Graphic-Name Properties<ABSTRACT,FINAL>
               [ 'EXTENDS' GraphicRef ]
               [ 'BASED' 'ON' ViewableRef ] '='
               { Selection }
               { DrawingRule }
             'END' Graphic-Name ';'.

GraphicRef = [ Model-Name '.' [ Topic-Name '.' ] ] Graphic-Name.

DrawingRule = DrawingRule-Name Properties<ABSTRACT,EXTENDED,FINAL>
                [ 'OF' Sign-ClassRef ]
                    ':' CondSignParamAssignment
                        { ',' CondSignParamAssignment } ';'.

CondSignParamAssignment = [ 'WHERE' Logical-Expression ]
                            '(' SignParamAssignment
                                { ';' SignParamAssignment } ')'.

SignParamAssignment = SignParameter-Name
                        ':=' ( '{' MetaObjectRef '}'
                             | Factor
                             | 'ACCORDING' Enum-AttributePath
                                 '(' EnumAssignment
                                     { ',' EnumAssignment } ')' ).

EnumAssignment = ( '{' MetaObjectRef '}' | Constant )
                   'WHEN' 'IN' EnumRange.

EnumRange = EnumerationConst [ '..' EnumerationConst ].
```

For the application in symbology models the class SIGN has been predefined by INTERLIS:

```
CLASS SIGN (ABSTRACT) EXTENDS METAOBJECT =
  PARAMETER
    Sign: METAOBJECT;
END SIGN;
```

For concrete symbol classes this basic class has to be extended, thereby defining on the one hand concrete data, on the other parameters.

The following example outlines how the corresponding graphics (point symbols and text labels) are defined from a point class with coordinates, string and an enumeration as attribute.

The symbology model is to be defined as follows:

```
SYMBOLOGY MODEL SimpleSignsSymbology (en) AT "http://www.interlis.ch/"
  VERSION "2005-06-16" =

  DOMAIN
    S_COORD2 (ABSTRACT) = COORD NUMERIC, NUMERIC;
```

```
  TOPIC SignsTopic =

    CLASS Symbol EXTENDS INTERLIS.SIGN =
      PARAMETER
        Pos: MANDATORY S_COORD2;
    END Symbol;

    CLASS Textlabel EXTENDS INTERLIS.SIGN =
      PARAMETER
        Pos: MANDATORY S_COORD2;
        Text: MANDATORY TEXT;
    END Textlabel;

  END SignsTopic;

END SimpleSignsSymbology.
```

In addition to this symbology model concrete (symbol-)objects are supposed to have been listed and filed under the symbol library name (i.e. basket name) SimpleSignsBasket. The symbol objects listed (class symbol) are named dot-symbol, square-symbol, circle-symbol; the font-types (class text label) labeling1 and labeling2.

```
MODEL DataModel (en) AT "http://www.interlis.ch/"
  VERSION "2005-06-16" =

  DOMAIN
    LCoord = COORD
      0.000 .. 200.000 [m],
      0.000 .. 200.000 [m],
      ROTATION 2 -> 1;


  TOPIC DotTopic =

    DOMAIN
      DotType = (Stone
                  (large,
                   small),
                 Bolt,
                 Pipe,
                 Cross,
                 nonmaterialized) ORDERED;

    CLASS Dot =
      Position: LCoord;      !! LCoord be a coordinate value domain
      Type: DotType;
      DotName: TEXT*12;
    END Dot;

  END DotTopic;

END DataModel.


MODEL SimpleGraphic (en) AT "http://www.interlis.ch/"
  VERSION "2005-06-16" =

  IMPORTS DataModel;
  IMPORTS SimpleSignsSymbology;

  SIGN BASKET SimpleSignsBasket ~ SimpleSignsSymbology.SignsTopic;


  TOPIC DotGraphicsTopic =
    DEPENDS ON DataModel.DotTopic;
```

```
    GRAPHIC SimpleDotGraphic BASED ON DataModel.DotTopic.Dot =

      Symbol OF SimpleSignsSymbology.SignsTopic.Symbol: (
        Sign := {DotSymbol};
        Pos := Position
      );

    END SimpleDotGraphic;

  END DotGraphicsTopic;

END SimpleGraphic.
```

By means of this graphic (based upon the symbology model SimpleSignsSymbology and the graphic representation SimpleGraphic) for all dots of class dot simple dot symbols are drawn.

It is also conceivable that an improved graphic is desirable. Such an improvement can be made in different respects, e.g.:

– Additional symbols are desired (point-symbols, cross-symbols, triangle-symbols). This requires a supplementary symbol library by name of SimpleSignsPlusBasket. Since it is an extension of the library SimpleSignsBasket, the symbol objects (resp. meta objects) will be searched for in both libraries. If the library SimpleSignBasket were directly extended (EXTENDED), then for all graphics created with GraphicPlus within the model – including those that have been inherited from the model SimpleGraphic – the symbols would be first searched for in the extended library and only then in the basic library SimpleSignsBasket.

– Symbols should be scalable, thus permitting the creation of small and big squares with the same point symbol. This requires an extended symbology model that contains a parameter defining the scaling of symbols. Since symbol classes do not feature any additional attributes, it is not compulsory that corresponding libraries are in existence.

– Depending on the type of point, various point-symbols should be drawn: stones as big or small squares, bolts as circles and crosses and pipes with the cross symbol. The actual point symbol can be directly derived from the point-type. The scaling factor for small squares for the representation of small stones is obtained by means of an additional assignment. Non-materialized points remain simple dots; hence in this case no new assignment ensues.

```
SYMBOLOGY MODEL ScalableSignsSymbology (en) AT "http://www.interlis.ch/"
  VERSION "2005-06-16" =

  IMPORTS SimpleSignsSymbology;


  TOPIC ScalableSignsTopic EXTENDS SimpleSignsSymbology.SignsTopic =

    CLASS Symbol (EXTENDED) =
      PARAMETER
        ScaleFactor: 0.1 .. 10.0;  !! Default 1.0
    END Symbol;

  END ScalableSignsTopic;

END ScalableSignsSymbology.


MODEL GraphicPlus (en) AT "http://www.interlis.ch/"
  VERSION "2005-06-16" =

  IMPORTS SimpleGraphic;
  IMPORTS SimpleSignsSymbology;
  IMPORTS ScalableSignsSymbology;
```

```
  SIGN BASKET SimpleSignsPlusBasket EXTENDS
    SimpleGrafik.SimpleSignsBasket ~ ScalableSignsSymbology.ScalableSignsTopic;


  TOPIC DotGraphicsPlusTop EXTENDS SimpleGraphic.DotGraphicsTopic =

    GRAPHIC DotGraphicPlus EXTENDS SimpleDotGraphic =

      Symbol (EXTENDED) OF ScalableSignsSymbology.ScalableSignsTopic.Symbol: (
        Sign := ACCORDING Art (
          {SquareSymbol} WHEN IN #Stone,
          {CircleSymbol} WHEN IN #Bolt,
          {CrossSymbol} WHEN IN #Pipe .. #Cross
        )
      ),
      WHERE Type == #Stone.small (
          ScaleFactor := 0.5
      );

      Text OF SimpleSignsSymbology.Signs.Textlabel: (
        Sign := {Labeling1};
        Pos := Position;
        Text := DotName
      );

    END DotGraphicPlus;

  END DotGraphicsPlusTop;

END GraphicPlus.
```

3 Sequential transfer

3.1 Introduction

In this chapter we describe the sequential INTERLIS-transfer service. It permits the system-neutral exchange of data stores between different systems. Our INTERLIS-transfer service supports the complete as well as the incremental (resp. differential) exchange of data stores (replication). This transfer service is applicable to every INTERLIS model. Thus it is possible to transfer data (data model) and symbol objects (symbology models) by means of the same mechanism.

At present the INTERLIS-transfer service is defined as an exchange of XML-files (www.w3.org/XML/). For more extensive utilization of these INTERLIS/XML-files it is, amongst others, also possible to create XML-schema-documents (www.w3.org/XML/Schema). Nevertheless it is conceivable that further INTERLIS-transfer services will be defined in the future (e.g. based upon web-services or COBRA). For this reason the description of our INTERLIS-transfer service has been subdivided into the paragraphs *General Rules* and *XML-Coding.* General rules apply to *every sequential* INTERLIS-transfer service, independently of the concrete coding or transmission. Rules stated under *XML-Coding* apply especially to XML-formatted transfer files.

3.2 General rules for the sequential transfer

3.2.1 Derivation from the data model

Each INTERLIS-transfer can be derived from the corresponding data model by applying rules (model-based data transfer).

3.2.2 Organization of a transfer: Preliminaries

An INTERLIS-transfer is a sequential object-current. The object-current is subdivided into preliminaries and data domain.

- Indication of the current INTERLIS-version number (cf. chapter 2.3 Principal rule).
- Reference to the corresponding data model(s).
- Indication of the sender (SENDER).

Within the preliminaries there is an option to give information concerning the author of the object identification structure.

The preliminaries may contain comments (optional).

The organization of the data domain will be described more accurately in the following paragraphs.

3.2.3 Transferable objects

Within the data-domain objects (i.e. object instances) of concrete classes, relationships, views and graphic definitions can be transferred. Within the transfer, objects of views are treated in the same way as objects of concrete classes. At present the incremental transfer of views is not yet possible. Objects of views are only transferred provided the pertinent views have been declared within a VIEW TOPIC, otherwise they will not be transmitted. Furthermore views will not be transferred if they have been marked TRANSIENT.

3.2.4 Order of objects within the data domain

A data domain consists of a series of baskets (topic instances). Baskets can only be transferred as a whole. With incremental transfers only altered, resp. deleted objects will be transferred. However even with incremental transfers conceptionally it is the entire basket that is transferred along with the prehistory. On principle it is possible that a transfer contains basket provided by different models. In turn each basket contains all its objects. Within the transfer any order of objects is permitted, above all the objects need not in any case be ordered according to relationship or grouped in classes within a basket (as opposed to INTERLIS 1). Void baskets need not be transferred.

3.2.5 Coding of objects

Within the object current each basket and each object receives an identification. The basket identification must be a general and stable object-identifier (OID). The identifier of baskets and objects must be unequivocal along the entire transfer. Furthermore with every object a basket-identification is supplied within which the object has originally been created (original basket). With incremental transfers, resp. with initial transfers the identification of both basket and object must be a general and stable object-identifier (cf. appendix F *Organization of object identifiers (OID)*).

All object attributes (including COORD, SURFACE, AREA, POLYLINE, STRUCTURE, BAG OF, LIST OF; etc.) are memorized directly with the object. Attributes of the type AREA are coded as attributes of the type SURFACE. Attributes of the type BAG are coded as attributes of the type LIST. STRUCTURE is coded as LIST {1}.

For the transmission of attribute values only the printable symbols of the US-ASCII character set (32 to 126) and the symbols according to the symbol table in appendix D are available.

3.2.6 Transfer-types

Along with each basket the following information must be supplied:

- Details concerning the type (KIND) of transfer: FULL, INITIAL or UPDATE.
- Details concerning the STARTSTATE resp. ENDSTATE of the transfer (only with the types INITIAL (ENDSTATE) or UPDATE (STARTSTATE and ENDSTATE)).
- Details concerning the consistency of its contents: COMPLETE, INCOMPLETE.

It is admissible to have baskets with different types of transfer (Full, INITIAL and/or UPDATE) within the same transfer. The various types of transfer have the following significance:

- FULL – complete transfer. Upon receipt of a FULL-basket the receiver must first initialize a new basket and then insert all objects into the basket by means of INSERT. FULL is inappropriate as base for transfers since the object identifications are only valid for this transfer. Transfer-files in accordance with INTERLIS 1 correspond to FULL. Within the transfer-type FULL only the operation INSERT may occur.
- INITIAL – Primary transfer. Corresponds to the transfer type FULL with the sole difference that both basket and objects contained must feature general and stable OID. Likewise within the transfer-type FULL only the operation INSERT may occur.

- UPDATE – Additional transfer. An UPDATE-basket contains objects with INSERT, UPDATE or DELETE operations. All objects and the basket feature general and stable OID. UPDATE baskets may only be processed by the target system, if the start-state of the basket has already been received with INITIAL or UPDATE.

In addition the following transfer-rules apply to the transfer-type UPDATE:

- The receiving system may proceed on the assumption that after complete processing of all data of one UPDATE-basket a consistent has been re-established, i.e. an UPDATE-basket transfers a basket from a consistent start-state to a consistent end-state.
- An UPDATE-basket itself is not consistent, since in most cases references can only be resolved together with former transfers.

Furthermore to each object a corresponding transfer operation must be indicated (cf. chapter 1.4.5 Baskets, replication and data transfer). The operations INSERT, UPDATE and DELETE have the following significance:

- The operation INSERT signifies "insert a new object" (insert object).
- The operation UPDATE signifies "update object attribute values" (update object). All attributes (not only these that have been altered) must be transferred.
- The operation DELETE signifies "delete objects" (delete object). All attributes (not only those of OID's) should be transferred.

In many cases it is not an entire set of data that has to be transferred but only part of it. Depending on your choice of section, some geometries (polylines and surfaces) may be incomplete. In order to allow this without having to create an additional model, it must be possible that objects (and consequently their corresponding basket) can be signalized INCOMPLETE.

3.2.7 Normative references

In the following XML-coding resp. derivation of the XML-schema, we refer to various external documentations as listed below:

{1} W3C: XML 1.0 Specification, issue 2008

{2} W3C: XML Schema 1.1 Specification, issue 2012

{3} IETF: RFC 3629, UTF-8 Specification

{4} IETF: RFC 2045, paragraph 6.8 (Base64 coding)

3.3 XML-coding

3.3.1 Introduction

As opposed to the rules in chapter 3.2 General rules for the sequential transfer, the rules under XML-coding apply only to transfer files formatted according to XML-1.0 standard {1}. For formalizing the derivation rules of transfer formats we use the EBNF-notation already introduced in chapter 2.1 Syntax applied. Hereby the following rules are already predefined:

```
XML-Any = any XML-elements (well-formed XML).
XML-base64Binary = any binary data coded in Base64 {4}.
XML-String = any text without tags (incl. carriage return (#xD),
             line feed (#xA) and tab character(#x9)).
XML-NormalizedString = any single-line text.
XML-NcName = ( Letter | '_' ) { Letter | Digit | '_' | '-' | '.' }.
```

```
XML-ID = [ XML-NcName ':' ] ( Letter | Digit | '_' )
         { Letter | Digit | '_' | '-' | '.' }.
```

3.3.2 Symbol coding

As a standard for XML-String, resp. XML-NormalizedString only ASCII symbols 32 to 126, resp. symbols listed in Appendix D are available. These symbols are coded in accordance with the coding rule UTF-8 {3} or as XML Character Reference.

An XML Entity Reference is only admissible in the case of a few special symbols. Its value is an ASCII-symbol sequence which can only be used in this exact form within a transfer. Note: XML-entities such as ü (for the symbol ü) are not admitted in an INTERLIS 2-transfer file, because an INTERLIS 2-transfer file does not refer to a document type definition. DTD-admitted XML Entities are pre-defined by the XML 1.0 specifications:

– '&' must be replaced by the sequence '&'
– '<' must be replaced by the sequence '<'
– '>' must be replaced by the sequence '>'
– ''' must be replaced by the sequence '''
– '"' must be replaced by the sequence '"'.

A complete summary of this symbol coding with all possible forms of coding per symbol is to be found in appendix D *Symbol table*. If several coding forms per symbol are available it is up to an INTERLIS 2 writing program to select a pertinent form. An INTERLIS 2 reading program must be able to recognize all coding forms. If within a transfer, any symbols appear that are not listed in the symbol set defined by the model (cf. Chapter 2.5 Models, Topics, Classes or according to Appendix D if the symbol set is not indicated with the model) the software itself abitrarily decides how to deal with this situation (tacitly ignore or replace by a surrogate and comment accordingly).

Note: Several coding forms per symbol and symbols outside the possibly limited characterl set are admitted in order to allow for maximum compatibility with XML-tools.

3.3.3 General structure of a transfer file

An INTERLIS-transfer file is structured in accordance with the following EBNF-main rule:

```
Transfer =
  <?xml version="1.0" encoding="UTF-8"?>
  <%ili%:transfer
    {NameSpaceDef}
    xmlns:%ili%="http://www.interlis.ch/xtf/2.4/INTERLIS"
    [ xmlns:%geom%="http://www.interlis.ch/geometry/1.0" ]
    [ xmlns:%xsi%="http://www.w3.org/2001/XMLSchema-instance" ] >
    HeaderSection
    DataSection
  </%ili%:transfer>.

NameSpaceDef =
  xmlns="%XMLNS%" | xmlns="http://www.interlis.ch/xtf/2.4/%ModelName%".
```

The rule HeaderSection generates the header section of the transfer file and the rule DataSection generates the data section.

At any time an INTERLIS-transfer file generated by the transfer rule is also a well formed XML 1.0-transfer file {1}. Thus in an INTERLIS-transfer file any number of comment lines in the form of

```
<!-- %Comment% -->
```

may occur at places assigned by XML 1.0. However, the contents of these comment lines `%Comment%` may not be interpreted by the transfer software.

On principle, an INTERLIS 2 writing program may arbitrarily decide which prefix (`%ili%`, `%geom%`, `%ns%`) is used for which XML-namespace, resp. which XML-namespace is used as default XML-namespace. By `%ns%` we mean the respective XML-namespace of the INTERLIS model in which the corresponding model element has been defined.

The XML-schema for the namespace "http://www.interlis.ch/xtf/2.4/INTERLIS" is defined in Appendix B and in Appendix C for "http://www.interlis.ch/geometry/1.0". All other XML-schemas are derived from the rules stated in chapter 3.4 Derivation of an XML-schema from a data model.

Data are transferred as XML-objects. The tag names of XML-objects are derived from their respective object names in the INTERLIS-data model. In the case of translated data models (TRANSLATION OF) all tag names remain in the original language, i.e. the transfer format remains the same.

If the model contains an explicit namespace definition (XMLNS; cf. ruleModelDef), it will be adopted. In all other cases the namespace will be composed of "http://www.interlis.ch/xtf/2.4/" and the model name.

Note: In order to enhance lisibility and workability with various different text editors, we recommend formatting of XML-data before writing them in a transfer file.

3.3.4 Header section

A header section is structured as follows:

```
HeaderSection =
  <%ili%:headersection>
    <%ili%:models> (* Model *) </%ili%:models>
    [ <%ili%:sender>%Sender%</%ili%:sender> ]
    [ <%ili%:comment>%Comment%</%ili%:comment> ]
    </ili:headersection>.

Model =
  <%ili%:model>
    %ModelName%
  </%ili%:model>.
```

Under `models` all principal models must be listed if referring objects may appear in the transfer.

In `%Comment%` a comment can be added which further describes the transfer (optional).

3.3.5 Data section

The data section is structured as follows:

```
DataSection =
  <%ili%:datasection>
    { Basket }
  </%ili%:datasection>.
```

3.3.6 Coding of topics

Baskets are instances of a concrete TOPIC, resp. VIEW TOPIC. Baskets are coded as follows:

```
Basket =
  <%ns%:%TopicName% %ili%:bid="%BasketId%"
    [ %ili%:kind="%TransferKind%"
    %ili%:startstate="%StartState%" %ili%:endstate="%EndState%" ] >
    [ %ili%:domains="%Domain-Assignments%" ]
    [ %ili%:consistency="%Consistency%" ]
    { Object | Link | DeleteObject }
  </%ns%:%TopicName%>.
```

The value `%ns%:%TopicName%` has to be substituted correspondingly for each concrete topic (e.g. basic data ControlPoints). XML-attributes of the basket have the following significance:

- `%BasketId%`. In `%BasketId%` the basket identification must be listed. The basket identification is formatted in the same way as an XML-ID and additionally, in the case of an incremental update it must be an OID.
- `%TransferKind%`. Transfer-type (possible values: FULL, UPDATE, INITIAL. Where the attribute is omitted, FULL is presumed.
- `%StartState%`. Initial state of the basket before the transfer (only in connection with incremental update).
- `%EndState%`. Final state of the basket after the transfer (only in connection with incremental update).
- If in TOPIC generic domain definitions (GENERIC) are employed either directly or indirectly, then a concrete domain must be assigned to the generic domain in `%Domain-Assignments%`. If several generic domains are employed, each assignment must be separated from the next by a blank. An assignment consists of the two fully qualified domain names (`%ModelName%[.%TopicName%].%GenerichDomainName%=%ModelName%[.%TopicName%].%ConcreteDomainName%`).
- If only a geographic section is transferred from a data stock, this must be noted in `%Consistency%` by stating the value `INCOMPLETE`. Without this indication the value `COMPLETE` (= complete data stock) is assumed for `%Consistency%`. In a cut off part of a data stock it is also possible to have MULTIPOLYLINE instead of POLYLINE, resp. MULTISURFACE instead of SURFACE.

3.3.7 Coding of classes and relationships

The object instances of a concrete classe, resp. a non-inbedded relationship are coded as follows:

```
Object =
  <[ %ns%: ] [ %TopicName%. ] %ClassName%
    %ili%:tid="%Tid%" [ %ili%:operation="%Operation%" ] >
    [ { Attribute | EmbeddedLink } ]
  </[ %ns%: ][ %TopicName%. ] ClassName%>.

Link =
  <[ %ns%: ] [ %TopicName%. ] %AssociationName%
    [ %ili%:tid="%Tid%" ] [ %ili%:operation="%Operation%" ] >
    [ { Role | Attribute | EmbeddedLink } ]
  </[ %ns%: ] [ %TopicName%. ] %AssociationName%>.
```

```
DeleteObject =
  <%ili%:delete [ %ili%:tid= "%Tid%" ] )
    { <[ %ns%: ] %RoleName% %ili%:ref="%Tid%"/> }
   </%ili%:delete>.
```

The value `%ClassName%` must be substituted correspondingly for each concrete class (e.g. LFP). In addition to the attributes defined within the model each class – and thus each object instance – is assigned implicitly a transfer identification (XML-attribute `tid`). The `%Tid%` must be formatted in the same way as an XML-ID.

`%TopicName%` must only be indicated for classes / relationships in the case of name conflicts with classes / relationships / structures contained in other topics of the same model.

Instances of relationships (links) will only have a transfer identification, if it has been required explicitly by introducing the property OID within the scope of the definition (cf. chapter 2.7.1 Description of relationships).

In connection with the transfer-type FULL all `%Tid%` must be unequivocal along the entire transfer. In case of the transfer type INITIAL or UPDATE all `%Tid%` must be globally unique OID. Furthermore, in connection with the transfer-types INITIAL and UPDATE each object is assigned an attribute for the update-operation (XML-attribute `operation`). The XML-attribute `operation` can adopt the values INSERT, UPDATE or DELETE. Whenever `operation` is not indicated, the value INSERT will be presumed.

By means of `DeleteObject` it is possible to demand deletion of a specific object within the basket via its OID when executing an incremental update. With relationships without OID the instance (link) is identified by the combination of the OIDs of all objects referred to. When using `DeleteObject` as compared to `operation="DELETE"` it is not necessary to provide further attributes of the object.

The following order applies for the sequence of roles, attributes, reference atttributes and embedded relationships within a class: In the first place all roles of the basic class are coded, follwed by all attributes/ reference attributes of the basic class, all embedded relationships of the basic class then all attributes/ reference attributes of the extension, all embedded relationships of the extension etc. (onion layer principle). Within the same extension level all attributes/ reference attributes and roles are coded according to their definition order within the model file. Within the same extension level embedded relationships are sorted in alphabetical order.

Parameters are never transferred with the sole exception described in chapter 3.3.10 Coding of graphic definitions.

3.3.8 Coding of views

For the coding of views cf. chapter 3.2.4 Transferable objects. It is only the XML-attribut `tid` that is transferred, but not `operation`. In terms of attributes of the view-object only those attributes are transferred which have been indicated within the view explicitly under ATTRIBUTE, resp. implicitly with ALL OF.

3.3.9 Coding of relationships

There are two different manners for the coding of relationships: embedded or link. An embedded relationship is coded in terms of a sub-element of a class involved in the association. The instance of a link is coded in the same way as the instance of a class.

Relationships are always embedded, unless

- they have more than two roles or
- maximum cardinality is greater than 1 for both (basic) roles or
- an OID is required for the relationship or
- in the case of certain topic-spanning relationships (see below).

If maximum cardinality is greater than 1 in one of the two (basic) roles, the embedding takes place with the target-class of this role. If this target-class has been defined within a different topic than the (basic) association, then no embedding can take place.

If with both (basic) roles maximum cardinality is smaller or equal 1, the embedding takes place with the target class of the second role. If this target class has been defined in a different topic than the (basic) association and the target class of the first role has been defined in the same topic as the (basic) association, embedding will take place with the target class of the first role (in other words: if the target classes of both roles have been defined in a different topic than the (basic) association, no embedding can take place).

3.3.9.1 Embedded relationships

Embedded relationships are transferred in the same way as structure attributes of the class where the relationship is embedded.

The sub-structure is constructed as follows:

```
EmbeddedLink =
  <[ %ns%: ] %RoleName% %ili%:ref="%Tid%"
    [ %ili%:order_pos=”%PosNumber%” ]>
    [ EmbeddedLinkStruct ]
  </[ %ns%: ] %RoleName%>.

EmbeddedLinkStruct =
  <[ %ns%: ] [ %TopicName%. ] %AssociationName%>
    (* Attribute *)
  </[ %ns%: ] [ TopicName%. ] %AssociationName%>.
```

For `%RoleName%` you have to indicate the name of the role which refers to the opposite object (the other role will not be coded). In EmbeddedLinkStruct possible attributes of the relationship are coded. The XML-attributes `ref` and `order_pos` have the same significance as with non-embedded relationships. `%TopicName%` must only be indicated in the case of possible name conflicts with classes/ associations or structures from other topics of the same model.

3.3.9.2 Non-embedded relationships

Non-embedded relationships are transferred in the same way as object instances of classes.

Note: For relationships without explicit names the (class) name is a result of combining the individual role names (i.e. e.g. %RoleName1RoleName2%).

```
Role =
  <[ %ns%: ] %RoleName% %ili%:ref="%Tid%"
    [ %ili%:order_pos="%PosNumber%" ]>
  </[ %ns% ] %RoleName%>.
```

It is the transfer identification of the object that has been referred to that is stated in `ref`.

In ordered relationships the attribute `order_pos` (value > 0!) defines the absolute position of this reference in the ordered list of references that are part of this transfer basket.

3.3.10 Coding of graphic definitions

In the transfer the symbol classes referred to by the graphic definition (Sign-ClassRef) are transmitted for each graphic definition. The object instances of the symbol classes are created by executing graphic definitions on a concrete input data-set. Parameters are coded in the same way as attributes.

3.3.11 Coding of attributes

3.3.11.1 General rules for the coding of attributes

Each attribute of an object instance (including complex attributes such as (MULTI)COORD, (MULTI)POLYLINE, (MULTI)SURFACE, (MULTI)AREA, STRUCTURE, LIST OF; BAG OF, etc.) is coded as follows:

```
Attribute = {
    <[ %ns%:  ]%AttributeName%>
      AttributeValue
    </[%ns%:]%AttributeName%>
  }
  | ReferenceAttribute.

Attribute Value = (
    TextValue | MTextValue | EnumValue | NumericValue |
    FormattedValue | ClassTypeValue |
    DateValue | TimeValue | DateTimeValue |
    AttributePathTypeValue | OIDAttributeValue |
    BlackboxValue | StructureValue |
    CoordValue | MultiCoordValue | PolylineValue | MultiPolylineValue |
    SurfaceValue | MultiSurfaceValue
  ).
```

With undefined attribute values the attribute is not transferred. The measuring unit of the attribute value is not coded. Example of a simple attribute:

```
<Number>12345</Number>
```

With LIST and BAG multiple indication of the attribute tag is possible. Example for a simple list of numbers:

```
<Number>12345</Number>
<Number>23456</Number>
<Number>34567</Number>
```

3.3.11.2 Coding of strings

Attributes of the base type TEXT resp. MTEXT (and consequently also NAME and URI) are coded as follows:

```
MTextValue = XML-String.
TextValue = XML-NormalizedString.
```

3.3.11.3 Coding of enumerations

Enumerations are coded as follows:

```
EnumValue = ( EnumElement-Name { '.' EnumElement-Name } ) | 'OTHERS'.
```

For the coding of enumerations (without taking into consideration whether the domain comprises only its leaves or the nodes as well) the syntax of enumeration constants is applied (rule EnumValue). The sign # is omitted. The pre-defined text orientation-types HALIGNMENT and VALIGNMENT are coded in the same way as enumerations. Equally the type BOOLEAN is transmitted like an enumeration.

3.3.11.4 Coding of numeric data types

Numeric values are coded as follows:

```
NumericValue = NumericConst.
```

Note: Float numbers can be transferred in different representations (with or without mantissa). They can also be transferred with a higher degree of precision than required by the domain. However it is essential that the value of the float number do not violate the required domain. Thus it is possible that for instance the number 100 (for an assumed domain of 0..999) be transferred as 100, 100.0000001, 10.0e1 or 1.0e2.

We advise the receiving party to round all values except intermediat points of arcs (cf. chapter 3.3.11.14 Coding of line strings) according to the numeric domain.

3.3.11.5 Coding of formatted domains

Formatted domains are coded according to the format definition:

```
FormattedValue = XML-NormalizedString.
```

3.3.11.6 Coding of date

The type DATE is coded as follows:

```
DateValue = JJJJ-MM-TT.
```

JJJJ stands for the year, MM for the number of the month (01 .. 12), TT for the day (01 .. 31). So December 1st 1997 will be transferred as `1997-12-01`.

3.3.11.7 Coding of time

The type TIME is coded as follows:

```
TimeValue = HH:MM:SS.
```

For hour (HH), minute (MM) and second (SS) two-digit numbers must be indicated (e.g. `01:13:00`).

3.3.11.8 Coding of date and with time

The type DATETIME is coded as follows:

```
DateTimeValue = DateValueTTimeValue.
```

Between date and time the letter T appears (e.g. `1997-12-01T01:13:00`).

3.3.11.9 Coding of blackboxes

Attribute values of the BLACKBOX type are coded as follows:

```
BlackboxValue = XML-Any | XML-base64Binary.
```

The XML-variety of the BLACKBOX type is coded as XML-Any, the binary variety as XML-base64Binary.

3.3.11.10 Coding of class types

Attribute values of the type CLASS or STRUCTURE are coded as follows:

```
ClassTypeValue = XML-NormalizedString.
```

The XML-NormalizedString contains the fully qualified class, structure or relationship name (e.g. DM01AVCH24D.ControlPointsCategory1.LFP1).

3.3.11.11 Coding of attribute path types

Attribute values of the ATTRIBUTE type are coded as follows:

```
AttributePathTypeValue = XML-NormalizedString.
```

The XML-NormalizedString contains the fully qualified class name, followed by the attribute name separated by a dot (e.g. BaseDataSet.ControlPoints.LFP.Number).

3.3.11.12 Coding of structure attributes

Structure elements of the type STRUCTURE are coded as follows:

```
StructureValue =
  <[ %ns%: ] [ %TopicName%. ] %StructureName%>
    (* Attribute *)
  </[ %ns%: ] [ %TopicName%. ] %StructureName%>.
```

%TopicName% must only be indicated for such structures where a name conflict with classes/ relationships/ structures of other topics of the same model occur.

3.3.11.13 Coding of coordinates

Attribute values of the type COORD are coded as follows:

```
CoordValue =
  <geom:coord>
    <geom:c1>NumericConst</geom:c1>
    <geom:c2>NumericConst</geom:c2>
    [ <geom:c3>NumericConst</geom:c3> ]
  </geom:coord>.
```

The individual XML-sub-objects must be filled as follows:

- c1. First component of the coordinate (coded as numeric value).
- c2. Second component of the coordinate (only with 2D- and 3D- coordinates, coded as numeric value).
- c3. Third component of the coordinate (only with 3D-coordinates, coded as numeric value).

Attribute values of the MULTICOORD type are coded as follows:

```
MultiCoordValue =
  <geom:multicoord>
    (* CoordValue *)
  </geom:multicoord>.
```

3.3.11.14 Coding of line strings

Attribute values of the type POLYLINE are coded as follows:

```
PolylineValue =
  <geom:polyline>
    SegmentSequence
  </geom:polyline>.

StartSegment = CoordValue.

StraightSegment = CoordValue.

ArcSegment =
  <geom:arc>
    <geom:c1>NumericConst</geom:c1>
    <geom:c2>NumericConst</geom:c2>
    [ <geom:c3>NumericConst</geom:c3> ]
    <geom:a1>NumericConst</geom:a1>
    <geom:a2>NumericConst</geom:a2>
    [ <geom:r>NumericConst</geom:r> ]
  </geom:arc>.

LineFormSegment = StructureValue.

SegmentSequence = StartSegment (* StraightSegment
                               | ArcSegment
                               | LineFormSegment *).
```

Straight segments of a line string are coded in accordance with the rule StraightSegment, for arc segments the rule ArcSegment applies. Line segments defined with LINE FORM are coded as structures (LineStructure).

Note: For arc segments (rule ArcSegment) the radius (optional XML-attribute r) can be transmitted redundantly to the intermediate point coordinate (a1/a2). The intermediate point of an arc is only of significance for its position. Its height must be interpolated linearly between start and end point. If the arc has been defined clockwise (from start to end point), the radius will have a positive sign, otherwise negative. If differences occur between radius and coordinate values, it is the radius that prevails (cf. chapter 2.8.12.2 Line strings with straight line segments and circle arcs as predefined curve segments). Coordinate values of intermediate points should not be rounded according to domain definition, in order to internally determine arc attributes with a maximum degree of precision in the case of missing radius. The vertex height (c3) only has to be transferred with 3D-line strings.

Attribute values of the MULTIPOLYLINE type (and also for POLYLINE in the case of a data stock-section) are coded as follows:

```
MultiPolylineValue =
  <geom:multipolyline>
   (* PolylineValue *)
  </geom:multipolyline>.
```

3.3.11.15 Coding of surfaces and tessellations

SURFACE and AREA are coded as follows:

```
SurfaceValue =
  <geom:surface>
    Boundaries
  </geom:surface>.
```

```
Boundaries = OuterBoundary { InnerBoundary }.

OuterBoundary =
  <geom:exterior>
    PolylineValue
  </geom:exterior>.

InnerBoundary =
  <geom:interior>
    PolylineValue
  </geom:interior>.
```

Surfaces are transmitted as a sequence of boundaries. A boundary is a sequence of boundary lines, the next boundary line starting with the end point of the preceding boundary line. The end point of the last boundary line is identical with the start point of the first boundary line. Thus the boundary lines form a closed line string (polygon). A boundary may be parted into boundary lines at any vertex of your choosing. The segments may differ with every transfer – above all with incremental updates.

The first boundary of a surface (OuterBoundary) is the outer boundary of a surface, possibly followed by inner boundaries (InnerBoundary) of the surface which limit the enclosures of the surface. The inner boundaries must be located completely within the outer boundaries. The individual boundaries of a surface may not overlap.

With tessellation (AREA) all boundary lines of the surface must coincide with the boundary lines of the neighboring surface(s), unless they form part of the perimeter of the area network. Two boundary lines are considered identical if in every segment of the boundary line all vertices are identical with the corresponding segment of the neighboring surface. With arc vertices merely the sign of the arc radius may differ.

Attribute values of the MULTISURFACE-type (as well as SURFACE in the case of data stock-section) are coded as follows:

```
MultiSurfaceValue =
   <geom:multisurface>
     (* SurfaceValue *)
  </geom:multisurface>.
```

3.3.11.16 Coding of references

Attributes of the type REFERENCE TO are coded as follows:

```
ReferenceAttribute =
  <[ %ns%: ] %AttributeName% %ili%:ref="%Tid%">
  </[ %ns%: ] %AttributeName%>.
```

The XML-attribute `ref` has the same significance as with proper relationships.

3.3.11.17 Coding of meta objects

Attributes of the METAOBJECT-type (cf. Appendix A *The internal INTERLIS-data model*) are coded in the same way as LIST OF or BAG OF. However parameters of the type METAOBJECT (rule ParameterDef) are not transmitted. Parameters of the type METAOBJECT OF are transmitted as attributes of the type NAME.

3.3.11.18 Coding of the OIDType

Attribute values of the type OIDType are coded in the same way as an XML-ID incl. OID-domain. If OIDType is a NumericType the rules governing the coding of numeric types also have to be applied for the value (without the OID-domain).

```
OIDAttributeValue = XML-ID.
```

3.4 Derivation of an XML-schema from a data model

3.4.1 Introduction

Each XML-schema file (XSD) must comply with the rules according to {2}. Furthermore, each XSD-file derived from a INTERLIS 2.4 data model is of the following basic structure:

```
XSDDef =
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.interlis.ch/xtf/2.4/%ModelName%"
    targetNamespace="http://www.interlis.ch/xtf/2.4/%ModelName%"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    xmlns:ili="http://www.interlis.ch/xtf/2.4/INTERLIS"
    xmlns:geom="http://www.interlis.ch/geometry/1.0"
    >

    { TypeDef | ClassTypeDef | ObjectDef }
    { BasketDef }

  </xsd:schema>
```

The XSD-file consists of a header with information regarding the model (ModelName), followed by declarations concerning the type (TypeDef) and classes (ClassTypeDef) and finally the basket definitions (BasketDef). Each XML-schema for an INTERLIS model imports the external XML-schemas "http://www.interlis.ch/xtf/2.4/INTERLIS" (INTERLIS 2.4 basic definitions, see also Appendix B) and "http://www.interlis.ch/geometry/1.0" (INTERLIS geometry schema, see also Appendix C).

The XML-schema thus obtained may contain any number of comments (<!-- … -->) or additional xsd:annotation-elements. This XML-schema may realize the INTERLIS model in a more precise way, i.e with additional "facets".

3.4.2 Type declarations

All types of a model (global or at topic level) are coded as follows:

```
TypeDef = SimpleTypeDef | ComplexTypeDef

SimpleTypeDef =
  <xsd:simpleType name=" [ %TopicName%. ] %TypeName%Type">
    SimpleTypeRestriction
  </xsd:simpleType>

ComplexTypeDef =
  <xsd:complexType name=" [ %TopicName%. ] %TypeName%Type">
    ComplexTypeContent
  </xsd:complexType>
```

For all types "Type" must be added at the end of the name (e.g. LCoordType for the INTERLIS type LCoord). For type declarations at topic level the type name must be additionally preceded by %TopicName% (e.g. LandCover.LCSType), provided the type name was already used at model level.

3.4.3 Simple types (SimpleTypeRestriction)

3.4.3.1 Numeric types

Numeric types without gradation (lengths, surfaces, volumes, angles, domains) are translated as follows:

```
<xsd:restriction base="xsd:decimal">
  [ <xsd:minInclusive value="%min-Dez%">
  <xsd:maxInclusive value="%max-Dez%"> ]
</xsd:restriction>
```

A restricted domain occurs only if the type is declared FINAL in the model.

Numeric types with decimal points and gradation (lengths, surfaces, volumes, angles, ranges) are translated as follows:

```
<xsd:restriction base="xsd:double">
  [ <xsd:minInclusive value="%min-Dez%">
  <xsd:maxInclusive value="%max-Dez%"> ]
</xsd:restriction>
```

A restricted domain occurs only if the type is declared FINAL in the model.

Numeric types without decimal points and gradation (lengths, surfaces, volumes, angles, ranges) are translated as follows:

```
<xsd:restriction base="xsd:integer">
  <xsd:minInclusive value="%min-Dez%">
  <xsd:maxInclusive value="%max-Dez%">
</xsd:restriction>
```

3.4.3.2 Text

The type TEXT*n is translated as follows:

```
<xsd:restriction base="xsd:normalizedString">
  <xsd:maxLength value="%N%">
</xsd:restriction>
```

3.4.3.3 MText

The type MTEXT*n is translated as follows:

```
<xsd:restriction base="xsd:string">
  <xsd:maxLength value="%N%">
</xsd:restriction>
```

3.4.3.4 OIDType, formatted domains, class types, attribute paths

OIDType, formatted domains, class types and attribute paths are translated as follows:

```
<xsd:restriction base="xsd:NCName"/>
```

3.4.3.5 XML-blackbox

The type BLACKBOX XML is translated as follows:

```
<xsd:restriction base="xsd:anyType"/>
```

3.4.3.6 Binary blackbox

The type BLACKBOX BINARY is translated as follows:

```
<xsd:restriction base="xsd:base64Binary"/>
```

3.4.3.7 Date

The type DATE is translated as follows:

```
<xsd:restriction base="xsd:date"/>
```

3.4.3.8 Time

The type TIMEOFDAY is translated as follows:

```
<xsd:restriction base="xsd:time"/>
```

3.4.3.9 Date including time

The type DATETIME is translated as follows:

```
<xsd:restriction base="xsd:dateTime"/>
```

3.4.3.10 Enumerations

Enumerations (with corresponding text orientation) are translated as follows:

```
<xsd:restriction base="xsd:normalizedString">
  { EnumerationValue }
</xsd:restriction>

EnumerationValue =
  <xsd:enumeration value="%EnumerationValue%"/>
```

The `%EnumerationValue%` is only listed if the type has been declared FINAL in the model. In this case all valid values of the enumeration type must be listed for `%EnumerationValue%` . No listing occurs if the type is not FINAL in the model.

3.4.4 Complex types (ComplexTypeContent)

3.4.4.1 Coordinates

COORD2 resp. COORD3 are translated as follows:

```
<xsd:sequence>
<xsd:element ref="geom:coord" minOccurs="1" maxOccurs="1">
</xsd:sequence>
```

3.4.4.2 Multi-coordinates

MULTICOORD2 resp. MULTICOORD3 are translated as follows:

```
<xsd:sequcence>
<xsd:element ref="geom:multicoord" minOccurs="1" maxOccurs="1">
</xsd:sequcence>
```

3.4.4.3 Polylines

The type POLYLINE is translated as follows:

```
<xsd:sequcence>
<xsd:element ref="geom:polyline" minOccurs="1" maxOccurs="1">
</xsd:sequcence>
```

3.4.4.4 Multi-polylines

The type MULTIPOLYLINE is translated as follows:

```
<xsd:sequcence>
<xsd:element ref="geom:multipolyline" minOccurs="1" maxOccurs="1">
</xsd:sequcence>
```

3.4.4.5 Surfaces

The type SURFACE is translated as follows:

```
<xsd:sequcence>
<xsd:element ref="geom:surface" minOccurs="1" maxOccurs="1">
</xsd:sequcence>
```

3.4.4.6 Multi-surfaces

The type MULTISURFACE is translated as follows:

```
<xsd:sequcence>
<xsd:element ref="geom:multisurface" minOccurs="1" maxOccurs="1">
</xsd:sequcence>
```

3.4.4.7 Roles

Roles are translated as follows:

```
<xsd:attribute ref="ili:ref" use="required"/>
<xsd:attribute ref="ili:order_pos"/>
```

3.4.4.8 Reference attributes

Reference attributes are translated as follows:

```
<xsd:attribute ref="ili:ref" use="required"/>
```

3.4.5 Classes, structures and relationships

Classes, strucures and independent relationships are translated as follows:

```
ClassTypeDef = RootClassTypeDef | ExtendedClassTypeDef

RootClassTypeDef =
  <xsd:complexType name=" [ %TopicName%. ] %ClassName%Type">
    <xsd:sequence>
```

```
        <xsd:element ref="ili:extensions" minOccurs="0"/>
        (* AttributeDef | EmbeddedLinkDef | RoleDef *)
      </xsd:sequence>
      <xsd:anyAttribute processContents="lax"/>
      [ <xsd:attribute ref="ili:tid"/> ]
      [ <xsd:attribute ref="ili:operation"/> ]
    </xsd:complexType>

  ExtendedClassTypeDef =
    <xsd:complexType name=" [ %TopicName%. ] %ClassName%Type">
      <xsd:complexContent>
        <xsd:extension base=" [ %TopicName%. ] %ClassName%Type">
          <xsd:sequence>
            (* AttributeDef | EmbeddedLinkDef | RoleDef *)
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>

  ObjectDef =
    <xsd:element name= [ %TopicName%. ] "%ClassName%" type="%ClassName%Type"
  [ substitutionGroup=" [ %TopicName%. ] "%ClassName%" ] />
```

For classes an indication of the XML-attribute TID is mandatory. In structures the attributes TID and OPERATION may never occur.

```
  AttributeDef =
    AttributeWithNamedTypeDef | AttributeWithLocalTypeDef

  AttributeWithNamedTypeDef =
    <xsd:element name="%Attribut-Name%"
        minOccurs="%MinCard%" maxOccurs="%MaxCard%"
        type="%NamedTypeRef%"/>

  AttributeWithLocalTypeDef =
    <xsd:element name="%Attribut-Name%"
        minOccurs="%MinCard%" maxOccurs="%MaxCard%">
      TypeDef
    </xsd:element>

  EmbeddedLinkDef =
    <xsd:element name="%Role-Name%">
      <xsd:complexType>
        [ <xsd:sequence>
            <xsd:element ref=" [ %TopicName%. ] %ClassName%" minOccurs="0"/>
          </xsd:sequence> ]
        <xsd:attribute ref="ili:ref" use="required"/>
        [ <xsd:attribute ref="ili:order_pos" /> ]
        <xsd:anyAttribute processContents="lax"/>
      </xsd:complexType>
    </xsd:element>

  RoleDef =
    <xsd:element name="%Role-Name%">
      <xsd:complexType>
        <xsd:attribute ref="ili:ref" use="required"/>
        [ <xsd:attribute ref="ili:order_pos" /> ]
        <xsd:anyAttribute processContents="lax"/>
      </xsd:complexType>
    </xsd:element>
```

For attributes, roles and EmbeddedLinks within one class the following order applies: Attributes, reference attributes and roles are coded according to their definition order in the model file. EmbeddedLinks within the same extension level are sorted alphabetically. Roles, attributes and EmbeddedLinks inherited from a basic class will not be repeated.

For List and Bag attributes `%MinCard%` for minimum, resp. `%MaxCard%` for maximum cardinality must be indicated.

For all other attributes `%MaxCard%` equals 1. For MANDATORY attributes `%MinCard%` also equals 1, otherwise 0.

Embedded relationships are defined according to the rule `EmbeddedLinkDef`.

TypeDef for attributes with local type declaration are listed without any XSD-attribute `name`.

3.4.6 BasketDef

Per topic the following definition is listed:

```
BasketDef =
  <xsd:element name="%Thema-Name%">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="ili:extensions"/>
        { <xsd:element ref=" [ %TopicName%. ] %ClassName%"
            minoccurs="0" maxOccurs="unbounded"/> }
      </xsd:choice>
      <xsd:attribute ref="ili:bid" use="required"/>
      <xsd:attribute ref="ili:consistency"/>
      [ <xsd:attribute ref="ili:domains"/> ]
      [ <xsd:attribute ref="ili:kind"/>
        <xsd:attribute ref="ili:startstate"/>
        <xsd:attribute ref="ili:endstate"/> ]
      <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
  </xsd:element>
```

For element definitions of all classes within the BasketDef the following applies: In the first place all classes of basic topics are coded, followed by all classes of extension topics etc. (onion layer principle). However, classes of the extension topic are only coded if the basic class has not been coded as part of the basic topic.

In the XSD-file topics must be defined in the same order as in the data model.

4 Disclaimer/Reference to third party rights

eCH-standards which the registered association **eCH** provides the user free of charge or which make reference to **eCH**, shall only have the status of recommendations. The registered association **eCH** will not be liable in any event for any decisions made or measures taken by the user based on these documents. The user will be responsible for verifying the documents himself prior to their use and to seek advice if required. **eCH**-standards can and shall not replace the technical, organizational or legal advice in the individual case.

Documents, procedures, methods, products and standards which are made reference to in **eCH**-standards are possibly protected by trademarks, copyrights or patents. It is the exclusive responsibility of the user to obtain the necessary licenses from the entitled persons and/or organizations.

Although the registered association **eCH** has taken adequate care to prepare the **eCH**-standards with due diligence, it cannot grant any warranty or guarantee that the information and documents provided are up-to-date, complete, true or without any errors. **eCH** reserves the right to change the contents of the **eCH**-standards at any time and without prior announcement.

Any liability for damage caused by the use of the **eCH**-standards by the user shall be excluded to the extent legally admissible.

5 Copyrights

Persons preparing **eCH**-standards shall remain the owners of their intellectual property rights. These persons, however, obligate themselves to provide their intellectual property rights or other rights in third party intellectual property rights, to the extent possible, to the relevant technical units and the registered association **eCH** for free and for unlimited use and further development as part of the purpose of the association.

The standards prepared by the technical units can be used, distributed and developed further for free and to an unlimited extent by stating the name of the respective author of **eCH.**

eCH-standards are fully documented and free of any restrictions of licence and/or patent law. The associated documentation can be requested for free.

These provisions shall apply to the standards prepared by **eCH** only, however, not to any standards or products of third parties which include reference to **eCH**-standards. The standards include the relevant references to third party rights.

Annex A (normative) – The internal INTERLIS-data model

Hereafter the entire internal data model has been summarized once more. This model only serves to illustrate our explanations and cannot be compiled (because for instance it uses names which are keywords according to the language definition (cf. chapter 2.2.7 Special symbols and reserved words). Any software processing INTERLIS, such as the INTERLIS-compiler placed at your disposal by KOGIS, must be able to recognize all elements of this model.

```
INTERLIS 2.4;

TYPE MODEL INTERLIS (en) AT "http://www.interlis.ch/"
  VERSION "2014-07-09" =

  LINE FORM
    STRAIGHTS;
    ARCS;

  UNIT
    ANYUNIT (ABSTRACT);
    DIMENSIONLESS (ABSTRACT);
    LENGTH (ABSTRACT);
    MASS (ABSTRACT);
    TIME (ABSTRACT);
    ELECTRIC_CURRENT (ABSTRACT);
    TEMPERATURE (ABSTRACT);
    AMOUNT_OF_MATTER (ABSTRACT);
    ANGLE (ABSTRACT);
    SOLID_ANGLE (ABSTRACT);
    LUMINOUS_INTENSITY (ABSTRACT);
    MONEY (ABSTRACT);

    METER [m] EXTENDS LENGTH;
    KILOGRAM [kg] EXTENDS MASS;
    SECOND [s] EXTENDS TIME;
    AMPERE [A] EXTENDS ELECTRIC_CURRENT;
    DEGREE_KELVIN [K] EXTENDS TEMPERATURE;
    MOLE [mol] EXTENDS AMOUNT_OF_MATTER;
    RADIAN [rad] EXTENDS ANGLE;
    STERADIAN [sr] EXTENDS SOLID_ANGLE;
    CANDELA [cd] EXTENDS LUMINOUS_INTENSITY;

  DOMAIN
    URI (FINAL) = TEXT*1023;
    NAME (FINAL) = TEXT*255;
    INTERLIS_1_DATE (FINAL) = TEXT*8;
    BOOLEAN (FINAL) = (
      false,
      true) ORDERED;
    HALIGNMENT (FINAL) = (
      Left,
      Center,
      Right) ORDERED;
    VALIGNMENT (FINAL) = (
      Top,
      Cap,
      Half,
      Base,
      Bottom) ORDERED;
    NOOID = OID ANY;
    ANYOID (ABSTRACT) EXTENDS NOOID = OID ANY;
    I32OID EXTENDS ANYOID = OID 0 .. 2147483647;
    STANDARDOID EXTENDS ANYOID = OID TEXT*16;
    UUIDOID EXTENDS ANYOID = OID TEXT*36;
```

```
  LineCoord (ABSTRACT) = COORD NUMERIC, NUMERIC;

FUNCTION myClass (Object: ANYSTRUCTURE): STRUCTURE;
FUNCTION isSubClass (potSubClass: STRUCTURE; potSuperClass: STRUCTURE):
  BOOLEAN;
FUNCTION isOfClass (Object: ANYSTRUCTURE; Class: STRUCTURE): BOOLEAN;
FUNCTION elementCount (bag: BAG OF ANYSTRUCTURE): NUMERIC;
FUNCTION objectCount (Objects: OBJECTS OF ANYCLASS): NUMERIC;
FUNCTION len (TextVal: TEXT): NUMERIC;
FUNCTION lenM (TextVal: MTEXT): NUMERIC;
FUNCTION trim (TextVal: TEXT): TEXT;
FUNCTION trimM (TextVal: MTEXT): MTEXT;
FUNCTION isEnumSubVal (SubVal: ENUMTREEVAL; NodeVal: ENUMTREEVAL): BOOLEAN;
FUNCTION inEnumRange (Enum: ENUMVAL;
                      MinVal: ENUMTREEVAL;
                      MaxVal: ENUMTREEVAL): BOOLEAN;
FUNCTION convertUnit (from: NUMERIC): NUMERIC;
FUNCTION areAreas (Objects: OBJECTS OF ANYCLASS;
                   SurfaceBag: ATTRIBUTE OF @ Objects
                                      RESTRICTION (BAG OF ANYSTRUCTURE);
                   SurfaceAttr: ATTRIBUTE OF @ SurfaceBag
                                       RESTRICTION (SURFACE)): BOOLEAN;
FUNCTION areAreas2 (Object: OBJECT OF ANYCLASS;
                    SurfaceBag: TEXT;
                    SurfaceAttr: TEXT): BOOLEAN;
FUNCTION areAreas3 (Objects: OBJECTS OF ANYCLASS;
                    SurfaceBag: TEXT;
                    SurfaceAttr: TEXT): BOOLEAN;

CLASS METAOBJECT (ABSTRACT) =
  Name: MANDATORY NAME;
UNIQUE Name;
END METAOBJECT;

CLASS METAOBJECT_TRANSLATION =
  Name: MANDATORY NAME;
  NameInBaseLanguage: MANDATORY NAME;
UNIQUE Name;
UNIQUE NameInBaseLanguage;
END METAOBJECT_TRANSLATION;


STRUCTURE AXIS =
  PARAMETER
    Unit: NUMERIC [ANYUNIT];
END AXIS;

CLASS REFSYSTEM (ABSTRACT) EXTENDS METAOBJECT =
END REFSYSTEM;

CLASS COORDSYSTEM (ABSTRACT) EXTENDS REFSYSTEM =
  ATTRIBUTE
    Axis: LIST {1..3} OF AXIS;
END COORDSYSTEM;

CLASS SCALSYSTEM (ABSTRACT) EXTENDS REFSYSTEM =
  PARAMETER
    Unit: NUMERIC [ANYUNIT];
END SCALSYSTEM;

CLASS SIGN (ABSTRACT) EXTENDS METAOBJECT =
  PARAMETER
    Sign: METAOBJECT;
END SIGN;


TOPIC TIMESYSTEMS =
```

```
    CLASS CALENDAR EXTENDS INTERLIS.SCALSYSTEM =
      PARAMETER
        Unit(EXTENDED): NUMERIC [TIME];
    END CALENDAR;

    CLASS TIMEOFDAYSYS EXTENDS INTERLIS.SCALSYSTEM =
      PARAMETER
        Unit(EXTENDED): NUMERIC [TIME];
    END TIMEOFDAYSYS;

  END TIMESYSTEMS;

  UNIT
    Minute [min] = 60 [INTERLIS.s];
    Hour   [h]   = 60 [min];
    Day    [d]   = 24 [h];
    Month [M] EXTENDS INTERLIS.TIME;
    Year [Y] EXTENDS INTERLIS.TIME;

  REFSYSTEM BASKET BaseTimeSystems ~ TIMESYSTEMS
    OBJECTS OF CALENDAR: GregorianCalendar
    OBJECTS OF TIMEOFDAYSYS: UTC;

  STRUCTURE TimeOfDay (ABSTRACT) =
    Hours: 0 .. 23 CIRCULAR [h];
    CONTINUOUS SUBDIVISION Minutes: 0 .. 59 CIRCULAR [min];
    CONTINUOUS SUBDIVISION Seconds: 0.000 .. 59.999 CIRCULAR [INTERLIS.s];
  END TimeOfDay;

  STRUCTURE UTC EXTENDS TimeOfDay =
    Hours(EXTENDED): 0 .. 23 {UTC};
  END UTC;

  DOMAIN
    GregorianYear = 1582 .. 2999 [Y] {GregorianCalendar};

  STRUCTURE GregorianDate =
    Year: GregorianYear;
    SUBDIVISION Month: 1 .. 12 [M];
    SUBDIVISION Day: 1 .. 31 [d];
  END GregorianDate;

  STRUCTURE GregorianDateTime EXTENDS GregorianDate =
    SUBDIVISION Hours: 0 .. 23 CIRCULAR [h] {UTC};
    CONTINUOUS SUBDIVISION Minutes: 0 .. 59 CIRCULAR [min];
    CONTINUOUS SUBDIVISION Seconds: 0.000 .. 59.999 CIRCULAR [INTERLIS.s];
  END GregorianDateTime;

  DOMAIN XMLTime = FORMAT BASED ON UTC ( Hours/2 ":" Minutes/2 ":" Seconds/2 );
  DOMAIN XMLDate = FORMAT BASED ON GregorianDate ( Year/4 "-" Month/2 "-" Day/2 );
  DOMAIN XMLDateTime EXTENDS XMLDate = FORMAT BASED ON GregorianDateTime
                                            ( INHERITANCE "T" Hours/2 ":"
Minutes/2
                                                           ":" Seconds/2 );

  STRUCTURE LineSegment (ABSTRACT) =
    SegmentEndPoint: MANDATORY LineCoord;
  END LineSegment;

  STRUCTURE StartSegment (FINAL) EXTENDS LineSegment =
  END StartSegment;

  STRUCTURE StraightSegment (FINAL) EXTENDS LineSegment =
  END StraightSegment;

  STRUCTURE ArcSegment (FINAL) EXTENDS LineSegment =
    ArcPoint: MANDATORY LineCoord;
    Radius: NUMERIC [LENGTH];
```

```
    END ArcSegment;

    STRUCTURE SurfaceEdge =
      Geometry: DIRECTED POLYLINE;
    END SurfaceEdge;

    STRUCTURE SurfaceBoundary =
      Lines: LIST OF SurfaceEdge;
    END SurfaceBoundary;

    STRUCTURE LineGeometry =
      Segments: LIST OF LineSegment;
    MANDATORY CONSTRAINT isOfClass (Segments[FIRST],StartSegment);
    END LineGeometry;

  END INTERLIS.
```

Annex B (normative) – The XML-schema for the model INTERLIS

The following XML-schema defines basic types for the schema-generating-rule according to chapter 3.4 and the model INTERLIS according to Appendix A.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.interlis.ch/xtf/2.4/INTERLIS"
  targetNamespace="http://www.interlis.ch/xtf/2.4/INTERLIS"
  xmlns:geom="http://www.interlis.ch/geometry/1.0" elementFormDefault="qualified"
  attributeFormDefault="unqualified" version="2014-08-05">
  <xsd:import namespace="http://www.interlis.ch/geometry/1.0"/>

  <xsd:element name="extensions">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:attribute name="tid" type="tidType"/>
  <xsd:attribute name="ref" type="tidType"/>
  <xsd:attribute name="operation" type="operationType"/>
  <xsd:attribute name="order_pos" type="orderposType"/>
  <xsd:attribute name="bid" type="bidType"/>
  <xsd:attribute name="consistency" type="consistencyType"/>
  <xsd:attribute name="domains" type="domainsType"/>
  <xsd:attribute name="kind" type="transferKindType"/>
  <xsd:attribute name="startstate" type="basketStateType"/>
  <xsd:attribute name="endstate" type="basketStateType"/>

  <xsd:simpleType name="tidType">
    <xsd:restriction base="xsd:token"/>
  </xsd:simpleType>
  <xsd:simpleType name="refType">
    <xsd:restriction base="xsd:token"/>
  </xsd:simpleType>
  <xsd:simpleType name="orderposType">
    <xsd:restriction base="xsd:positiveInteger"/>
  </xsd:simpleType>
  <xsd:simpleType name="bidType">
    <xsd:restriction base="xsd:token"/>
  </xsd:simpleType>
  <xsd:simpleType name="consistencyType">
    <xsd:restriction base="xsd:token">
      <xsd:enumeration value="COMPLETE"/>
      <xsd:enumeration value="INCOMPLETE"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="domainsType">
    <xsd:restriction base="xsd:NMTOKENS"> </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="transferKindType">
    <xsd:restriction base="xsd:token">
      <xsd:enumeration value="FULL"/>
      <xsd:enumeration value="UPDATE"/>
      <xsd:enumeration value="INITIAL"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="operationType">
    <xsd:restriction base="xsd:token">
      <xsd:enumeration value="INSERT"/>
      <xsd:enumeration value="UPDATE"/>
```

```
        <xsd:enumeration value="DELETE"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="basketStateType">
      <xsd:restriction base="xsd:token"/>
    </xsd:simpleType>

    <xsd:element name="transfer" type="TransferType"/>
    <xsd:complexType name="TransferType">
      <xsd:sequence>
        <xsd:element name="headersection">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="models">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="model" type="xsd:token"
  maxOccurs="unbounded"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="sender" type="xsd:string" minOccurs="0"/>
              <xsd:element name="comment" type="xsd:string" minOccurs="0"/>
              <xsd:element ref="extensions" minOccurs="0"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="datasection">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="strict"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:simpleType name="HALIGNMENT">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Left"/>
        <xsd:enumeration value="Center"/>
        <xsd:enumeration value="Right"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="VALIGNMENT">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Top"/>
        <xsd:enumeration value="Cap"/>
        <xsd:enumeration value="Half"/>
        <xsd:enumeration value="Base"/>
        <xsd:enumeration value="Bottom"/>
      </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="I32OID">
      <xsd:restriction base="xsd:int">
        <xsd:minInclusive value="0"/>
      </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="STANDARDOID">
      <xsd:restriction base="xsd:token">
        <xsd:length value="16"/>
        <xsd:pattern value="[a-zA-Z][a-zA-Z0-9]*"/>
      </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="UUIDOID">
```

```
      <xsd:restriction base="xsd:token">
        <xsd:length value="36"/>
        <xsd:pattern
  value="[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"/>
      </xsd:restriction>
    </xsd:simpleType>


    <xsd:element name="METAOBJECT" type="METAOBJECTType"/>
    <xsd:complexType name="METAOBJECTType">
      <xsd:sequence>
        <xsd:element ref="extensions" minOccurs="0"/>
        <xsd:element name="Name">
          <xsd:simpleType>
            <xsd:restriction base="xsd:token">
              <xsd:maxLength value="255"/>
              <xsd:pattern value="[a-zA-Z][a-zA-Z0-9_]*"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute ref="tid" use="required"/>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <xsd:element name="METAOBJECT_TRANSLATION" type="METAOBJECT_TRANSLATIONType"/>
    <xsd:complexType name="METAOBJECT_TRANSLATIONType">
      <xsd:sequence>
        <xsd:element ref="extensions" minOccurs="0"/>
        <xsd:element name="Name">
          <xsd:simpleType>
            <xsd:restriction base="xsd:token">
              <xsd:maxLength value="255"/>
              <xsd:pattern value="[a-zA-Z][a-zA-Z0-9_]*"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="NameInBaseLanguage">
          <xsd:simpleType>
            <xsd:restriction base="xsd:token">
              <xsd:maxLength value="255"/>
              <xsd:pattern value="[a-zA-Z][a-zA-Z0-9_]*"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute ref="tid" use="required"/>
      <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <xsd:element name="AXIS" type="AXISType"/>
    <xsd:complexType name="AXISType">
      <xsd:sequence> </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="REFSYSTEM" type="REFSYSTEMType"
  substitutionGroup="METAOBJECT"/>
    <xsd:complexType name="REFSYSTEMType">
      <xsd:complexContent>
        <xsd:extension base="METAOBJECTType">
          <xsd:sequence> </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="COORDSYSTEM" type="COORDSYSTEMType"
  substitutionGroup="REFSYSTEM"/>
    <xsd:complexType name="COORDSYSTEMType">
      <xsd:complexContent>
        <xsd:extension base="REFSYSTEMType">
          <xsd:sequence>
            <xsd:element name="Axis" maxOccurs="3">
```

```
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element ref="AXIS"/>
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="SCALSYSTEM" type="SCALSYSTEMType"
  substitutionGroup="REFSYSTEM"/>
    <xsd:complexType name="SCALSYSTEMType">
      <xsd:complexContent>
        <xsd:extension base="REFSYSTEMType">
          <xsd:sequence> </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="SIGN" type="SIGNType" substitutionGroup="METAOBJECT"/>
    <xsd:complexType name="SIGNType">
      <xsd:complexContent>
        <xsd:extension base="METAOBJECTType">
          <xsd:sequence> </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="CALENDAR" type="CALENDARType" substitutionGroup="SCALSYSTEM"/>
    <xsd:complexType name="CALENDARType">
      <xsd:complexContent>
        <xsd:extension base="SCALSYSTEMType">
          <xsd:sequence> </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="TIMEOFDAYSYS" type="TIMEOFDAYSYSType"
  substitutionGroup="SCALSYSTEM"/>
    <xsd:complexType name="TIMEOFDAYSYSType">
      <xsd:complexContent>
        <xsd:extension base="SCALSYSTEMType">
          <xsd:sequence> </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="TIMESYSTEMS" type="TIMESYSTEMSType"/>
    <xsd:complexType name="TIMESYSTEMSType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element ref="CALENDAR"/>
          <xsd:element ref="TIMEOFDAYSYS"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="TimeOfDay" type="TimeOfDayType"/>
    <xsd:complexType name="TimeOfDayType">
      <xsd:sequence>
        <xsd:element ref="extensions" minOccurs="0"/>
        <xsd:element name="Hours" minOccurs="0">
          <xsd:simpleType>
            <xsd:restriction base="xsd:integer">
              <xsd:minInclusive value="0"/>
              <xsd:maxInclusive value="23"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="Minutes" minOccurs="0">
          <xsd:simpleType>
            <xsd:restriction base="xsd:integer">
```

```
            <xsd:minInclusive value="0"/>
            <xsd:maxInclusive value="59"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="Seconds" minOccurs="0">
        <xsd:simpleType>
          <xsd:restriction base="xsd:decimal">
            <xsd:minInclusive value="0.0"/>
            <xsd:maxInclusive value="59.999"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>
  <xsd:element name="UTC" type="UTCType" substitutionGroup="TimeOfDay"/>
  <xsd:complexType name="UTCType">
    <xsd:complexContent>
      <xsd:extension base="TimeOfDayType">
        <xsd:sequence> </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="GregorianDate" type="GregorianDateType"/>
  <xsd:complexType name="GregorianDateType">
    <xsd:sequence>
      <xsd:element ref="extensions" minOccurs="0"/>
      <xsd:element name="Year" type="xsd:gYear" minOccurs="0"/>
      <xsd:element name="Month" minOccurs="0">
        <xsd:simpleType>
          <xsd:restriction base="xsd:integer">
            <xsd:minInclusive value="1"/>
            <xsd:maxInclusive value="12"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="Day" minOccurs="0">
        <xsd:simpleType>
          <xsd:restriction base="xsd:integer">
            <xsd:minInclusive value="1"/>
            <xsd:maxInclusive value="31"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>
  <xsd:element name="GregorianDateTime" type="GregorianDateTimeType"
    substitutionGroup="GregorianDate"/>
  <xsd:complexType name="GregorianDateTimeType">
    <xsd:complexContent>
      <xsd:extension base="GregorianDateType">
        <xsd:sequence>
          <xsd:element name="Hours" minOccurs="0">
            <xsd:simpleType>
              <xsd:restriction base="xsd:integer">
                <xsd:minInclusive value="0"/>
                <xsd:maxInclusive value="23"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:element>
          <xsd:element name="Minutes" minOccurs="0">
            <xsd:simpleType>
              <xsd:restriction base="xsd:integer">
                <xsd:minInclusive value="0"/>
                <xsd:maxInclusive value="59"/>
              </xsd:restriction>
```

```
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="Seconds" minOccurs="0">
        <xsd:simpleType>
          <xsd:restriction base="xsd:decimal">
            <xsd:minInclusive value="0.0"/>
            <xsd:maxInclusive value="59.999"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="LineSegment" type="LineSegmentType"
substitutionGroup="geom:customLineSegment"/>
 <xsd:complexType name="LineSegmentType">
  <xsd:complexContent>
   <xsd:extension base="geom:LineSegmentType"> </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="SurfaceEdge" type="SurfaceEdgeType"/>
 <xsd:complexType name="SurfaceEdgeType">
  <xsd:sequence>
   <xsd:element ref="extensions" minOccurs="0"/>
   <xsd:element name="Geometry" minOccurs="0" type="geom:PolylineType"/>
  </xsd:sequence>
  <xsd:anyAttribute processContents="lax"/>
 </xsd:complexType>
 <xsd:element name="SurfaceBoundary" type="SurfaceBoundaryType"/>
 <xsd:complexType name="SurfaceBoundaryType">
  <xsd:sequence>
   <xsd:element ref="extensions" minOccurs="0"/>
   <xsd:element name="Lines" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element ref="SurfaceEdge"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
  <xsd:anyAttribute processContents="lax"/>
 </xsd:complexType>
 <xsd:element name="LineGeometry" type="LineGeometryType"/>
 <xsd:complexType name="LineGeometryType">
  <xsd:sequence>
   <xsd:element ref="extensions" minOccurs="0"/>
   <xsd:element name="Segments" type="geom:LineSegmentType"
maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:anyAttribute processContents="lax"/>
 </xsd:complexType>

</xsd:schema>
```

Annex C (normative) – The XML-schema for geometry-types

The following XML-schema defines basic types for geometry-types.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.interlis.ch/geometry/1.0"
  targetNamespace="http://www.interlis.ch/geometry/1.0"
elementFormDefault="qualified"
  attributeFormDefault="unqualified" version="2014-08-15">

  <xsd:element name="extensions">
    <xsd:annotation>
      <xsd:documentation>any vendor specifics</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:attribute name="epsg" type="xsd:string"/>

  <xsd:element name="geometry" type="GeometryType"/>
  <xsd:complexType name="GeometryType" abstract="true">
    <xsd:sequence>
      <xsd:element ref="extensions" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute ref="epsg" use="optional"/>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>

  <xsd:element name="coord" type="CoordType" substitutionGroup="geometry"/>
  <xsd:complexType name="CoordType">
    <xsd:complexContent>
      <xsd:extension base="GeometryType">
        <xsd:sequence>
          <xsd:element name="c1" type="xsd:double" minOccurs="1" maxOccurs="1"/>
          <xsd:element name="c2" type="xsd:double" minOccurs="1" maxOccurs="1"/>
          <xsd:element name="c3" type="xsd:double" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="multicoord" type="MultiCoordType"/>
  <xsd:complexType name="MultiCoordType">
    <xsd:sequence>
      <xsd:element ref="coord" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="customLineSegment" type="LineSegmentType"/>
  <xsd:complexType name="LineSegmentType" abstract="true">
    <xsd:sequence>
      <xsd:element ref="extensions" minOccurs="0"/>
      <xsd:element name="c1" type="xsd:double" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="c2" type="xsd:double" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="c3" type="xsd:double" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:anyAttribute processContents="lax"/>
  </xsd:complexType>
```

```
  <xsd:complexType name="StartSegmentType">
    <xsd:complexContent>
      <xsd:extension base="LineSegmentType">
        <xsd:sequence> </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="StraightSegmentType">
    <xsd:complexContent>
      <xsd:extension base="LineSegmentType">
        <xsd:sequence> </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="ArcSegmentType">
    <xsd:complexContent>
      <xsd:extension base="LineSegmentType">
        <xsd:sequence>
          <xsd:element name="a1" type="xsd:double" minOccurs="1" maxOccurs="1"/>
          <xsd:element name="a2" type="xsd:double" minOccurs="1" maxOccurs="1"/>
          <xsd:element name="r" type="xsd:double" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="polyline" type="PolylineType" substitutionGroup="geometry"/>
  <xsd:complexType name="PolylineType">
    <xsd:complexContent>
      <xsd:extension base="GeometryType">
        <xsd:sequence>
          <xsd:element ref="coord" minOccurs="1" maxOccurs="1"/>
          <xsd:choice minOccurs="1" maxOccurs="unbounded">
            <xsd:element ref="coord" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="arc" type="ArcSegmentType" minOccurs="1"
maxOccurs="1"/>
            <xsd:element ref="customLineSegment" minOccurs="1" maxOccurs="1"/>
          </xsd:choice>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="multipolyline" type="MultiPolylineType"/>
  <xsd:complexType name="MultiPolylineType">
    <xsd:sequence>
      <xsd:element ref="polyline" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="BoundaryType">
    <xsd:sequence>
      <xsd:element ref="polyline" minOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="surface" type="SurfaceType" substitutionGroup="geometry"/>
  <xsd:complexType name="SurfaceType">
    <xsd:complexContent>
      <xsd:extension base="GeometryType">
        <xsd:sequence>
          <xsd:element name="exterior" type="BoundaryType" minOccurs="1"
maxOccurs="unbounded"/>
          <xsd:element name="interior" type="BoundaryType" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
```

```
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="multisurface" type="MultiSurfaceType"/>
  <xsd:complexType name="MultiSurfaceType">
    <xsd:sequence>
      <xsd:element ref="surface" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

Annex D (normative for CH) – Symbol table

The table below shows a list of all symbols that a software must be able to process (such as save, search sort, print), unless explicitly stated otherwise in the model (cf. chapter 2.5 Models, topics, classes). If during transfer symbols appear which do not figure in the character set defined by the model, the software will tacitly (or maybe with a note, but without error message) replace such a symbol.

For some symbols several coding forms are available in XML. In such a case the most common coding forms of the symbol are indicated. If there are several coding possibilities per symbol, an INTERLIS writing program will arbitrarily select one of them. Hence an INTERLIS 2-reading programme must be capable of recognising any symbol and all of ist possible coding forms.

The table only applies to XML-content (i.e. XML-String, XML-NormalizedString resp. XML-Value). XML-tags are transmitted exclusively as ASCII-coded symbols in accordance with the syntax of chapter 3.

Tabulator (TAB, #x9), Carriage return (CR, #xD) and Line feed (LF, #xA) are the only admissible control codes. However they may only occur within the scope of the coding of MTEXT (cf. chapter 2.8.1 Strings and chapter 3.3.11.2 Coding of strings).

UCS Hex	UCS Dec	UTF-8 Coding Octet Hex	XML Coding Character Reference Dec	XML Coding Character Reference Hex	XML Coding Entity Reference	Represented as
0020	32	20				
0021	33	21				!
0022	34		"	"	"	"
0023	35	23				#
0024	36	24				$
0025	37	25				%
0026	38		&	&	&	&
0027	39		'	'	'	'
0028	40	28				(
0029	41	29				)
002A	42	2A				*
002B	43	2B				+
002C	44	2C				,
002D	45	2D				-
002E	46	2E				.
002F	47	2F				/
0030	48	30				0
0031	49	31				1

UCS Hex	UCS Dec	UTF-8 Coding Octet Hex	XML Coding Character Reference Dec	XML Coding Character Reference Hex	XML Coding Entity Reference	Represen-ted as
0032	50	32				2
0033	51	33				3
0034	52	34				4
0035	53	35				5
0036	54	36				6
0037	55	37				7
0038	56	38				8
0039	57	39				9
003A	58	3A				:
003B	59	3B				;
003C	60		<	<	<	<
003D	61	3D				=
003E	62		>	>	>	>
003F	63	3F				?
0040	64	40				@
0041	65	41				A
0042	66	42				B
0043	67	43				C
0044	68	44				D
0045	69	45				E
0046	70	46				F
0047	71	47				G
0048	72	48				H
0049	73	49				I
004A	74	4A				J
004B	75	4B				K
004C	76	4C				L
004D	77	4D				M
004E	78	4E				N
004F	79	4F				O
0050	80	50				P
0051	81	51				Q

UCS Hex	UCS Dec	UTF-8 Coding Octet Hex	XML Coding Character Reference Dec	XML Coding Character Reference Hex	XML Coding Entity Reference	Represen-ted as
0052	82	52				R
0053	83	53				S
0054	84	54				T
0055	85	55				U
0056	86	56				V
0057	87	57				W
0058	88	58				X
0059	89	59				Y
005A	90	5A				Z
005B	91	5B				[
005C	92	5C				\
005D	93	5D				]
005E	94	5E				^
005F	95	5F				_
0060	96	60				`
0061	97	61				a
0062	98	62				b
0063	99	63				c
0064	100	64				d
0065	101	65				e
0066	102	66				f
0067	103	67				g
0068	104	68				h
0069	105	69				i
006A	106	6A				j
006B	107	6B				k
006C	108	6C				l
006D	109	6D				m
006E	110	6E				n
006F	111	6F				o
0070	112	70				p
0071	113	71				q

UCS Hex	UCS Dec	UTF-8 Coding Octet Hex	XML Coding Character Reference Dec	XML Coding Character Reference Hex	XML Coding Entity Reference	Represen-ted as
0072	114	72				r
0073	115	73				s
0074	116	74				t
0075	117	75				u
0076	118	76				v
0077	119	77				w
0078	120	78				x
0079	121	79				y
007A	122	7A				z
007B	123	7B				{
007C	124	7C				|
007D	125	7D				}
007E	126	7E				~
00A3	163	C2 A3	£	£		£
00A7	167	C2 A7	§	§		§
00AB	171	C2 AB	«	«		«
00BB	187	C2 BB	»	»		»
00C0	192	C3 80	À	À		À
00C1	193	C3 81	Á	Á		Á
00C2	194	C3 82	Â	Â		Â
00C3	195	C3 83	Ã	Ã		Ã
00C4	196	C3 84	Ä	Ä		Ä
00C5	197	C3 85	Å	Å		Å
00C6	198	C3 86	Æ	Æ		Æ
00C7	199	C3 87	Ç	Ç		Ç
00C8	200	C3 88	È	È		È
00C9	201	C3 89	É	É		É
00CA	202	C3 8A	Ê	Ê		Ê
00CB	203	C3 8B	Ë	Ë		Ë
00CC	204	C3 8C	Ì	Ì		Ì
00CD	205	C3 8D	Í	Í		Í
00CE	206	C3 8E	Î	Î		Î

UCS Hex	UCS Dec	UTF-8 Coding Octet Hex	XML Coding Character Reference Dec	XML Coding Character Reference Hex	XML Coding Entity Reference	Represen-ted as
00CF	207	C3 8F	Ï	Ï		Ï
00D0	208	C3 90	Ð	Ð		Ð
00D1	209	C3 91	Ñ	Ñ		Ñ
00D2	210	C3 92	Ò	Ò		Ò
00D3	211	C3 93	Ó	Ó		Ó
00D4	212	C3 94	Ô	Ô		Ô
00D5	213	C3 95	Õ	Õ		Õ
00D6	214	C3 96	Ö	Ö		Ö
00D8	216	C3 98	Ø	Ø		Ø
00D9	217	C3 99	Ù	Ù		Ù
00DA	218	C3 9A	Ú	Ú		Ú
00DB	219	C3 9B	Û	Û		Û
00DC	220	C3 9C	Ü	Ü		Ü
00DD	221	C3 9D	Ý	Ý		Ý
00DE	222	C3 9E	Þ	Þ		Þ
00DF	223	C3 9F	ß	ß		ß
00E0	224	C3 A0	à	à		à
00E1	225	C3 A1	á	á		á
00E2	226	C3 A2	â	â		â
00E3	227	C3 A3	ã	ã		ã
00E4	228	C3 A4	ä	ä		ä
00E5	229	C3 A5	å	å		å
00E6	230	C3 A6	æ	æ		æ
00E7	231	C3 A7	ç	ç		ç
00E8	232	C3 A8	è	è		è
00E9	233	C3 A9	é	é		é
00EA	234	C3 AA	ê	ê		ê
00EB	235	C3 AB	ë	ë		ë
00EC	236	C3 AC	ì	ì		ì
00ED	237	C3 AD	í	í		í
00EE	238	C3 AE	î	î		î
00EF	239	C3 AF	ï	ï		ï

UCS Hex	UCS Dec	UTF-8 Coding Octet Hex	XML Coding Character Reference Dec	XML Coding Character Reference Hex	XML Coding Entity Reference	Represen-ted as
00F0	240	C3 B0	ð	ð		ð
00F1	241	C3 B1	ñ	ñ		ñ
00F2	242	C3 B2	ò	ò		ò
00F3	243	C3 B3	ó	ó		ó
00F4	244	C3 B4	ô	ô		ô
00F5	245	C3 B5	õ	õ		õ
00F6	246	C3 B6	ö	ö		ö
00F8	248	C3 B8	ø	ø		ø
00F9	249	C3 B9	ù	ù		ù
00FA	250	C3 BA	ú	ú		ú
00FB	251	C3 BB	û	û		û
00FC	252	C3 BC	ü	ü		Ü
00FD	253	C3 BD	ý	ý		ý
00FE	254	C3 BE	þ	þ		þ
00FF	255	C3 BF	ÿ	ÿ		ÿ
20AC	8364	E2 82 AC	€	€		€

Table 2: UCS/Unicode-symbols and their coding to be employed by default in INTERLIS 2.

Notes:

- In the columns UCS/Hex resp. UCS/Decimal the UCS (resp. Unicode) code of the symbol has been indicated (hexadecimal, resp. decimal).
- In the column UTF-8 coding the coding of the symbol has been indicated according to UTF-8 as 8bit bytes (octet) in hexadecimal notation. The symbols >= Hex 80 are coded as multiple byte sequences. Note: The hexadecimal notation is only used to illustrate the coding. Only binary coded octets are transmitted on the transfer-file.
- In the columns XML coding character reference (Dec), resp. character reference (Hex) the coding of the symbol is indicated as XML character reference (decimal, resp. hexadecimal variant). This value is an ASCII-symbol sequence and must be used strictly accurate in the transfer. Whenever possible a writing program should select the character reference coding for the symbol. The advantage of character reference coding lies in its ability to be indicated, resp. edited with simple ASCII-editors on any platform (Unix, PC etc.). Note: With the hexadecimal coding variant the letters a-f can be listed in capitals or small letters (however the x in the hexadecimal variant must always be a small letter).
- The XML-coding (Entity Reference) is only admissible for some few special symbols. Its value is an ASCII-sequence which has to be used accurately in the transfer. Note: XML-entities such as ü (for the symbol ü) are not admissible in an INTERLIS 2 transfer-file, since no DTD will be referenced by an INTERLIS 2 transfer-file (permissible entities such as & have been predefined in the XML 1.0 specification).

– In the column representation you will find the representation of the symbol in a UCS, resp. Unicode compatible editor.

Annex E (informative) – A small example Roads

Introduction

In order to facilitate your entry into INTERLIS 2, we provide a small but nevertheless complete example. It describes a data set designed for a complete data transfer. For examples of application with incremental update (including OID), we supply the necessary test data sets and user manuals.

The example consists of the following parts:

- Data model RoadsExdm2ben and RoadsExdm2ien.
- XML-data set RoadsExdm2ien (file RoadsExdm2ien.xtf) which contains objects in accordance with the data model RoadsExdm2ien.
- Graphic model RoadsExgm2ien. One possible representation for the data model RoadsExdm2ien is defined in the graphic model (Note: any number of representations is possible for one single data model).
- Collection of symbol objects (symbol library) in RoadsExgm2ien_Symbols (file RoadsExgm2ien_Symbols.xtf). The symbol library is an XML-data set in accordance with the symbology model StandardSymbology (cf. appendix L *Symbology models*). The symbol library is used in the graphic model RoadsExgm2ien for the representation of exemplifying data from the RoadsExdm2ien.xtf-data set.

The name "RoadsExdm2ben" is an abbreviation of "**RoadsEx**ample, **d**ata **m**odel, INTERLIS **2**, **b**asic model, **en**glish". Hereafter the individual parts will be described more in detail.

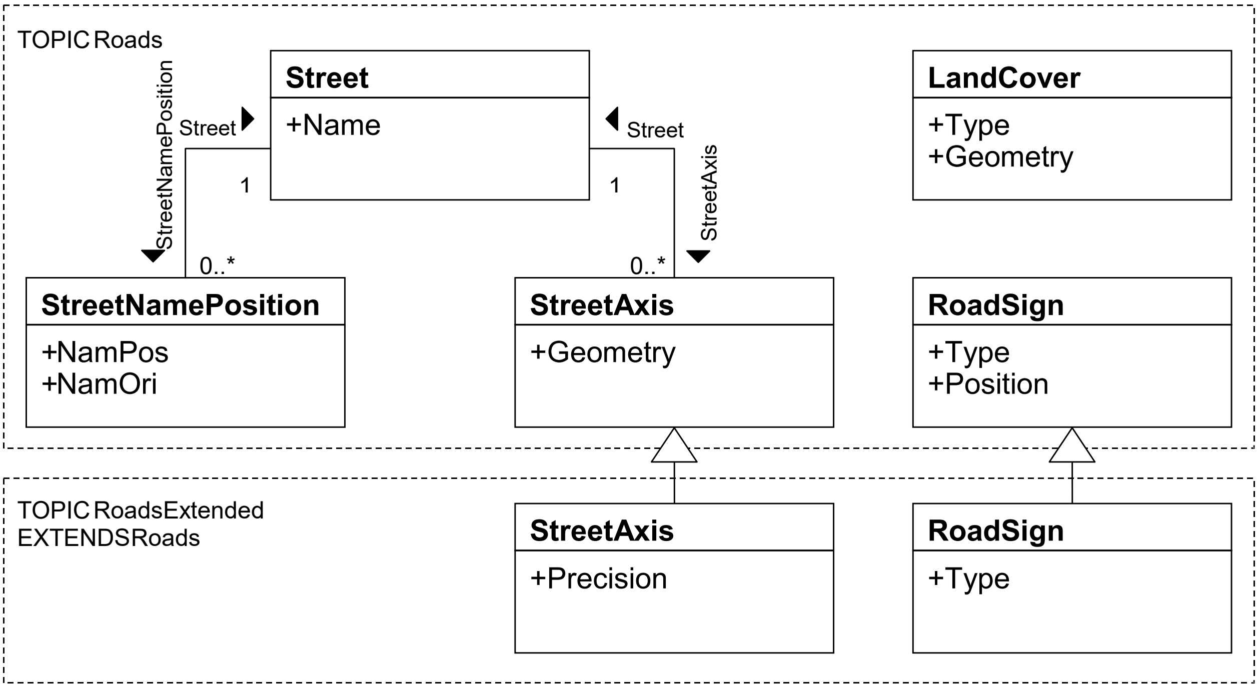


Figure 26: UML-class diagram of data models.

Data model RoadsExdm2ben and RoadsExdm2ien

The data model RoadsExdm2ben contains the objects LandCover, StreetAxis, StreetName and PointObject. The data model RoadsExdm2ien is an extension of RoadsExdm2ben. The UML-class diagram above (see figure 26) supplies a comprehensive over-view of the data models.

Note: It is on purpose that the example chosen is very simple and we do not lay claim to completeness. The corresponding models defined in INTERLIS 2 read as follows:

```
!! File RoadsExdm2ben.ili Release 2014-07-09

INTERLIS 2.4;

MODEL RoadsExdm2ben (en) AT "http://www.interlis.ch/models/refhb24"
  VERSION "2014-07-09" =

  UNIT
    Angle_Degree = 180 / PI [INTERLIS.rad];

  DOMAIN
    Point2D = COORD
      0.000 .. 200.000 [INTERLIS.m], !! Min_East  Max_East
      0.000 .. 200.000 [INTERLIS.m], !! Min_North Max_North
      ROTATION 2 -> 1;
    Orientation = 0.0 .. 359.9 CIRCULAR [Angle_Degree];


  TOPIC Roads =

    CLASS LandCover =
      Type: MANDATORY (
        building,
        street,
        water,
        other);
      Geometry: MANDATORY SURFACE WITH (STRAIGHTS)
        VERTEX Point2D WITHOUT OVERLAPS > 0.100
    END LandCover;

    CLASS Street =
      Name: MANDATORY TEXT*32;
    END Street;

    CLASS StreetAxis =
      Geometry: MANDATORY POLYLINE WITH (STRAIGHTS)
        VERTEX Point2D;
    END StreetAxis;

    ASSOCIATION StreetAxisAssoc =
      Street -- {1} Street;
      StreetAxis -- StreetAxis;
    END StreetAxisAssoc;

    CLASS StreetNamePosition =
      NamPos: MANDATORY Point2D;
      NamOri: MANDATORY Orientation;
    END StreetNamePosition;

    ASSOCIATION StreetNamePositionAssoc =
      Street -- {0..1} Street;
      StreetNamePosition -- StreetNamePosition;
    END StreetNamePositionAssoc;

    CLASS RoadSign =
      Type: MANDATORY (
        prohibition,
        indication,
        danger,
        velocity);
      Position: MANDATORY Point2D;
    END RoadSign;

  END Roads; !! of TOPIC
```

```
END RoadsExdm2ben. !! of MODEL


!! File RoadsExdm2ien.ili Release 2014-07-09

INTERLIS 2.4;

MODEL RoadsExdm2ien (en) AT "http://www.interlis.ch/models/refhb24"
  VERSION "2014-07-09" =

  IMPORTS RoadsExdm2ben;


  TOPIC RoadsExtended EXTENDS RoadsExdm2ben.Roads =

    CLASS StreetAxis (EXTENDED) =
      Precision: MANDATORY (
        precise,
        unprecise);
    END StreetAxis;

    CLASS RoadSign (EXTENDED) =
      Type (EXTENDED): (
        prohibition (
          noentry,
          noparking,
          other));
    END RoadSign;

  END RoadsExtended; !! of TOPIC

END RoadsExdm2ien. !! of MODEL
```

Data set RoadsExdm2ien in accordance with data model RoadsExdm2ien

Below you will find an exemplary data set for the data model RoadsExdm2ien. The XML-formatting has been derived from the data model RoadsExdm2ien by means of the rules stated in chapter 3 Sequential Transfer.

```
<!-- File RoadsExdm2ien.xtf 2014-08-05 (http://www.interlis.ch/models/refhb24) -->

<?xml version="1.0" encoding="UTF-8"?>
<ili:transfer xmlns:ili="http://www.interlis.ch/xtf/2.4/INTERLIS"
  xmlns:geom="http://www.interlis.ch/geometry/1.0"
  xmlns:roads="http://www.interlis.ch/xtf/2.4/RoadsExdm2ben"
  xmlns="http://www.interlis.ch/xtf/2.4/RoadsExdm2ien"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ili:headersection>
    <ili:models>
      <ili:model>RoadsExdm2ien</ili:model>
    </ili:models>
    <ili:sender>KOGIS</ili:sender>
    <ili:comment>example dataset ili2 refmanual appendix C</ili:comment>
  </ili:headersection>

  <ili:datasection>
    <RoadsExtended ili:bid="REFHANDB00000001">

      <!-- === LandCover === -->
      <roads:LandCover ili:tid="16">
        <roads:Type>water</roads:Type>
        <roads:Geometry>
          <geom:surface>
            <geom:exterior>
              <geom:polyline>
                <geom:coord>
```

```
                <geom:c1>39.038</geom:c1><geom:c2>60.315</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>41.200</geom:c1><geom:c2>59.302</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>43.362</geom:c1><geom:c2>60.315</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>44.713</geom:c1><geom:c2>66.268</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>45.794</geom:c1><geom:c2>67.662</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>48.766</geom:c1><geom:c2>67.408</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>53.360</geom:c1><geom:c2>64.115</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>56.197</geom:c1><geom:c2>62.595</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>57.818</geom:c1><geom:c2>63.862</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>58.899</geom:c1><geom:c2>68.928</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>55.927</geom:c1><geom:c2>72.348</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>47.955</geom:c1><geom:c2>75.515</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>42.281</geom:c1><geom:c2>75.388</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>39.308</geom:c1><geom:c2>73.235</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>36.741</geom:c1><geom:c2>69.688</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>35.525</geom:c1><geom:c2>66.268</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>35.661</geom:c1><geom:c2>63.735</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>37.957</geom:c1><geom:c2>61.455</geom:c2>
              </geom:coord>
              <geom:coord>
                <geom:c1>39.038</geom:c1><geom:c2>60.315</geom:c2>
              </geom:coord>
            </geom:polyline>
          </geom:exterior>
        </geom:surface>
      </roads:Geometry>
    </roads:LandCover>

    <roads:LandCover ili:tid="18">
      <roads:Type>building</roads:Type>
      <roads:Geometry>
        <geom:surface>
          <geom:exterior>
            <geom:polyline>
```

```
          <geom:coord>
            <geom:c1>101.459</geom:c1><geom:c2>65.485</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>108.186</geom:c1><geom:c2>69.369</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>102.086</geom:c1><geom:c2>79.936</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>95.359</geom:c1><geom:c2>76.053</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>101.459</geom:c1><geom:c2>65.485</geom:c2>
          </geom:coord>
        </geom:polyline>
     </geom:exterior>
    </geom:surface>
  </roads:Geometry>
</roads:LandCover>

<roads:LandCover ili:tid="20">
  <roads:Type>building</roads:Type>
  <roads:Geometry>
    <geom:surface>
      <geom:exterior>
        <geom:polyline>
          <geom:coord>
            <geom:c1>60.489</geom:c1><geom:c2>49.608</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>79.900</geom:c1><geom:c2>55.839</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>75.351</geom:c1><geom:c2>70.932</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>67.678</geom:c1><geom:c2>68.781</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>69.938</geom:c1><geom:c2>61.721</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>57.582</geom:c1><geom:c2>58.029</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>60.489</geom:c1><geom:c2>49.608</geom:c2>
          </geom:coord>
        </geom:polyline>
     </geom:exterior>
    </geom:surface>
  </roads:Geometry>
</roads:LandCover>

<roads:LandCover ili:tid="22">
  <roads:Type>street</roads:Type>
  <roads:Geometry>
    <geom:surface>
      <geom:exterior>
        <geom:polyline>
          <geom:coord>
            <geom:c1>45.067</geom:c1><geom:c2>58.655</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>50.669</geom:c1><geom:c2>42.579</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>57.060</geom:c1><geom:c2>44.638</geom:c2>
```

```
            </geom:coord>
            <geom:coord>
              <geom:c1>51.432</geom:c1><geom:c2>60.469</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>45.067</geom:c1><geom:c2>58.655</geom:c2>
            </geom:coord>
          </geom:polyline>
        </geom:exterior>
      </geom:surface>
    </roads:Geometry>
  </roads:LandCover>

  <roads:LandCover ili:tid="24">
    <roads:Type>other</roads:Type>
    <roads:Geometry>
      <geom:surface>
        <geom:exterior>
          <geom:polyline>
            <geom:coord>
              <geom:c1>114.027</geom:c1><geom:c2>99.314</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>31.351</geom:c1><geom:c2>99.314</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>31.140</geom:c1><geom:c2>92.530</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>70.419</geom:c1><geom:c2>86.177</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>86.481</geom:c1><geom:c2>79.710</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>114.027</geom:c1><geom:c2>99.314</geom:c2>
            </geom:coord>
          </geom:polyline>
        </geom:exterior>
      </geom:surface>
    </roads:Geometry>
  </roads:LandCover>

  <roads:LandCover ili:tid="26">
    <roads:Type>other</roads:Type>
    <roads:Geometry>
      <geom:surface>
        <geom:exterior>
          <geom:polyline>
            <geom:coord>
              <geom:c1>113.559</geom:c1><geom:c2>62.880</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>114.027</geom:c1><geom:c2>99.314</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>86.481</geom:c1><geom:c2>79.710</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>94.381</geom:c1><geom:c2>66.289</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>96.779</geom:c1><geom:c2>57.177</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>113.559</geom:c1><geom:c2>62.880</geom:c2>
            </geom:coord>
          </geom:polyline>
```

```
        </geom:exterior>
        <geom:interior>
          <geom:polyline>
            <geom:coord>
              <geom:c1>108.186</geom:c1><geom:c2>69.369</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>101.459</geom:c1><geom:c2>65.485</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>95.359</geom:c1><geom:c2>76.053</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>102.086</geom:c1><geom:c2>79.936</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>108.186</geom:c1><geom:c2>69.369</geom:c2>
            </geom:coord>
          </geom:polyline>
        </geom:interior>
      </geom:surface>
    </roads:Geometry>
  </roads:LandCover>

  <roads:LandCover ili:tid="29">
    <roads:Type>street</roads:Type>
    <roads:Geometry>
      <geom:surface>
        <geom:exterior>
          <geom:polyline>
            <geom:coord>
              <geom:c1>100.621</geom:c1><geom:c2>24.239</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>109.729</geom:c1><geom:c2>24.239</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>105.640</geom:c1><geom:c2>48.068</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>96.779</geom:c1><geom:c2>45.088</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>100.621</geom:c1><geom:c2>24.239</geom:c2>
            </geom:coord>
          </geom:polyline>
        </geom:exterior>
      </geom:surface>
    </roads:Geometry>
  </roads:LandCover>

  <roads:LandCover ili:tid="31">
    <roads:Type>other</roads:Type>
    <roads:Geometry>
      <geom:surface>
        <geom:exterior>
          <geom:polyline>
            <geom:coord>
              <geom:c1>30.900</geom:c1><geom:c2>24.478</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>100.621</geom:c1><geom:c2>24.239</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>96.779</geom:c1><geom:c2>45.088</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>30.900</geom:c1><geom:c2>24.478</geom:c2>
```

```
            </geom:coord>
          </geom:polyline>
        </geom:exterior>
      </geom:surface>
    </roads:Geometry>
  </roads:LandCover>

  <roads:LandCover ili:tid="33">
    <roads:Type>other</roads:Type>
    <roads:Geometry>
      <geom:surface>
        <geom:exterior>
          <geom:polyline>
            <geom:coord>
              <geom:c1>31.110</geom:c1><geom:c2>83.750</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>31.140</geom:c1><geom:c2>36.458</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>50.669</geom:c1><geom:c2>42.579</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>45.067</geom:c1><geom:c2>58.655</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>51.432</geom:c1><geom:c2>60.469</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>57.060</geom:c1><geom:c2>44.638</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>87.839</geom:c1><geom:c2>54.138</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>85.282</geom:c1><geom:c2>63.410</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>78.847</geom:c1><geom:c2>73.433</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>67.549</geom:c1><geom:c2>77.788</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>31.110</geom:c1><geom:c2>83.750</geom:c2>
            </geom:coord>
          </geom:polyline>
        </geom:exterior>
        <geom:interior>
          <geom:polyline>
            <geom:coord>
              <geom:c1>41.200</geom:c1><geom:c2>59.302</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>39.038</geom:c1><geom:c2>60.315</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>37.957</geom:c1><geom:c2>61.455</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>35.661</geom:c1><geom:c2>63.735</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>35.525</geom:c1><geom:c2>66.268</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>36.741</geom:c1><geom:c2>69.688</geom:c2>
            </geom:coord>
```

```
            <geom:coord>
              <geom:c1>39.308</geom:c1><geom:c2>73.235</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>42.281</geom:c1><geom:c2>75.388</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>47.955</geom:c1><geom:c2>75.515</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>55.927</geom:c1><geom:c2>72.348</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>58.899</geom:c1><geom:c2>68.928</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>57.818</geom:c1><geom:c2>63.862</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>56.197</geom:c1><geom:c2>62.595</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>53.360</geom:c1><geom:c2>64.115</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>48.766</geom:c1><geom:c2>67.408</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>45.794</geom:c1><geom:c2>67.662</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>44.713</geom:c1><geom:c2>66.268</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>43.362</geom:c1><geom:c2>60.315</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>41.200</geom:c1><geom:c2>59.302</geom:c2>
            </geom:coord>
          </geom:polyline>
        </geom:interior>
        <geom:interior>
          <geom:polyline>
            <geom:coord>
              <geom:c1>79.900</geom:c1><geom:c2>55.839</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>60.489</geom:c1><geom:c2>49.608</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>57.582</geom:c1><geom:c2>58.029</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>69.938</geom:c1><geom:c2>61.721</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>67.678</geom:c1><geom:c2>68.781</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>75.351</geom:c1><geom:c2>70.932</geom:c2>
            </geom:coord>
            <geom:coord>
              <geom:c1>79.900</geom:c1><geom:c2>55.839</geom:c2>
            </geom:coord>
          </geom:polyline>
        </geom:interior>
      </geom:surface>
    </roads:Geometry>
```

```
</roads:LandCover>

<roads:LandCover ili:tid="37">
  <roads:Type>street</roads:Type>
  <roads:Geometry>
    <geom:surface>
      <geom:exterior>
        <geom:polyline>
          <geom:coord>
            <geom:c1>31.140</geom:c1><geom:c2>92.530</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>31.110</geom:c1><geom:c2>83.750</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>67.549</geom:c1><geom:c2>77.788</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>78.847</geom:c1><geom:c2>73.433</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>85.282</geom:c1><geom:c2>63.410</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>87.839</geom:c1><geom:c2>54.138</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>96.779</geom:c1><geom:c2>57.177</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>94.381</geom:c1><geom:c2>66.289</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>86.481</geom:c1><geom:c2>79.710</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>70.419</geom:c1><geom:c2>86.177</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>31.140</geom:c1><geom:c2>92.530</geom:c2>
          </geom:coord>
        </geom:polyline>
      </geom:exterior>
    </geom:surface>
  </roads:Geometry>
</roads:LandCover>

<roads:LandCover ili:tid="39">
  <roads:Type>other</roads:Type>
  <roads:Geometry>
    <geom:surface>
      <geom:exterior>
        <geom:polyline>
          <geom:coord>
            <geom:c1>113.811</geom:c1><geom:c2>51.168</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>105.640</geom:c1><geom:c2>48.068</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>109.729</geom:c1><geom:c2>24.239</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>114.269</geom:c1><geom:c2>24.017</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>113.811</geom:c1><geom:c2>51.168</geom:c2>
          </geom:coord>
```

```
        </geom:polyline>
      </geom:exterior>
    </geom:surface>
  </roads:Geometry>
</roads:LandCover>

<roads:LandCover ili:tid="41">
  <roads:Type>street</roads:Type>
  <roads:Geometry>
    <geom:surface>
      <geom:exterior>
        <geom:polyline>
          <geom:coord>
            <geom:c1>105.640</geom:c1><geom:c2>48.068</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>113.811</geom:c1><geom:c2>51.168</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>113.559</geom:c1><geom:c2>62.880</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>96.779</geom:c1><geom:c2>57.177</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>87.839</geom:c1><geom:c2>54.138</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>57.060</geom:c1><geom:c2>44.638</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>50.669</geom:c1><geom:c2>42.579</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>31.140</geom:c1><geom:c2>36.458</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>30.900</geom:c1><geom:c2>24.478</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>96.779</geom:c1><geom:c2>45.088</geom:c2>
          </geom:coord>
          <geom:coord>
            <geom:c1>105.640</geom:c1><geom:c2>48.068</geom:c2>
          </geom:coord>
        </geom:polyline>
      </geom:exterior>
    </geom:surface>
  </roads:Geometry>
</roads:LandCover>

<!-- === Street === -->
<roads:Street ili:tid="1">
  <roads:Name>Austrasse</roads:Name>
</roads:Street>

<roads:Street ili:tid="2">
  <roads:Name>Eymattstrasse</roads:Name>
</roads:Street>

<roads:Street ili:tid="3">
  <roads:Name>Feldweg</roads:Name>
</roads:Street>

<roads:Street ili:tid="4">
  <roads:Name>Seeweg</roads:Name>
</roads:Street>
```

```xml
<!-- === StreetAxis / StreetAxisAssoc === -->
<StreetAxis ili:tid="8">
  <roads:Geometry>
    <geom:polyline>
      <geom:coord>
        <geom:c1>55.600</geom:c1><geom:c2>37.649</geom:c2>
      </geom:coord>
      <geom:coord>
        <geom:c1>15.573</geom:c1><geom:c2>25.785</geom:c2>
      </geom:coord>
    </geom:polyline>
  </roads:Geometry>
  <roads:Street ili:ref="1"></roads:Street>
  <Precision>precise</Precision>
</StreetAxis>

<StreetAxis ili:tid="9">
  <roads:Geometry>
    <geom:polyline>
      <geom:coord>
        <geom:c1>55.600</geom:c1><geom:c2>37.649</geom:c2>
      </geom:coord>
      <geom:coord>
        <geom:c1>94.990</geom:c1><geom:c2>50.109</geom:c2>
      </geom:coord>
    </geom:polyline>
  </roads:Geometry>
  <roads:Street ili:ref="1"></roads:Street>
  <Precision>precise</Precision>
</StreetAxis>

<StreetAxis ili:tid="10">
  <roads:Geometry>
    <geom:polyline>
      <geom:coord>
        <geom:c1>94.990</geom:c1><geom:c2>50.109</geom:c2>
      </geom:coord>
      <geom:coord>
        <geom:c1>101.099</geom:c1><geom:c2>52.279</geom:c2>
      </geom:coord>
    </geom:polyline>
  </roads:Geometry>
  <roads:Street ili:ref="1"></roads:Street>
  <Precision>precise</Precision>
</StreetAxis>

<StreetAxis ili:tid="11">
  <roads:Geometry>
    <geom:polyline>
      <geom:coord>
        <geom:c1>101.099</geom:c1><geom:c2>52.279</geom:c2>
      </geom:coord>
      <geom:coord>
        <geom:c1>126.100</geom:c1><geom:c2>62.279</geom:c2>
      </geom:coord>
    </geom:polyline>
  </roads:Geometry>
  <roads:Street ili:ref="1"></roads:Street>
  <Precision>precise</Precision>
</StreetAxis>

<StreetAxis ili:tid="12">
  <roads:Geometry>
    <geom:polyline>
      <geom:coord>
        <geom:c1>94.990</geom:c1><geom:c2>50.109</geom:c2>
      </geom:coord>
      <geom:coord>
```

```
        <geom:c1>89.504</geom:c1><geom:c2>65.795</geom:c2>
      </geom:coord>
      <geom:coord>
        <geom:c1>83.594</geom:c1><geom:c2>75.598</geom:c2>
      </geom:coord>
      <geom:coord>
        <geom:c1>71.774</geom:c1><geom:c2>80.712</geom:c2>
      </geom:coord>
      <geom:coord>
        <geom:c1>11.423</geom:c1><geom:c2>91.154</geom:c2>
      </geom:coord>
    </geom:polyline>
  </roads:Geometry>
  <roads:Street ili:ref="2"></roads:Street>
  <Precision>precise</Precision>
</StreetAxis>

<StreetAxis ili:tid="13">
  <roads:Geometry>
    <geom:polyline>
      <geom:coord>
        <geom:c1>101.099</geom:c1><geom:c2>52.279</geom:c2>
      </geom:coord>
      <geom:coord>
        <geom:c1>107.400</geom:c1><geom:c2>14.603</geom:c2>
      </geom:coord>
    </geom:polyline>
  </roads:Geometry>
  <roads:Street ili:ref="3"></roads:Street>
  <Precision>unprecise</Precision>
</StreetAxis>

<StreetAxis ili:tid="15">
  <roads:Geometry>
    <geom:polyline>
      <geom:coord>
        <geom:c1>55.600</geom:c1><geom:c2>37.649</geom:c2>
      </geom:coord>
      <geom:coord>
        <geom:c1>49.359</geom:c1><geom:c2>56.752</geom:c2>
      </geom:coord>
    </geom:polyline>
  </roads:Geometry>
  <roads:Street ili:ref="4"></roads:Street>
  <Precision>unprecise</Precision>
</StreetAxis>

<!-- === StreetNamePosition / StreetNamePositionAssoc === -->
<roads:StreetNamePosition ili:tid="5">
  <roads:NamPos>
    <geom:coord>
      <geom:c1>71.660</geom:c1><geom:c2>45.231</geom:c2>
    </geom:coord>
  </roads:NamPos>
  <roads:NamOri>15.0</roads:NamOri>
  <roads:Street ili:ref="1"></roads:Street>
</roads:StreetNamePosition>

<roads:StreetNamePosition ili:tid="6">
  <roads:NamPos>
    <geom:coord>
      <geom:c1>58.249</geom:c1><geom:c2>85.081</geom:c2>
    </geom:coord>
  </roads:NamPos>
  <roads:NamOri>351.0</roads:NamOri>
  <roads:Street ili:ref="2"></roads:Street>
</roads:StreetNamePosition>
```

```
        <roads:StreetNamePosition ili:tid="7">
          <roads:NamPos>
            <geom:coord>
              <geom:c1>106.095</geom:c1><geom:c2>33.554</geom:c2>
            </geom:coord>
          </roads:NamPos>
          <roads:NamOri>280.0</roads:NamOri>
          <roads:Street ili:ref="3"></roads:Street>
        </roads:StreetNamePosition>

        <roads:StreetNamePosition ili:tid="14">
          <roads:NamPos>
            <geom:coord>
              <geom:c1>53.031</geom:c1><geom:c2>51.367</geom:c2>
            </geom:coord>
          </roads:NamPos>
          <roads:NamOri>291.3</roads:NamOri>
          <roads:Street ili:ref="4"></roads:Street>
        </roads:StreetNamePosition>

        <!-- === RoadSign === -->
        <RoadSign ili:tid="501">
          <roads:Type>prohibition.noparking</roads:Type>
          <roads:Position>
            <geom:coord>
              <geom:c1>69.389</geom:c1><geom:c2>92.056</geom:c2>
            </geom:coord>
          </roads:Position>
        </RoadSign>

        <RoadSign ili:tid="502">
          <roads:Type>prohibition.noparking</roads:Type>
          <roads:Position>
            <geom:coord>
              <geom:c1>80.608</geom:c1><geom:c2>88.623</geom:c2>
            </geom:coord>
          </roads:Position>
        </RoadSign>

        <RoadSign ili:tid="503">
          <roads:Type>prohibition.noparking</roads:Type>
          <roads:Position>
            <geom:coord>
              <geom:c1>58.059</geom:c1><geom:c2>93.667</geom:c2>
            </geom:coord>
          </roads:Position>
        </RoadSign>

        <RoadSign ili:tid="504">
          <roads:Type>danger</roads:Type>
          <roads:Position>
            <geom:coord>
              <geom:c1>92.741</geom:c1><geom:c2>38.295</geom:c2>
            </geom:coord>
          </roads:Position>
        </RoadSign>
      </RoadsExtended>
      <!-- end of basket REFHANDB00000001 -->
    </ili:datasection>
  </ili:transfer>
```

Graphic description RoadsExgm2ien

With regard to the data model a representation is defined by means of the graphic description RoadsExgm2ien. The graphic model reads as follows:

```
!! File RoadsExgm2ien.ili Release 2014-07-09
```

```
INTERLIS 2.4;

MODEL RoadsExgm2ien (en) AT "http://www.interlis.ch/models/refhb24"
  VERSION "2014-07-09" = !! Roads graphics

  IMPORTS RoadsExdm2ben;
  IMPORTS RoadsExdm2ien;
  IMPORTS StandardSymbology;

  SIGN BASKET StandardSymbology ~ StandardSymbology.StandardSigns
    OBJECTS OF SurfaceSign: Building, Street, Water, Other
    OBJECTS OF PolylineSign: continuous, dotted
    OBJECTS OF TextSign: Linefont_18
    OBJECTS OF SymbolSign: NoParking, GP;


  TOPIC Graphics =
    DEPENDS ON RoadsExdm2ben.Roads, RoadsExdm2ien.RoadsExtended;

    GRAPHIC Surface_Graphics
      BASED ON RoadsExdm2ien.RoadsExtended.LandCover =

      Building OF StandardSymbology.StandardSigns.SurfaceSign:
        WHERE Type == #building (
          Sign := {Building};
          Geometry := Geometry;
          Priority := 100);

      Street OF StandardSymbology.StandardSigns.SurfaceSign:
        WHERE Type == #street (
          Sign := {Street};
          Geometry := Geometry;
          Priority := 100);

      Water OF StandardSymbology.StandardSigns.SurfaceSign:
        WHERE Type == #water (
          Sign := {Water};
          Geometry := Geometry;
          Priority := 100);

      Other OF StandardSymbology.StandardSigns.SurfaceSign:
        WHERE Type == #other (
          Sign := {Other};
          Geometry := Geometry;
          Priority := 100);

    END Surface_Graphics;

    VIEW Surface_Boundary
      INSPECTION OF RoadsExdm2ien.RoadsExtended.LandCover -> Geometry;
      =
    ATTRIBUTE
      ALL OF LandCover;
    END Surface_Boundary;

    VIEW Surface_Boundary2
      INSPECTION OF Base ~ Surface_Boundary -> Lines;
      =
    ATTRIBUTE
      Geometry := Base -> Geometry;
    END Surface_Boundary2;

    GRAPHIC SurfaceBoundary_Graphics
      BASED ON Surface_Boundary2 =

      Boundary OF StandardSymbology.StandardSigns.PolylineSign: (
        Sign := {continuous};
        Geometry := Geometry;
```

```
          Priority := 101);

      END SurfaceBoundary_Graphics;

      GRAPHIC Polyline_Graphics
        BASED ON RoadsExdm2ien.RoadsExtended.StreetAxis =

        Street_precise OF StandardSymbology.StandardSigns.PolylineSign:
          WHERE Precision == #precise (
            Sign := {continuous};
            Geometry := Geometry;
            Priority := 110);

        Street_unprecise OF StandardSymbology.StandardSigns.PolylineSign:
          WHERE Precision == #unprecise (
            Sign := {dotted};
            Geometry := Geometry;
            Priority := 110);

      END Polyline_Graphics;

      GRAPHIC Text_Graphics
        BASED ON RoadsExdm2ien.RoadsExtended.StreetNamePosition =

        StreetName OF StandardSymbology.StandardSigns.TextSign: (
            Sign := {Linefont_18};
            Txt := Street -> Name;
            Geometry := NamPos;
            Rotation := NamOri;
            Priority := 120);

      END Text_Graphics;

      GRAPHIC Point_Graphics
        BASED ON RoadsExdm2ien.RoadsExtended.RoadSign =

        Tree OF StandardSymbology.StandardSigns.SymbolSign:
          WHERE Type == #prohibition.noparking (
            Sign := {NoParking};
            Geometry := Position;
            Priority := 130);

        GP OF StandardSymbology.StandardSigns.SymbolSign:
          WHERE Type == #danger (
            Sign := {GP};
            Geometry := Position;
            Priority := 130);

      END Point_Graphics;

    END Graphics;

  END RoadsExgm2ien.
```

The graphic model RoadsExgm2ien has recourse to the symbols in the symbol library Roads-Exgm2ien_Symbols (file RoadsExgm2ien_Symbols.xtf). A description of the symbol library is to be found in the following paragraph.

Symbol library RoadsExgm2ien_Symbols.xtf

Hereafter the symbol library RoadsExgm2ien_Symbols is represented in the form of an XML-data set (file RoadsExgm2ien_Symbols.xtf). The symbol library contains symbol definitions for control points and trees, as well as line, text and surface symbols. The corresponding symbology model (StandardSymbology) is to be found in appendix L *Symbology models*.

```
<!-- File RoadsExgm2ien_Symbols.xtf 2014-08-05
(http://www.interlis.ch/models/refhb24) -->
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ili:transfer xmlns:ili="http://www.interlis.ch/xtf/2.4/INTERLIS"
  xmlns:geom="http://www.interlis.ch/geometry/1.0"
  xmlns="http://www.interlis.ch/xtf/2.4/StandardSymbology"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ili:headersection>
    <ili:models>
      <ili:model>StandardSymbology</ili:model>
    </ili:models>
    <ili:sender>KOGIS</ili:sender>

    <ili:comment>example symbology dataset ili2 refmanual appendix C</ili:comment>
  </ili:headersection>

  <ili:datasection>
    <StandardSigns ili:bid="REFHANDB00000002">

      <!-- Color Library -->
      <Color ili:tid="1">
        <Name>red</Name>
        <L>40.0</L>
        <C>70.0</C>
        <H>0.0</H>
        <T>1.0</T>
      </Color>

      <Color ili:tid="2">
        <Name>green</Name>
        <L>49.4</L>
        <C>48.5</C>
        <H>153.36</H>
        <T>1.0</T>
      </Color>

      <Color ili:tid="3">
        <Name>light_gray</Name>
        <L>75.0</L>
        <C>0.0</C>
        <H>0.0</H>
        <T>1.0</T>
      </Color>

      <Color ili:tid="4">
        <Name>dark_grey</Name>
        <L>25.0</L>
        <C>0.0</C>
        <H>0.0</H>
        <T>1.0</T>
      </Color>

      <Color ili:tid="5">
        <Name>dark_blue</Name>
        <L>50.3</L>
        <C>43.5</C>
        <H>261.1</H>
        <T>1.0</T>
      </Color>

      <Color ili:tid="6">
        <Name>black</Name>
        <L>0.0</L>
        <C>0.0</C>
        <H>0.0</H>
        <T>1.0</T>
      </Color>

      <Color ili:tid="7">
```

```
    <Name>white</Name>
    <L>100.0</L>
    <C>0.0</C>
    <H>0.0</H>
    <T>1.0</T>
  </Color>

  <PolylineAttrs ili:tid="4001">
    <Width>0.01</Width>
    <Join>round</Join>
    <Caps>butt</Caps>
  </PolylineAttrs>

  <PolylineAttrs ili:tid="4002">
    <Width>0.01</Width>
    <Join>miter</Join>
    <MiterLimit>2.0</MiterLimit>
    <Caps>butt</Caps>
  </PolylineAttrs>

  <!-- Font/Symbol Library -->
  <FontSymbol ili:tid="101">
    <Name>Triangle</Name>
    <Geometry>
      <FontSymbol_Surface>
        <Geometry>
          <geom:surface>
            <geom:exterior>
              <geom:polyline>
                <geom:coord>
                  <geom:c1>-0.5</geom:c1><geom:c2>-0.5</geom:c2>
                </geom:coord>
                <geom:coord>
                  <geom:c1>0.0</geom:c1><geom:c2>0.5</geom:c2>
                </geom:coord>
                <geom:coord>
                  <geom:c1>0.5</geom:c1><geom:c2>-0.5</geom:c2>
                </geom:coord>
                <geom:coord>
                  <geom:c1>-0.5</geom:c1><geom:c2>-0.5</geom:c2>
                </geom:coord>
              </geom:polyline>
            </geom:exterior>
          </geom:surface>
        </Geometry>
      </FontSymbol_Surface>
    </Geometry>
    <Geometry>
      <FontSymbol_Polyline>
        <Color ili:ref="6"></Color>
        <Geometry>
          <geom:polyline>
            <geom:coord>
              <geom:c1>-0.5</geom:c1><geom:c2>0.0</geom:c2>
            </geom:coord>
            <geom:arc>
              <geom:c1>0.5</geom:c1><geom:c2>0.0</geom:c2>
              <geom:a1>0.0</geom:a1><geom:a2>0.5</geom:a2><geom:r>0.5</geom:r>
            </geom:arc>
            <geom:arc>
              <geom:c1>-0.5</geom:c1><geom:c2>0.0</geom:c2>
              <geom:a1>0.0</geom:a1><geom:a2>-0.5</geom:a2><geom:r>0.5</geom:r>
            </geom:arc>
          </geom:polyline>
        </Geometry>
      </FontSymbol_Polyline>
    </Geometry>
    <Font ili:ref="10"></Font>
```

```
</FontSymbol>

<FontSymbol ili:tid="102">
  <Name>NoParking</Name>
  <Geometry>
    <FontSymbol_Polyline>
      <Color ili:ref="6"></Color>
      <Geometry>
        <geom:polyline>
          <geom:coord>
            <geom:c1>-0.5</geom:c1><geom:c2>0.0</geom:c2>
          </geom:coord>
          <geom:arc>
            <geom:c1>0.5</geom:c1><geom:c2>0.0</geom:c2>
            <geom:a1>0.0</geom:a1><geom:a2>0.5</geom:a2><geom:r>0.5</geom:r>
          </geom:arc>
          <geom:arc>
            <geom:c1>-0.5</geom:c1><geom:c2>0.0</geom:c2>
            <geom:a1>0.0</geom:a1><geom:a2>-0.5</geom:a2><geom:r>0.5</geom:r>
          </geom:arc>
        </geom:polyline>
      </Geometry>
    </FontSymbol_Polyline>
  </Geometry>
  <Geometry>
    <FontSymbol_Surface>
      <FillColor ili:ref="1"></FillColor>
      <Geometry>
        <geom:surface>
          <geom:exterior>
            <geom:polyline>
              <geom:coord>
                <geom:c1>-0.233</geom:c1><geom:c2>0.325</geom:c2>
              </geom:coord>
              <geom:arc>
                <geom:c1>0.325</geom:c1><geom:c2>-0.233</geom:c2>
                <geom:a1>0.283</geom:a1>
                <geom:a2>0.283</geom:a2><geom:r>0.4</geom:r>
              </geom:arc>
              <geom:coord>
                <geom:c1>-0.233</geom:c1><geom:c2>0.325</geom:c2>
              </geom:coord>
            </geom:polyline>
          </geom:exterior>
        </geom:surface>
      </Geometry>
    </FontSymbol_Surface>
  </Geometry>
  <Geometry>
    <FontSymbol_Surface>
      <FillColor ili:ref="1"></FillColor>
      <Geometry>
        <geom:surface>
          <geom:exterior>
            <geom:polyline>
              <geom:coord>
                <geom:c1>0.228</geom:c1><geom:c2>-0.324</geom:c2>
              </geom:coord>
              <geom:arc>
                <geom:c1>-0.327</geom:c1><geom:c2>0.238</geom:c2>
                <geom:a1>-0.283</geom:a1>
                <geom:a2>-0.283</geom:a2><geom:r>0.4</geom:r>
              </geom:arc>
              <geom:coord>
                <geom:c1>0.228</geom:c1><geom:c2>-0.324</geom:c2>
              </geom:coord>
            </geom:polyline>
          </geom:exterior>
```

```
        </geom:surface>
      </Geometry>
    </FontSymbol_Surface>
  </Geometry>
  <Geometry>
    <FontSymbol_Surface>
      <FillColor ili:ref="5"></FillColor>
      <Geometry>
        <geom:surface>
          <geom:exterior>
            <geom:polyline>
              <geom:coord>
                <geom:c1>-0.5</geom:c1><geom:c2>0.0</geom:c2>
              </geom:coord>
              <geom:arc>
                <geom:c1>0.5</geom:c1><geom:c2>0.0</geom:c2>
                <geom:a1>0.0</geom:a1>
                <geom:a2>0.5</geom:a2><geom:r>0.5</geom:r>
              </geom:arc>
              <geom:arc>
                <geom:c1>-0.5</geom:c1><geom:c2>0.0</geom:c2>
                <geom:a1>0.0</geom:a1>
                <geom:a2>-0.5</geom:a2><geom:r>0.5</geom:r>
              </geom:arc>
            </geom:polyline>
          </geom:exterior>
          <geom:interior>
            <geom:polyline>
              <geom:coord>
                <geom:c1>-0.233</geom:c1><geom:c2>0.325</geom:c2>
              </geom:coord>
              <geom:arc>
                <geom:c1>0.325</geom:c1><geom:c2>-0.233</geom:c2>
                <geom:a1>0.283</geom:a1>
                <geom:a2>0.283</geom:a2><geom:r>0.4</geom:r>
              </geom:arc>
              <geom:coord>
                <geom:c1>-0.233</geom:c1><geom:c2>0.325</geom:c2>
              </geom:coord>
            </geom:polyline>
          </geom:interior>
          <geom:interior>
            <geom:polyline>
              <geom:coord>
                <geom:c1>0.228</geom:c1><geom:c2>-0.324</geom:c2>
              </geom:coord>
              <geom:arc>
                <geom:c1>-0.327</geom:c1><geom:c2>0.238</geom:c2>
                <geom:a1>-0.283</geom:a1>
                <geom:a2>-0.283</geom:a2><geom:r>0.4</geom:r>
              </geom:arc>
              <geom:coord>
                <geom:c1>0.228</geom:c1><geom:c2>-0.324</geom:c2>
              </geom:coord>
            </geom:polyline>
          </geom:interior>
        </geom:surface>
      </Geometry>
    </FontSymbol_Surface>
  </Geometry>
  <Geometry>
    <FontSymbol_Polyline>
      <Color ili:ref="7"></Color>
      <Geometry>
        <geom:polyline>
          <geom:coord>
            <geom:c1>-0.233</geom:c1><geom:c2>0.325</geom:c2>
          </geom:coord>
```

```
            <geom:arc>
              <geom:c1>0.325</geom:c1><geom:c2>-0.233</geom:c2>
              <geom:a1>0.283</geom:a1>
              <geom:a2>0.283</geom:a2><geom:r>0.4</geom:r>
            </geom:arc>
            <geom:coord>
              <geom:c1>-0.233</geom:c1><geom:c2>0.325</geom:c2>
            </geom:coord>
          </geom:polyline>
        </Geometry>
      </FontSymbol_Polyline>
    </Geometry>
    <Geometry>
      <FontSymbol_Polyline>
        <Color ili:ref="7"></Color>
        <Geometry>
          <geom:polyline>
            <geom:coord>
              <geom:c1>0.228</geom:c1><geom:c2>-0.324</geom:c2>
            </geom:coord>
            <geom:arc>
              <geom:c1>-0.327</geom:c1><geom:c2>0.238</geom:c2>
              <geom:a1>-0.283</geom:a1>
              <geom:a2>-0.283</geom:a2><geom:r>0.4</geom:r>
            </geom:arc>
            <geom:coord>
              <geom:c1>0.228</geom:c1><geom:c2>-0.324</geom:c2>
            </geom:coord>
          </geom:polyline>
        </Geometry>
      </FontSymbol_Polyline>
    </Geometry>
    <Font ili:ref="10"></Font>
  </FontSymbol>

  <!-- Internal Symbol Font "Symbols" -->
  <Font ili:tid="10">
    <Name>Symbols</Name>
    <Internal>true</Internal>
    <Type>symbol</Type>
  </Font>

  <!-- External Text Font "Leroy" -->
  <Font ili:tid="11">
    <Name>Leroy</Name>
    <Internal>false</Internal>
    <Type>text</Type>
    <BottomBase>0.3</BottomBase>
  </Font>

  <!-- Line Styles -->
  <LineStyle_Solid ili:tid="21">
    <Name>LineSolid_01</Name>
    <Color ili:ref="6"></Color>
  </LineStyle_Solid>

  <LineStyle_Dashed ili:tid="22">
    <Name>LineDashed_01</Name>
    <Dashes>
      <DashRec>
        <DLength>0.1</DLength>
      </DashRec>
    </Dashes>
    <Dashes>
      <DashRec>
        <DLength>0.1</DLength>
      </DashRec>
    </Dashes>
```

```
          <Color ili:ref="6"></Color>
        </LineStyle_Dashed>

        <!-- Text Signs -->
        <TextSign ili:tid="1001">
          <ili:Name>Linefont_18</ili:Name>
          <Height>1.8</Height>
          <Font ili:ref="11"></Font>
        </TextSign>

        <!-- Symbol Signs -->
        <SymbolSign ili:tid="2001">
          <ili:Name>GP</ili:Name>
          <Scale>1.0</Scale>
          <Color ili:ref="2"></Color>
          <Symbol ili:ref="101"></Symbol>
        </SymbolSign>

        <SymbolSign ili:tid="2002">
          <ili:Name>NoParking</ili:Name>
          <Scale>1.0</Scale>
          <Symbol ili:ref="102"></Symbol>
        </SymbolSign>

        <!-- Polyline Signs -->
        <PolylineSign ili:tid="3001">
          <ili:Name>continuous</ili:Name>
          <Style ili:ref="21">
            <PolylineSignLineStyleAssoc>
              <Offset>0.0</Offset>
            </PolylineSignLineStyleAssoc>
          </Style>
        </PolylineSign>

        <PolylineSign ili:tid="3002">
          <ili:Name>dotted</ili:Name>
          <Style ili:ref="22">
            <PolylineSignLineStyleAssoc>
              <Offset>0.0</Offset>
            </PolylineSignLineStyleAssoc>
          </Style>
        </PolylineSign>

        <!-- Surface Signs -->
        <SurfaceSign ili:tid="5001">
          <ili:Name>Building</ili:Name>
          <FillColor ili:ref="4"></FillColor>
        </SurfaceSign>

        <SurfaceSign ili:tid="5002">
          <ili:Name>Street</ili:Name>
          <FillColor ili:ref="3"></FillColor>
        </SurfaceSign>

        <SurfaceSign ili:tid="5003">
          <ili:Name>Water</ili:Name>
          <FillColor ili:ref="5"></FillColor>
        </SurfaceSign>

        <SurfaceSign ili:tid="5005">
          <ili:Name>Other</ili:Name>
          <FillColor ili:ref="2"></FillColor>
        </SurfaceSign>
      </StandardSigns>
      <!-- end of basket REFHANDB00000002 -->
    </ili:datasection>
  </ili:transfer>
```

Graphic representation of our example

Combining information provided by the data set RoadsExdm2ien (file RoadsExdm2ien.xtf), the descriptions in the graphic model RoadsExgm2ien (file RoadsExgm2ien.ili) and the symbol library RoadsExgm2ien_Symbols (file RoadsExgm2ien_Symbols.xtf), an INTERLIS 2-graphic processor will generate the following graphic:

Figure 27: Graphic generated from graphic and data descriptions.

Annex F (standard extension suggestion) – Organization of object identifiers (OID)

Preliminary note

The following specification is not a normative component of INTERLIS. This is a standard extension suggestion based upon the INTERLIS Version 2-reference manual in the sense of a recommendation. However we intend to put it up for discussion and possibly convert it into a more definite regulation. Consult the corresponding INTERLIS 2-user manuals for examples of application.

Introduction

Steadily increasing availability of geodata in turn demands its updating and integration in various data-bases. These are some of the reasons why there is a demand for uniform regulations concerning *object identifyers (OID)*: An OID will identify an object instance from its beginning to its end, even if attribute values should alter. In contrast to user keys (cf. appendix G *Uniqueness of user keys* in the INTERLIS 2- Reference Manual) the user has to consider an OID as a non-talking ("opaque") attribute which typically will be administered by system functions.

At least within one transfer community an OID must be *unique, unequivocal* and *unchangeable*.

Amongst others, the following demands are made on the generation and the utilization of OID's:

- The OID is a general and stable identifier, even with extensive quantities of data. As an identifier it is an attribute whose value unequivocally designates an object in its class. Being a general identifier its value not only clearly designates an object within its class but within all classes of a transfer community. Furthermore being a stable identifier it is independent of time, i.e. during the life-cycle of an object it cannot be modified and the OID of any deleted object no longer can be used.
- Independent of hardware and software producers.
- Independent of platforms.
- Serviceable for multiple as well as individual users, resp. in autonomous systems (e.g. in field work).
- Little space required and if need be still further to be optimized.
- Easy to implement.

Other possible demands are not necessarily of technical nature, e.g. a minimum of expenditure in organization, under national control, also utilizable with older systems and approval of system providers. These are high demands which partially point to opposite directions. A special requirement states that an OID can be placed at least 10 million times by a producing system; furthermore that the OID has a set length in order to facilitate its manipulation (thus excluding other well-known procedures, such as a so-called URI as a prefix). There is no call for control numbers, it is assumed that lower communication levels provide the necessary tools.

On principle, the uniqueness of an OID will always be achieved through a central mechanism. The two extremes, i.e. placing of each OID through a central authority on the one hand, and the completely decentral and autonomous generating of OID's on the other, lead to unsatisfying results. An OID places via an MAC address of a network adapter and a time stamp for example is neither deemed practicable nor very promising, since it would mean that each computer be equipped with a MAC and it is not to be foreseen if this technology will not be outdated within the next few years.

There is a long history to the development of this specification. Over the years we have conducted studies, conferences and reviews. Some of the documents established in its course is available to interested parties (place your order with www.interlis.ch, resp. info@interlis.ch).

Structure of an object identifier (OID)

An object identifier (OID) consists of a prefix and a postfix and has got a set length of 16 alpha-numerical characters. An OID is always treated as a unity, above all on the data interface and where the user is concerned. The OID domain STANDARDOID of the INTERLIS model corresponds precisely to this definition. However it only defines its entire length and not its detailed structure.

```
OIDDef = Prefix Postfix.
```

Prefix

A central authority generates the prefix. Thus uniqueness is guaranteed within a transfer community. Typically every basket (i.e. a database process, which administers data of a concrete topic) demands a new prefix. It is the country of the prefix-creating process that is considered to be the destination of the prefix. This process is not automatically in the same place as the producing system that creates the entire OID.

A prefix consists of 8 characters, the following symbols being admissible:

```
Prefix = Letter { Letter | Digit }.  !! sequence of 8 characters
Letter = ( 'A' | .. | 'Z' | 'a' | .. | 'z' ).
Digit = ( '0' | '1' | .. | '9' ).
```

A prefix is defined as a sequence of letters and digits, the first symbol having to be a letter (cf. also structure of XML-tag names or chapter *Names* in the INTERLIS Version 2-Reference Manual).

Moreover, the first two prefix characters have to be determined according to the country codes of ISO-Norm 3166. Thus the letters "ch" have been selected for all prefixes created in Switzerland, "de" for Germany, "at" for Austria etc. Thus for the creation of a prefix 62 different varieties are available per character (0..9: ASCII 48 to 57; A..Z: ASCII 65 to 90; a..z: ASCII 97 to 122). The combining of 62 symbols with the number of 10 characters results in a number that exceeds the probable exigencies of most of the applications at present conceivable.

Postfix

A postfix is created by the data-producer, resp. the producing system itself. It consists of 8 characters, Its ASCII-compatible; column-oriented approach demands that possible "void" characters on the left be filled with noughts ("0") (see example 1 and 3 of an OID below). Thus the smallest possible ordinal value of the postfix part is depicted as "00000000".

```
Postfix = { Letter | Digit }.  !! sequence of 8 characters
```

If need be further restrictions in the prefix or postfix part can be defined in additional specifications.

Summary and application examples

OID	*Length*	*Significance*	*Notes*
Prefix	2 + 6 Char.	Country specification + a unique ‚global' identification-part assigned by a central authority.	Worldwide unequivocal country specification e.g. de (Germany), at (Austria), ch (Switzerland) according to ISO-norm 3166. Further re-

OID	*Lengt h*	*Significance*	*Notes*
			strictions require additional specifi-cations.
Postfix	8 Char.	Sequence (numeric or al-phanumeric) of the produc-ing system as a 'local' identi-fications-part	Further restrictions such as e.g. date stamp with sequence number require additional specifications.

Examples

```
1234567812345678    Comment
----------------    -------

A000000000000000    Theoretically the smallest possible OID

zwzzzzzzzzzzzzzz    The maximum possible OID with zw for Zimbabwe

deg5mQXX2000004a    OID of German origin (de) selected at random

chgAAAAAAAAA0azD    OID of Swiss origin (ch) selected at random
```

Organization

Some authority (possibly federal), universally acknowledged by the transfer community, maintains a central service charged with the generation of OID. Via appropriate communication channels data-producers may obtain one or several prefixes. This might be for example an Internet-page connected with an e-mail service. Such a service can be made relatively secure and safeguarded against abuse.

It is up to the implementation of the source and target system to utilize the characteristics explicitly stated in this specification and to use an OID appropriately, e.g. for sorting or internal optimizing. Administering the prefix part within the system at a central locality may attain such optimizing; furthermore the different objects would only contain the postfix part and in addition relate to a common prefix part. Other economies may result if the postfix part is memorized system internally as a binary number.

For practice exercises use:

– The OID-prefix "chB00000" with OID for baskets.

– The OID-prefix "ch100000" with all other OID.

Annex G (standard extension suggestion) – Uniqueness of user keys

Note

The following specification is not a normative component of INTERLIS. This is a standard extension suggestion based upon the INTERLIS Version 2-Reference Manual in the sense of a recommendation. However we intend to put it up for discussion and possibly convert it into a more definite regulation. Consult the corresponding INTERLIS 2-user manuals for examples of application.

Modeling alternatives

If uniqueness is a requirement in user keys, the question arises within which limits this uniqueness applies. From a purely technical point of view it is often obvious, that uniqueness can only be guaranteed within one specific basket, since all other baskets are not accessible. From a modeling point of view however, a basket is meaningless as long as no statement concerning its extent can be made.

It is doubtful whether uniqueness even is of necessity in a base model. As seen by a superior authority (e.g. Federation) it is quite conceivable that uniqueness is not the rule for all, but only for the internal (federate) data model.

Hereafter we present two possible ways of dealing with the problem of unique user keys:

- Variety central regulation.
- Variety decentralist regulation (delegation principle).

Variant Central regulation

Without further reflection, a central regulation would probably be in the foreground. A central authority determines for all objects of one class that a certain user key has to be unique within the entire area. This may be achieved by taking certain organizational steps, or all parties concerned may have access to a central database.

```
TOPIC Property =

  CLASS Allotment =
    Number: 1 .. 99999;
    Geometry: AREA WITH (STRAIGHTS, ARCS) VERTEX CHCoord
              WITHOUT OVERLAPS > 0.005;
  UNIQUE
    Number;
  END Allotment;

END Property.
```

Often the central authority will determine a tessellation and hence unique area numbers within the entire area. If allotments, which in turn are situated within these areas, should be unique in all respects, then the user key must necessarily consist of a combination of both area number and allotment number:

```
  CLASS Allotment =
    Area_Number: 1 .. 9999;
    Number: 1 .. 99999;
    Geometry: AREA WITH (STRAIGHTS, ARCS) VERTEX CHCoord
              WITHOUT OVERLAPS > 0.005;
  UNIQUE
    Area_Number, Number; !! User key
  END Allotment;
```

Variant Decentralist Regulation (Delegation Principle)

If the necessary data structures have been set in a data model, it is possible to comprehend objects primarily in smaller baskets (e.g. one basket per county), which further along can be collected without any problems in bigger baskets (e.g. one for a whole canton). Supposing furthermore, that the federal authority demands allotment numbers with five digits, without determining the limits where uniqueness is required, and presuming at the same time that a canton requires uniqueness within the limits of one county, then the following modeling is possible:

```
MODEL Federation (en) AT "http://www.interlis.ch/"
  VERSION "2005-06-16" =

  DOMAIN
    CHCoord = COORD
      0.000 .. 200.000 [INTERLIS.m], !! Min_East  Max_East
      0.000 .. 200.000 [INTERLIS.m], !! Min_North Max_North
    ROTATION 2 -> 1;


  TOPIC Property =

    CLASS Allotment =
      Number: 1..99999;
      Geometry: AREA WITH (STRAIGHTS, ARCS) VERTEX CHCoord
                WITHOUT OVERLAPS > 0.005;
    END Allotment;

  END Property;

END Federation.



MODEL CantonA (en) AT "http://www.interlis.ch/"
  VERSION "2005-06-16" =

  IMPORTS Federation;


  TOPIC OrgStructure =

    CLASS County =
      Name: TEXT*30;
    UNIQUE
      Name;
    END County;

  END OrgStructure;


  TOPIC Property EXTENDS Federation.Property =
    DEPENDS ON OrgStruktur;

    ASSOCIATION CountyAllotment =
      County (EXTERNAL) -- {1} CantonA.OrgStruktur.County;
      Allotment -- Allotment;
    END CountyAllotment;

    CONSTRAINTS OF Allotment =
      UNIQUE
        Number, County;
    END;

  END Property;

END CantonA.
```

According to definition, the names of counties must be unique within the scope of all objects of one class. It is irrelevant whether the observance of this requirement can be checked in view of their distribution into concrete baskets. Nevertheless this requirement prevails.

In order to determine uniqueness of the allotment number within a county, a relationship is established between allotment and county and it is required, that the combination of county and number be unique. Again it is irrelevant whether a basket comprises part of a county, a county as a whole or several counties. From the view point of modeling the requirement prevails.

Proceeding on the assumption that a system contains the allotments of a certain county, it is quite possible that system internally the relationship between county and allotments is omitted, only to be enclosed when transferring data to other systems.

Annex H (standard extension suggestion) – Definition of units

Note

The following specification is not a normative component of INTERLIS. This is a standard extension suggestion based upon the INTERLIS Version 2-reference manual in the sense of a recommendation. However we intend to put it up for discussion and possibly convert it into a more definite regulation. Consult the corresponding INTERLIS 2-user manuals for examples of application.

The type model

The following type model comprises the most common units. It extends units that have been directly defined by INTERLIS (cf. appendix A *The internal INTERLIS-data model*).

```
!! File Units.ili Release 2014-07-09

INTERLIS 2.4;

TYPE MODEL Units (en) AT "http://www.interlis.ch/models/refhb24"
  VERSION "2014-07-09" =

  UNIT
    !! abstract Units
    Area (ABSTRACT) = (INTERLIS.LENGTH*INTERLIS.LENGTH);
    Volume (ABSTRACT) = (INTERLIS.LENGTH*INTERLIS.LENGTH*INTERLIS.LENGTH);
    Velocity (ABSTRACT) = (INTERLIS.LENGTH/INTERLIS.TIME);
    Acceleration (ABSTRACT) = (Velocity/INTERLIS.TIME);
    Force (ABSTRACT) = (INTERLIS.MASS*INTERLIS.LENGTH/INTERLIS.TIME/INTERLIS.TIME);
    Pressure (ABSTRACT) = (Force/Area);
    Energy (ABSTRACT) = (Force*INTERLIS.LENGTH);
    Power (ABSTRACT) = (Energy/INTERLIS.TIME);
    Electric_Potential (ABSTRACT) = (Power/INTERLIS.ELECTRIC_CURRENT);
    Frequency (ABSTRACT) = (INTERLIS.DIMENSIONLESS/INTERLIS.TIME);

    Millimeter [mm] = 0.001 [INTERLIS.m];
    Centimeter [cm] = 0.01 [INTERLIS.m];
    Decimeter [dm] = 0.1 [INTERLIS.m];
    Kilometer [km] = 1000 [INTERLIS.m];

    Square_Meter [m2] EXTENDS Area   = (INTERLIS.m*INTERLIS.m);
    Cubic_Meter  [m3] EXTENDS Volume = (INTERLIS.m*INTERLIS.m*INTERLIS.m);

    Minute [min] = 60 [INTERLIS.s];
    Hour   [h]   = 60 [min];
    Day    [d]   = 24 [h];

    Kilometer_per_Hour [kmh] EXTENDS Velocity = (km/h);
    Meter_per_Second [ms] = 3.6 [kmh];
    Newton [N]  EXTENDS Force = (INTERLIS.kg*INTERLIS.m/INTERLIS.s/INTERLIS.s);
    Pascal [Pa] EXTENDS Pressure = (N/m2);
    Joule  [J]  EXTENDS Energy = (N*INTERLIS.m);
    Watt   [W]  EXTENDS Power = (J/INTERLIS.s);
    Volt   [V]  EXTENDS Electric_Potential = (W/INTERLIS.A);

    Inch [in] = 2.54 [cm];
    Foot [ft] = 0.3048 [INTERLIS.m];
    Mile [mi] = 1.609344 [km];

    Are [a] = 100 [m2];
    Hectare [ha] = 100 [a];
    Square_Kilometer [km2] = 100 [ha];
    Acre [acre] = 4046.873 [m2];
```

```
    Liter [L] = 1 / 1000 [m3];
    US_Gallon [USgal] = 3.785412 [L];

    Angle_Degree = 180 / PI [INTERLIS.rad];
    Angle_Minute = 1 / 60 [Angle_Degree];
    Angle_Second = 1 / 60 [Angle_Minute];

    Gon = 200 / PI [INTERLIS.rad];

    Gram [g] = 1 / 1000 [INTERLIS.kg];
    Ton [t] = 1000 [INTERLIS.kg];
    Pound [lb] = 0.4535924 [INTERLIS.kg];

    Calorie [cal] = 4.1868 [J];
    Kilowatt_Hour [kWh] = 0.36E7 [J];

    Horsepower = 746 [W];

    Techn_Atmosphere [at] = 98066.5 [Pa];
    Atmosphere [atm] = 101325 [Pa];
    Bar [bar] = 100000 [Pa];
    Millimeter_Mercury [mmHg] = 133.3224 [Pa];
    Torr = 133.3224 [Pa]; !! Torr = [mmHg]

    Decibel [dB] = FUNCTION // 10**(dB/20) * 0.00002 // [Pa];

    Degree_Celsius [oC] = FUNCTION // oC+273.15 // [INTERLIS.K];
    Degree_Fahrenheit [oF] = FUNCTION // (oF+459.67)/1.8 // [INTERLIS.K];

    CountedObjects EXTENDS INTERLIS.DIMENSIONLESS;

    Hertz [Hz] EXTENDS Frequency = (CountedObjects/INTERLIS.s);
    KiloHertz [KHz] = 1000 [Hz];
    MegaHertz [MHz] = 1000 [KHz];

    Percent = 0.01 [CountedObjects];
    Permille = 0.001 [CountedObjects];

    !! ISO 4217 Currency Abbreviation
    USDollar    [USD] EXTENDS INTERLIS.MONEY;
    Euro        [EUR] EXTENDS INTERLIS.MONEY;
    SwissFrancs [CHF] EXTENDS INTERLIS.MONEY;

END Units.
```

Examples

Cf. chapter *Base units* in the INTERLIS Version 2-Reference Manual.

Annex I (standard extension suggestion) – Time definitions

Note

The following specification is not a normative component of INTERLIS. This is a standard extension suggestion based upon the INTERLIS Version 2-reference manual in the sense of a recommendation. However we intend to put it up for discussion and possibly convert it into a more definite regulation. Consult the corresponding INTERLIS 2-user manuals for examples of application.

The time model

```
!! File Time.ili Release 2014-07-09

INTERLIS 2.4;

REFSYSTEM MODEL Time (en) AT "http://www.interlis.ch/models/refhb24"
  VERSION "2014-07-09" =

  IMPORTS Units;

  STRUCTURE DayOfYear =
    Month: 1 .. 12 [INTERLIS.M];
    SUBDIVISION Day: 1..31 [INTERLIS.d];
  END DayOfYear;

  STRUCTURE HMDiffWithinDay =
    Hours: -23 .. 23 CIRCULAR [INTERLIS.h];
    CONTINUOUS SUBDIVISION Minutes: 0 .. 59 [INTERLIS.min];
  END HMDiffWithinDay;

  DOMAIN
    WeekDay = (WorkingDay (Monday, Tuesday, Wednesday,
                           Thursday, Friday, Saturday),
               Sunday) CIRCULAR;

    HMDiffWDay = FORMAT BASED ON HMDiffWithinDay (Hours ":" Minutes);
    DifferenceToUTC EXTENDS HMDiffWDay = MANDATORY "-13:00" .. "13:00";
                                     !! UTC := LocTime + Diff

  FUNCTION DayInCurrentYear (dayOfYear: MANDATORY DayOfYear;
                             weekDay: WeekDay): DayOfYear
    // returns first parameter if second is undefined,
       returns first day from (incl) first parameter being the
       requested weekday within the current year //;

  FUNCTION DSTOrdered (day1: DayOfYear; day2: DayOfYear) : BOOLEAN
    // returns TRUE if the second parameter comes after the
       first parameter or if both parameters are equal //;

  STRUCTURE DSTransition =
    TransitionDSTime: MANDATORY HMDiffWDay;
    FirstDate: MANDATORY DayOfYear;
    DayOfWeek: WeekDay;
  END DSTransition;

  STRUCTURE DaylightSavingPeriod =
    DSToUTC: DifferenceToUTC;
    From: MANDATORY INTERLIS.GregorianYear;
    To:   MANDATORY INTERLIS.GregorianYear;
    DSStart: MANDATORY DSTransition;
    DSEnd: MANDATORY DSTransition;
  MANDATORY CONSTRAINT
    DSTOrdered (DSStart.FirstDate, DSEnd.FirstDate);
```

```
    MANDATORY CONSTRAINT
      To >= From;
    END DaylightSavingPeriod;

    FUNCTION DSPOverlaps (periods: BAG {1..*} OF DaylightSavingPeriod) : BOOLEAN
      // returns TRUE if any one of the periods overlap //;


    TOPIC TimeZone =

      CLASS TimeZone (ABSTRACT) EXTENDS INTERLIS.SCALSYSTEM =
      PARAMETER
        Unit (EXTENDED): NUMERIC [INTERLIS.TIME];
      END TimeZone;

      CLASS BaseTimeZone EXTENDS INTERLIS.TIMESYSTEMS.TIMEOFDAYSYS =
                                   !! TimeZone without daylight saving
        DiffToUTC: DifferenceToUTC;
      END BaseTimeZone;

      CLASS DaylightSavingTZ EXTENDS INTERLIS.TIMESYSTEMS.TIMEOFDAYSYS =
        Periods: BAG {1..*} OF DaylightSavingPeriod;
      MANDATORY CONSTRAINT
        NOT ( DSPOverlaps (Periods) );
      END DaylightSavingTZ;

      ASSOCIATION DaylightSavingTZOf =
        BaseTZ -<> BaseTimeZone;
        DSTZ -- DaylightSavingTZ;
      END DaylightSavingTZOf;

    END TimeZone;

  END Time.
```

Exemplary data for the time model

The following example corresponds to the time model above.

```
<!-- File SwissTimeData.xtf 2014-08-05 (http://www.interlis.ch/models/refhb24) -->

<?xml version="1.0" encoding="UTF-8"?>
<ili:transfer xmlns:ili="http://www.interlis.ch/xtf/2.4/INTERLIS"
  xmlns:geom="http://www.interlis.ch/geometry/1.0"
  xmlns="http://www.interlis.ch/xtf/2.4/Time"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ili:headersection>
    <ili:models>
      <ili:model>Time</ili:model>
    </ili:models>
    <ili:comment>example dataset ili2 refmanual appendix G</ili:comment>
  </ili:headersection>
  <ili:datasection>
    <TimeZone ili:bid="BTimeZones">
      <BaseTimeZone ili:tid="BTimeZones.MEZ">
        <ili:Name>MEZ</ili:Name>
        <DiffToUTC>-1:00</DiffToUTC>
      </BaseTimeZone>

      <DaylightSavingTZ ili:tid="BTimeZones.MESZ">
        <ili:Name>MESZ</ili:Name>
        <Periods>
          <DaylightSavingPeriod>
            <DSToUTC>-2:00</DSToUTC>
            <From>1983</From>
            <To>1995</To>
            <DSStart>
              <DSTransition>
```

```
              <TransitionDSTime>3:00</TransitionDSTime>
              <FirstDate>
                <DayOfYear>
                  <Month>3</Month>
                  <Day>25</Day>
                </DayOfYear>
              </FirstDate>
              <DayOfWeek>Sunday</DayOfWeek>
            </DSTransition>
          </DSStart>
          <DSEnd>
            <DSTransition>
              <TransitionDSTime>3:00</TransitionDSTime>
              <FirstDate>
                <DayOfYear>
                  <Month>9</Month>
                  <Day>24</Day>
                </DayOfYear>
              </FirstDate>
              <DayOfWeek>Sunday</DayOfWeek>
            </DSTransition>
          </DSEnd>
        </DaylightSavingPeriod>
      </Periods>
      <Periods>
        <DaylightSavingPeriod>
          <DSToUTC>-2:00</DSToUTC>
          <From>1996</From>
          <To>2999</To>
          <DSStart>
            <DSTransition>
              <TransitionDSTime>3:00</TransitionDSTime>
              <FirstDate>
                <DayOfYear>
                  <Month>3</Month>
                  <Day>25</Day>
                </DayOfYear>
              </FirstDate>
              <DayOfWeek>Sunday</DayOfWeek>
            </DSTransition>
          </DSStart>
          <DSEnd>
            <DSTransition>
              <TransitionDSTime>3:00</TransitionDSTime>
              <FirstDate>
                <DayOfYear>
                  <Month>10</Month>
                  <Day>25</Day>
                </DayOfYear>
              </FirstDate>
              <DayOfWeek>Sunday</DayOfWeek>
            </DSTransition>
          </DSEnd>
        </DaylightSavingPeriod>
      </Periods>
    </DaylightSavingTZ>

    <DaylightSavingTZOf>
      <BaseTZ ili:ref="BTimeZones.MEZ"></BaseTZ>
      <DSTZ ili:ref="BTimeZones.MESZ"></DSTZ>
    </DaylightSavingTZOf>
  </TimeZone>
</ili:datasection>
</ili:transfer>
```

Annex J (standard extension suggestion) – Color definitions

Note

The following specification is not a normative component of INTERLIS. This is a standard extension suggestion based upon the INTERLIS Version 2-Reference Manual in the sense of a recommendation. However we intend to put it up for discussion and possibly convert it into a more definite regulation. Consult the corresponding INTERLIS 2-user manuals for examples of application.

Introduction

This specification states in detail why a certain color space named $L^*C^*_{ab}h^*_{ab}$ is best suited for colour definitions. It gives an exhaustive description of this color space, cites conversion formulas related to other color spaces and gives instructions as to how a transformation of $L^*C^*_{ab}h^*_{ab}$-coordinates into the color-coordinate system of a concrete screen or printer may be implemented. Furthermore it lays the foundation for the domains and precisions selected hereafter and indicates coordinates of especially chosen examples.

Since, amongst other faculties, INTERLIS 2 enables the description of graphics, it must be possible to specify colors. However a system and equipment neutral definition of "color" is surprisingly complex and demands comprehension of concepts that are not generally known.

Color is a product of light (= color stimulus), eye (= color valence) and brain function (= sensation). It is virtually impossible to describe colors through numbers in such a way, that two persons will conceive them identically. However color values can be measured in a universally acknowledged way, thus permitting a precise understanding amongst experts.

A method of specifying colors as strings should meet several requirements:

- **Equipment independence** — It ought to be clearly defined which color actually corresponds to a certain indication. This is the only means of ascertaining that the result will fulfill all expectations, whatever equipment is being used.
- **Expressiveness** — It ought to be possible to specify all colors that "normal" equipment (especially also good quality printers and plotters) will be able to represent. The spectrum of colors to be specified should be as wide as possible. Ideally it would comprise all colors a human being can perceive.
- **Intuition** — While reading a color description, a human being should intuitively have a notion of the color being described. An INTERLIS model always has a certain documentary character and should be understandable to those concerned without demanding major efforts.
- **System neutrality** — The ways and methods of indicating color should neither give precedence to a certain system (GIS, operating system, hardware), nor cause the acquisition of special devices.

Color space

The table below shows the suitability of different color spaces as far as application in INTERLIS-graphic descriptions is concerned:

Color	Equipment In-	Expressive-	Intuitive	System Neutral-

space	dependence	ness	Intelligibility	ity
RGB	–	–	–	–
HLS	–	–	+ +	–
HSV	–	–	+ +	–
CMY(K)	–	–	–	–
XYZ	+	+	– –	+
SRGB	+	–	–	–
$L^* a^* b^*$	+	+	+	+
$L^* C^*_{ab} h^*_{ab}$	+	+	+ +	+

Figure H.1: Suitability of different colorspaces for the purposes of INTERLIS.

The latter of the colorspaces mentioned in figure H.1 $L^* C^*_{ab} h^*_{ab}$ (d.h. $L^*a^*b^*$ with polar coordinates) meets the requirements stated above in the most satisfactory manner.

L*a*b*

The colorspace $L^*a^*b^*$ (sometimes also called CIELAB) widely used in the graphic industry, can be derived via transformation from *XYZ* as described in figure H.2.

$$L^* = 116 \cdot f\left(\frac{Y}{Y_n}\right) - 16$$

$$a^* = 500 \cdot \left[f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right) \right] \quad \text{, whereby } f(x) = \begin{cases} \sqrt[3]{x} & \text{if } x > 0.008856; \\ 7.787x + \frac{16}{116} & \text{else} \end{cases}$$

$$b^* = 200 \cdot \left[f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right) \right]$$

Figure H.2: The conversion of XYZ to L*a*b*.

In the calculation in figure H.2 a "reference white" is introduced by means of ‹X_n, Y_n, Z_n› in order to compensate an eventual tinge of light. Very often the values of CIE-standard light sources (mainly D50, occasionally D65) are employed. The XYZ-coordinates of these light sources can be found for example in [Sangwine/Horne, 1989], Table 3.1.

This range possesses a number of useful properties:

- **Equipment independence** — $L^*a^*b^*$ is derived from *XYZ* and hence independent of a certain equipment. It is unequivocally defined which color belongs to a $L^*a^*b^*$-Triple.
- **Expressiveness** — In $L^*a^*b^*$ a point is assigned to each color that can be emitted by a reflecting surface.
- **Intuitive Intelligibility** — L^* means luminance, whereby a completely black surface (which reflects no light at all) possesses an L^* of 0 and a perfect reflector (which reflects all light) an L^* of 100. A human observer will judge a colour with $L^* = 50$ as average brightness. a* is the red-green-axis: colors with $a^* = 0$ will be perceived as neither red nor green, colours with a negative a^* are red, colors with positive a^* are green. Analogously b^* is the blue-yellow-axis.

Within a plane which is spanned with *a** and *b** there is a distance from the zero point to a specific color value, the greater the distance the more saturated a color becomes.

- **System neutrality** — *L*a*b** is absolutely system neutral; being an internationale standard the colorspace is independent of a specific firm.
- **Increasing utilization** — The utilization of *L*a*b** in professional printing is widely spread. Programs such as Adobe Photoshop or Acrobat (PDF) support *L*a*b**.
- **Easy transformability to RGB** — *L*a*b**-triples can be transformed into the RGB-values of any screen via multiplication with a 3x3-matrix, followed by a raise to higher power (gamma correction), which may be carried out efficiently by means of a table (cf. [Adobe, 1992], chapter 23). Thus system developers will only have to face minimal efforts.
- **Good capacity for compression** — There is only a marginal difference between *L*a*b** and RGB where processes are concerned that are likely to involve loss while compressing pictures. However in connection with INTERLIS this is irrelevant.

$$C^*_{ab} = \sqrt{(a^*)^2 + (b^*)^2} \qquad h^*_{ab} = \tan^{-1}\left(\frac{b^*}{a^*}\right)$$

Figure H.3: Conversion of the cartesian L*a*b*-space to the polar form L*C*abh*ab (according to [Sangwine/Horne, 1998]).

$L^*C^*_{ab}h^*_{ab}$

As described above, in the *L*a*b**-space every single axis *L** (dark —light), *a** (green — red) and *b** (blue — yellow) corresponds to a property of colour which is immediately perceivable.

Nevertheless *intuitive intelligibility* can be further increased by indicating color coordinates in a polar instead of a cartesian system (see figure H.4).

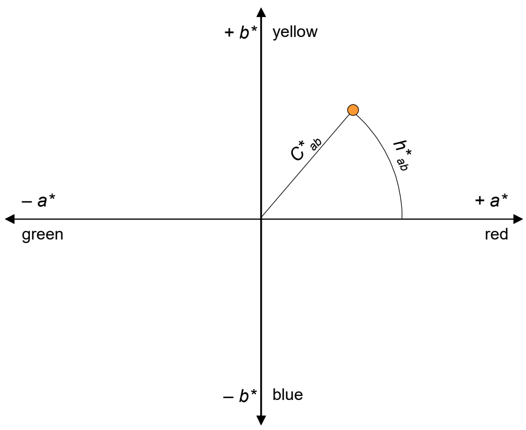


Figure H.4: The colorspace L*C*abh*ab functions with polar coordinates onto L*a*b*.

The formula in figure H.3 for h^*_{ab} is only applicable for positive a* and b*; a correct version would provide case differentiation for every single quadrant. This polar system combines the intuitive intelligibility of HLS and HSV with the numerous advantages of L*a*b* described above, since it means that the axes L* (*luminance*), C* (*chroma*) und h* (*hue*) become separately available.

In INTERLIS models, whenever precise color indications are desired, they should be made based upon this color coordinate system.

Required precision

It is part of an INTERLIS model to indicate the degree of precision to be applied when recording numeric values. The *L*a*b**-space is defined in such a way that the difference between two colors is only just perceptible, if the value calculated as shown in figure 1.5, equals 1.

Note: [Has/Newman, o.D.] states that the perceptibility of color differences also depends on the amount of time allowed for the comparison. The article relates an experiment, where the time needed to note differences was measured in the case of an inexperienced observer. The figures mentioned are 5 seconds for ΔE_{ab} = 15, 10 seconds for ΔE_{ab} = 10 and 15 seconds for ΔE_{ab} = 5.

$$\Delta E_{ab} = \sqrt{\Delta L^{*2} + \Delta a^{*2} + \Delta b^{*2}}$$

Figure H.5: Calculation of color differences in the Cartesian L*a*b*-space.

Precision of the *L**-axis: For luminance a precision of one decimal is sufficient.

Precision of C^*_{ab}- and h^*_{ab}-axis: Theoretically a* and b* may be unrestricted, but in fact limits of ±128, rounded off to whole numbers, are considered largely sufficient (cf. [Adobe, 1992]). Thus what degree of precision is necessary for C^*_{ab} und h^*_{ab} , to ensure that inaccuracy in the a*/b*-surface does not exceed 1?

Inaccuracy introduced through the indication of angle augments with increasing distance from the zero point. Thus precision can still be considered sufficient as long as ‹127, 128› and ‹128, 128› within the a*/b*-surface can be distinguished. As can be seen in figure H.6, one decimal may suffice in this extreme case. It is a matter of two barely distinguishable hues of orange, however saturated to such a degree that it seems improbable any apparatus would be able to reproduce them.

*a**	*b**	C^*_{ab}	h^*_{ab}
127	128	180.3	45.2
128	128	181.0	45.0

Figure H.6: Cartesian and polar coordinates of a color extremely far away from the zero point (conversion see figure H.3).

Combination with names

Color names are easier to handle than color codes (i.e. numbers), however this proves to be a disadvantage, as only a limited number of colors are thus available. In INTERLIS names can be combined with a numeric specification, enabling users to define their own color names and to exchange them among one another by the common means of INTERLIS.

Thanks to this definition it is also possible to employ INTERLIS – if need be – in the documenting and utilization of existing color name systems or color sample catalogues, such as the Pantone- or HKS-System.

This calls for the definition of a meta class (cf. chapter *Meta models and meta objects* in the INTERLIS Version 2-Reference Manual). Its instances, so-called meta objects, are retained in a special transfer-file and are read by the INTERLIS 2-compiler. They are available for INTERLIS data-models and thus can be used in graphic-definitions in order to determine the color of a certain symbol, etc.

Examples of application in INTERLIS models

```
!! Component of the symbology model

SYMBOLOGY MODEL SymbologyExample AT "http://www.interlis.ch/"
  VERSION "2014-07-09" =


  TOPIC Signs =

    CLASS LChColor EXTENDS INTERLIS.METAOBJECT =
      !! Attribute "Name" inherited from INTERLIS.METAOBJECT
      Luminance = MANDATORY 0.0 .. 100.0;
      Chroma = MANDATORY 0.0 .. 181.1;
      Hue = MANDATORY 0.0 .. 359.9 CIRCULAR [DEGREE] COUNTERCLOCKWISE;
    END LChColor;

    ...

    !! Component of the symbol class definition within the symbology model
    CLASS ColoredSymbology EXTENDS SIGN =
      ...
    PARAMETER
      Color: METAOBJECT OF SymbologyExample.LChColor;
    END ColoredSymbology;

    ...

  END Signs;
  ...
```

In a user-defined visualization command (here called SimplePointGr) the color of a user-defined colored symbol might appear as follows (cf. chapter *Grafic description* in the INTERLIS Version 2-Reference Manual):

```
...

MODEL SimpleGrafik AT "http://www.interlis.ch/"
  VERSION "2014-07-09" =

  IMPORTS SymbologyExample, Data;

  SIGN BASKET SimpleSigns ~ SymbologyExample.Signs
    OBJECTS OF Color: Brown
    OBJECTS OF ColoredSymbology: Dot;


  TOPIC ColoredDotGraphic =
    DEPENDS ON Data.Dots;

    GRAPHIC SimpleColoredDotGr BASED ON Data.Dots.Dot =
      Symbol OF SymbologyExample.Signs.ColoredSymbology: (
        Sign := {Dot};
        Pos := Position;
        Color := Brown;
      );
    END SimpleColoredDotGr;

  END ColoredDotGraphic;

END SimpleGraphic.
...
```

We neither give a complete example nor represent the necessary meta table, instead we refer you to the example stated in appendix E *A small example Roads* in the INTERLIS Version 2-Reference Manual.

Example values

Figure H.7 names some colors as well as their coordinates. Since it is uncertain whether this document has been conceived in a system (and most likely also printed) which is able to render colors correctly, we must at this stage do without a colorful representation.

Name	L^*	a^*	b^*	C^*_{ab}	h^*_{ab}
Black	0.0	0	0	0.0	0.0
Dark-grey	25.0	0	0	0.0	0.0
Middle-grey	50.0	0	0	0.0	0.0
Light-grey	75.0	0	0	0.0	0.0
White	100.0	0	0	0.0	0.0
Fuchsia	40.0	70	0	70.0	0.0
Light-blue	80.0	0	-30	30	270.0
Deep-yellow	90.0	0	100	100.0	90.0
Brown	50.0	30	50	58.3	59.0
Lilac	50.0	50	-50	70.7	315.0

Figure H.7: Cartesian and polar coordinates of some colors.

A concrete example of application with color-definitions is to be found in appendix E *A small example Roads*.

Notes for system developers

System developers of INTERLIS-conforming systems have to deal with the arising question, how to transform color values from the independent $L^* C^*_{ab} h^*_{ab}$-system into a color-coordinate-system of a specific screen or printer.

A standardized file-format will allow you to record color-distortions of a certain imaging component in so-called *component* or *color matching profiles* (so-called ICC-profile format). Amongst others, these files contain parameters needed in the conversion of an independent color space to a component-specific color-coordinate-system. The former are either *XYZ* or *L*a*b**, the latter commonly *RGB or CMYK*. Format and necessary conversion functions are defined by [ICC, 1996].

Thus in his product a system developer will be able to support directly ICC-profiles. The file format is of a relatively simple structure, and the conversion functions will be easily implemented. For some platforms ready-made program libraries (such as *Apple ColorSync* or *Kodak KCMS*) are available.

In this context we would like to draw your attention to the fact that PDF directly supports the colorspace *L*a*b**. PostScript even allows you to define your own color ranges in terms of any given transformation from *XYZ*. The inversion function of the formula indicated in figure H.2 is to be found as example 4.11 in [Adobe, 1990]. It is relatively simple to program an analogous function in PostScript which will directly accept $L^*\ C^*_{ab}\ h^*_{ab}$.

Literature

[Adobe, 1990] Adobe Systems: PostScript Language Reference Manual. 2nd Ed., 1990. ISBN 0-201-18127-4. 764 Seiten.

*The reference manual for PostScript, also provides recommendations for the treatment of colors and different conversion methods available in PostScript. Example 4.11 on page 191 defines the L*a*b*-colour range in PostScript.*

[Adobe, 1992] Adobe Developers Association: TIFF Revision 6.0. http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf

*Chapter 23 defines a variety of the TIFF-Format for pictures in the L*a*b*-color range and names a number of advantages in comparison with RGB. Furthermore you will find the outlines of a fast method for the conversion of L*a*b* to RGB.*

[Apple, 2005] Apple Computer, Inc.: Introduction to Color and Color Management Systems. In: *Inside Macintosh — Managing Color with ColorSync.* https://developer.apple.com/library/mac/documentation/GraphicsImaging/Conceptual/csintro/csintro_intro/csintro_intro.html

Easy to understand introduction into different color spaces, with illustrated graphic.

[Apple, 2005] Apple Computer, Inc.: A Brief Overview Of Color. https://developer.apple.com/library/mac/documentation/GraphicsImaging/Conceptual/csintro/csintro_ovrvw/csintro_ovrvw.html

Concise, easy to understand and rough introduction into different concepts in connection with colors. Intended for non-specialists.

[Has/Newman, o.D.] Michael Has, Todd Newman: Color Management: Current Practice and the Adoption of a New Standard. www.color.org/wpaper1.html

Names data for the red-, green- and blue reference point of two typical computer monitors and depicts that they differ widely from the xy-values of NTSC standard-phosphor-colours often quoted. Indicates a transformation from XYZ to RGB-space of a certain screen.

[ICC, 1996] International Color Consortium: ICC Profile Format Specification. www.color.org/profile.html

Defines a file-format which allows the characterization of any given component in respect to its colour representation. Appendix A comments on various colorspaces.

[Poynton, 1997] Charles A. Poynton: Frequently Asked Questions about Color. www.poynton.com/ColorFAQ.html

Explains in paragraph 36, why HLS and HSV are not suitable for the specification of colours.

[Sangwine/Horne, 1998] Sangwine, Stephen J. und Horne, Robin E. N. [Hrsg.]: The Colour Imaging Processing Handbook. Chapman & Hall: London […], 1998. ISBN 0-412-80620-7. 440 Seiten.

Well-founded introduction into the scientific fundamentals of color perception and its application in image processing.

[Stokes et al., 1996] Michael Stokes, Matthew Anderson, Srinivasan Chandrasekar und Ricardo Motta: A Standard Default Color Space for the Internet – sRGB. November 1996. www.color.org/sRGB.html

Specification of sRGB.

Annex K (standard extension suggestion) – Coordinate systems and coordinate reference systems

Note

The following specification is not a normative component of INTERLIS. This is a standard extension suggestion based upon the INTERLIS Version 2-Reference Manual in the sense of a recommendation. However we intend to put it up for discussion and possibly convert it into a more definite regulation. Consult the corresponding INTERLIS 2-user manuals for examples of application.

Introduction

Coordinates describe the position of one point in space, provided a corresponding coordinate system has been set up. If a coordinate system is fixedly positioned in relation to the earth – in other terms: referenced – then it is called a coordinate reference system. However coordinates not only determine positions, but also metric quantities which can be derived from coordinates, such as distances, surfaces, volumes, angles and directions, as well as other properties, e.g. grades and curves.

There is a multitude of classes (types) of coordinate systems, and a greater number still of objects, i.e. realizations (instances) of coordinate systems (cf. also e.g. [Voser1999]). The Swiss Federal coordinates e.g. rely on a special instance (object) of a coordinate reference system [Gubler et al. 1996], which can be derived from a geodetical reference system via map projection [Snyder 1987, Bugayevskiy 1995]. These geodetical reference systems form a category of its own of coordinate reference systems that describe the geometry of the earth model. For example to describe a two-dimensional position, a sphere or an ellipsoid is used on whose surface geographical coordinates can be defined. It is slightly more complicated as soon as altitude is concerned: To serve as geometrical-physical earth model we employ either a geoid [Marti 1997] or a gravity model [Torge 1975] that defines orthometrical, resp. normal altitudes. However in practice it is very often only heights in use that are applied.

Since geodata of geomatical applications always are space-related, each geodata-set must be based upon a coordinate system. Considering that individual coordinate systems differ widely, it is necessary to supply the corresponding reference-data along with the geodata. This is why INTERLIS enables you to describe data belonging to a coordinate system.

It is only through knowledge of the underlying coordinate system that it is possible to transfer geodata into another coordinate system. This again is necessary if geodata provided by different coordinate systems is to be of common use [Voser 1996].

First we consider coordinate systems of a general type, then the relations (representations) between (general) coordinate systems, thereafter we introduce coordinate reference systems and deal with those.

Coordinate systems

A *coordinate system* allows the "measuring" of metric space. A coordinate system possesses an origin, coordinate axis (their number corresponding to the dimension of the space spanned), as well as measure units assigned to the axis. Depending on whether the space in question is one-, two- or three-dimensional, the coordinate system assigns either a single digit, double digit or triple digit to every point in space as its coordinate(s).

The euclidic one-, two- or three-dimensional space is defined by its 1, 2 resp. 3 straight axis. Curved spaces demand additional parameters to define the embedding of their curved axis into a euclidic space. For geodetical purposes, two-dimensional elliptic spaces as well as various systems of heights, both treated as special cases of one-dimensional Euclidic spaces are needed in addition to Euclidic spaces of different dimensions. This is when a gravity model or geoid is called for.

Slightly differing from the hitherto existing usage in geodesy we employ the term of *geodetical date* as a synonym for *geodatical reference system, designating* thus nothing but a special coordinate system, that is to say a 3D Cartesian coordinate system which has been positioned in relation to the earth. This may be achieved in two different ways:

(a) The average center of gravity of the earth is defined as the zero point of the coordinate system, the first axis through the average rotation axis of the earth, the second axis perpendicular to the former through the average meridian of Greenwich, and the third axis again perpendicular to the former two, thus creating a clockwise-rotating system. For example the coordinate system WGS84 is defined in this manner.

(b) The surface of the earth of a certain area (mostly a country) is approximated in an optimal way by means of a globe or rotation ellipse whose axis of rotation is set parallel to the average rotation axis of the earth. This rotation ellipsoid defines a Cartesian coordinate system through its smaller half axis which is parallel to the earth axis, through one of its longer half axis and through a third axis perpendicular to the former two, thus again creating a clockwise-rotating system.

A 3D Cartesian coordinate system positioned on the earth in accordance to (a) or (b) is called a *geodetical date* or *geodetical reference system.*

Different origin of coordinate systems in geomatics

Different backgrounds lead to various definitions of coordinate systems in:

Sensor techniques: The data capturing methods in classical geodesy (e.g. with theodolites) as well as photogrammetry and remote sensing use a (local) coordinate system in accordance with each respective method in their data capturing sensors.

Geopositioning: The description of position on the earth by means of a (geodatical) earth model. There are three different types of geodatical earth models [Voser 1999]:

- *physical:* The earth model is either described by means of a gravitational field or a geoid.
- *mathematical:* The earth model is a symmetrical body (e.g. a globe or ellipsoid).
- *topographical:* The earth model also takes into consideration mountains and valleys (earth surface model).

The above-mentioned earth models correspond to different coordinate systems.

***Map positioning**:* Since the surfaces of the above-mentioned earth models are of curved or even more complex form, the calculation of distances, angles etc. is very difficult. Hence we employ map projections that represent the two-dimensional surface in a plane. A map projection is a geometrically clearly defined way of representing the surface of a mathematical earth model in a plane. This process involves distortions; these however can be determined and controlled in advance.

Mappings between coordinate systems

Since geodata usually are recorded in different coordinate systems or are administered by different institutions in various systems, it is necessary to know the methods that permit conversion of data supplied by a source coordinate system A into a target coordinate system Z. This conversion is called mapping of coordinate system A, resp. of the space defined by A, in coordinate system Z, resp. the space defined by Z. Mappings between two coordinate systems, resp. between the spaces they define, are determined by the classes of the two coordinate systems concerned.

We have to distinguish between two fundamentally different mappings of coordinate systems as far as the origin of formulas and their parameters are concerned, these being conversion and transformation.

The term *conversion* means mapping between two coordinate systems strictly defined by formulas and their parameters. These formulas and especially the values of the necessary parameters are determined in advance [Voser 1999]. Into the category of conversions fall amongst others map projections, i.e. mappings of ellipsoid surfaces in a plane, furthermore the conversion of elliptic coordinates into the corresponding Cartesian coordinates with their origin in the center of the ellipse or vice-versa.

A *transformation* is a mapping between two coordinate systems where rules (formulas) are determined based upon hypotheses, and parameters are established by means of a statistic analysis of measurements in both coordinate systems [Voser 1999]. Typical transformations are effected when replacing one geodatical earth model with another (geodatical date transformation) or when adjusting local coordinates in a superior system, e.g. With digitizing: the transfomation of map- or table-coordinates into projection-coordinates.

Coordinate reference systems

The term Coordinate Reference System describes a coordinate system, which can be derived from a geodetical reference system (i.e. a geodetical date) by means of conversion via a sequence of intermediate coordinate systems.

Geodetical reference systems (or geodetical dates) as such are the most important coordinate reference systems. They refer to a geodetical earth model (see above).

Survey of the most important coordinate reference systems

In the lower part of figure I.1 some of the most important geodetical and cartographical expressions of coordinate reference systems are depicted. It is the earth itself that is at the origin of any sequence of coordinate systems or mappings. We try to assign it a geometrical earth model that would allow the describing of positions on it. To begin with we can assign to the earth as a whole a 3D Cartesian coordinate system with its zero point in the gravity center of the earth (cf. method a) in the chapter Coordinate systems above). Subsequently however we treat the position and height of a point independently. Firstly let us consider only what needs to be done in order to determine its position. Geodetical measurements supply the necessary information to determine the size and form of a rotation ellipsoid that approaches the earth surface locally in an optimal way. According to method b) in chapter Coordinate systems this rotation ellipsoid can be assigned a geodetical date. Many of the earth models selected for national surveying are "locally" referenced, i.e. the center of the ellipsoid does not coincide with the gravity center of the earth. However, as stated above ((a) in the chapter Coordinate systems), there are geodetical reference systems which are referenced to the gravitation center. Thus nowadays it is relatively easy to determine the parameters of a date transformation to such a superior system. Once such a local rotation ellipsoid has been decided upon, it is on the other hand possible to represent its surface in a plane by means of an appropriate map projection and in accordance with all requirements.

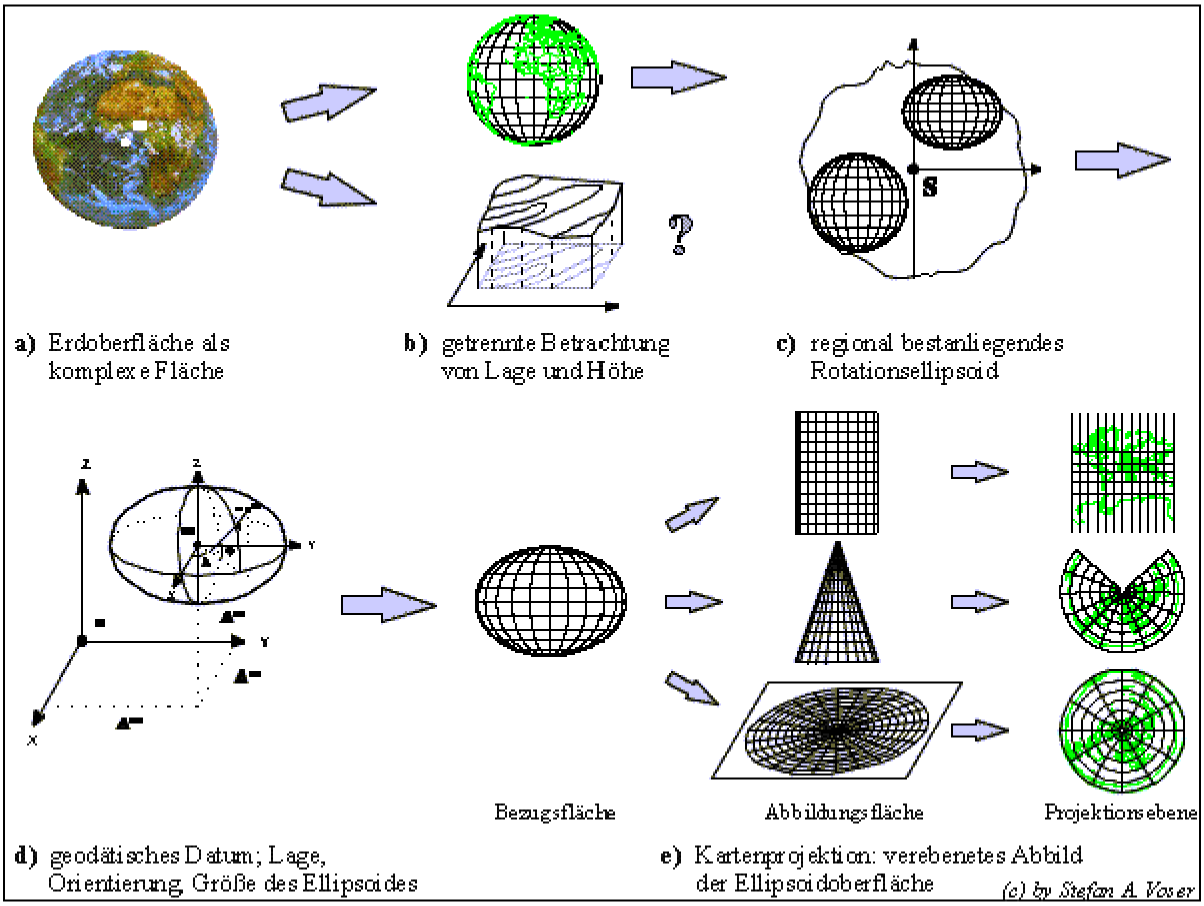


Figure I.1: How to transform the earth surface into 2D horizontal coordinates.

Data structure for coordinate systems and mappings between them

The proposed structure for data necessary for the description of coordinate systems and mapping between them is *not* limited to coordinate reference systems, but on purpose has been conceived for coordinate systems in general. It is our intention to also permit the description of digitizer- and screen coordinate systems or symbol coordinate systems without any explicit reference to the surface of the earth.

Coordinate systems and mapping between them are the two key-concepts for the exact characterization of spatial referencing of geodata. Accordingly the conceptual model (resp. the conceptual schema) of the data structure features two major groups of classes, these being "Coordinate systems for geodetic purposes" and "Mappings between coordinate systems". The third dimension, height, is treated as follows: In a 3D Cartesian coordinate system the height has been implicitly integrated as the third coordinate. However in daily use coordinate systems usually are a combination of a 2D horizontal coordinate systems and an additional 1D height system. The (meta) data of coordinate systems of this type are described by two independent data sets, firstly by the data of a 2D coordinate system (2D Cartesian or elliptic) and secondly by the data of a height system of appropriate type (normal, orthometric or elliptic).

How do the proposed data structures help to effect mapping between coordinate systems? In the following way: Coordinate systems form nodes and mappings between them constitute edges in a graph structure. In the DOMAIN section of an application model (application schema) the name of the coordinate system in use is to be found. If the given coordinates are to be mapped into another coordinate system or for example if GeoTIFF-parameters, which correspond to such a mapping are to be calculated, then an appropriate program within the graph-structure of coordinate systems and mappings has to find the shortest possible way from the node of the given coordinate system (according to DOMAIN) to the node of the target system. Thereafter the necessary mappings from the source system via possible intermediate coordinate systems to the target system have to be calculated.

For the description of coordinate systems two internal classes and key words are available in INTERLIS, these being: AXIS and COORDSYSTEM. These are employed within the conceptual data model (the coordinate system model or coordinate system schema) "CoordSys" (see below). Further details are to be found in chapter *Reference systems* in the INTERLIS Version 2-Reference Manual.

Literature

[Bugayevskiy 1995] Bugayevskiy Lev M., Snyder John P.: Map Projections, A Reference Manual, Taylor&Francis, London, Bristol 1995.

[Gubler et al. 1996] Gubler E., Gutknecht D., Marti U., Schneider D. Signer Th., Voge B., Wiget A.: Die neue Landesvermessung der Schweiz LV95; VPK 2/96.

[Marti 1997] Marti, Urs: Geoid der Schweiz 1997. Geodätisch-geophysikalische Arbeiten in der Schweiz, Schweizerische Geodätische Kommission, Volume 56, Zürich, 1997.

[Torge 1975] Torge, Wolfgang: Geodäsie. Sammlung Göschen 2163, de Gruyter, Berlin – New York, 1975.

[Snyder 1987] Snyder, John P.: Map Projections - A Working Manual, U.S. Geological Survey Professional Paper 1395, Washington, 1987.

[Voser 1996] Voser, Stefan A; Anforderungen an die Geometrie zur gemeinsamen Nutzung unterschiedlicher Datenquellen; 4. deutsche Arc/Info-Anwender-Konferenz, Proceedings, März 1996, Freising.

[Voser 1999] Voser, Stefan. A.: MapRef - The Internet Collection of Map Projections and Reference Systems for Europe; 14. ESRI European User Conference, Presentation and Proceedings; 15.-17. Nov. 1999; Munich: www.mapref.org/.

The reference system model

Data structure for coordinate systems and coordinate reference systems as well as mapping between them. Conceptual data model (conceptual schema) with INTERLIS.

```
!! File CoordSys.ili Release 2014-07-09

INTERLIS 2.4;

REFSYSTEM MODEL CoordSys (en) AT "http://www.interlis.ch/models/refhb24"
  VERSION "2014-07-09" =

  UNIT
    Angle_Degree = 180 / PI [INTERLIS.rad];
    Angle_Minute = 1 / 60 [Angle_Degree];
    Angle_Second = 1 / 60 [Angle_Minute];

  STRUCTURE Angle_DMS_S =
    Degrees: -180 .. 180 CIRCULAR [Angle_Degree];
    CONTINUOUS SUBDIVISION Minutes: 0 .. 59 CIRCULAR [Angle_Minute];
    CONTINUOUS SUBDIVISION Seconds: 0.000 .. 59.999 CIRCULAR [Angle_Second];
  END Angle_DMS_S;

  DOMAIN
    Angle_DMS = FORMAT BASED ON Angle_DMS_S (Degrees ":" Minutes ":" Seconds);
    Angle_DMS_90 EXTENDS Angle_DMS = "-90:00:00.000" .. "90:00:00.000";


  TOPIC CoordsysTopic =

    !! Special space aspects to be referenced
    !! **************************************

    CLASS Ellipsoid EXTENDS INTERLIS.REFSYSTEM =
      EllipsoidAlias: TEXT*70;
      SemiMajorAxis: MANDATORY 6360000.0000 .. 6390000.0000 [INTERLIS.m];
      InverseFlattening: MANDATORY 0.00000000 .. 350.00000000;
      !! The inverse flattening 0 characterizes the 2-dim sphere
      Remarks: TEXT*70;
    END Ellipsoid;

    CLASS GravityModel EXTENDS INTERLIS.REFSYSTEM =
      GravityModAlias: TEXT*70;
      Definition: TEXT*70;
    END GravityModel;

    CLASS GeoidModel EXTENDS INTERLIS.REFSYSTEM =
      GeoidModAlias: TEXT*70;
      Definition: TEXT*70;
    END GeoidModel;


    !! Coordinate systems for geodetic purposes
    !! ****************************************

    STRUCTURE LengthAXIS EXTENDS INTERLIS.AXIS =
      ShortName: TEXT*12;
      Description: TEXT*255;
    PARAMETER
      Unit (EXTENDED): NUMERIC [INTERLIS.LENGTH];
    END LengthAXIS;

    STRUCTURE AngleAXIS EXTENDS INTERLIS.AXIS =
      ShortName: TEXT*12;
      Description: TEXT*255;
    PARAMETER
      Unit (EXTENDED): NUMERIC [INTERLIS.ANGLE];
    END AngleAXIS;
```

```
CLASS GeoCartesian1D EXTENDS INTERLIS.COORDSYSTEM =
  Axis (EXTENDED): LIST {1} OF LengthAXIS;
END GeoCartesian1D;

CLASS GeoHeight EXTENDS GeoCartesian1D =
  System: MANDATORY (
    normal,
    orthometric,
    ellipsoidal,
    other);
  ReferenceHeight: MANDATORY -10000.000 .. +10000.000 [INTERLIS.m];
  ReferenceHeightDescr: TEXT*70;
END GeoHeight;

ASSOCIATION HeightEllips =
  GeoHeightRef -- {*} GeoHeight;
  EllipsoidRef -- {1} Ellipsoid;
END HeightEllips;

ASSOCIATION HeightGravit =
  GeoHeightRef -- {*} GeoHeight;
  GravityRef -- {1} GravityModel;
END HeightGravit;

ASSOCIATION HeightGeoid =
  GeoHeightRef -- {*} GeoHeight;
  GeoidRef -- {1} GeoidModel;
END HeightGeoid;

CLASS GeoCartesian2D EXTENDS INTERLIS.COORDSYSTEM =
  Definition: TEXT*70;
  Axis (EXTENDED): LIST {2} OF LengthAXIS;
END GeoCartesian2D;

CLASS GeoCartesian3D EXTENDS INTERLIS.COORDSYSTEM =
  Definition: TEXT*70;
  Axis (EXTENDED): LIST {3} OF LengthAXIS;
END GeoCartesian3D;

CLASS GeoEllipsoidal EXTENDS INTERLIS.COORDSYSTEM =
  Definition: TEXT*70;
  Axis (EXTENDED): LIST {2} OF AngleAXIS;
END GeoEllipsoidal;

ASSOCIATION EllCSEllips =
  GeoEllipsoidalRef -- {*} GeoEllipsoidal;
  EllipsoidRef -- {1} Ellipsoid;
END EllCSEllips;


!! Mappings between coordinate systems
!! ***********************************

ASSOCIATION ToGeoEllipsoidal =
  From -- {0..*} GeoCartesian3D;
  To -- {0..*} GeoEllipsoidal;
  ToHeight -- {0..*} GeoHeight;
MANDATORY CONSTRAINT
  ToHeight -> System == #ellipsoidal;
MANDATORY CONSTRAINT
  To -> EllipsoidRef -> Name == ToHeight -> EllipsoidRef -> Name;
END ToGeoEllipsoidal;

ASSOCIATION ToGeoCartesian3D =
  From2 -- {0..*} GeoEllipsoidal;
  FromHeight-- {0..*} GeoHeight;
  To3 -- {0..*} GeoCartesian3D;
```

```
MANDATORY CONSTRAINT
  FromHeight -> System == #ellipsoidal;
MANDATORY CONSTRAINT
  From2 -> EllipsoidRef -> Name == FromHeight -> EllipsoidRef -> Name;
END ToGeoCartesian3D;

ASSOCIATION BidirectGeoCartesian2D =
  From -- {0..*} GeoCartesian2D;
  To -- {0..*} GeoCartesian2D;
END BidirectGeoCartesian2D;

ASSOCIATION BidirectGeoCartesian3D =
  From -- {0..*} GeoCartesian3D;
  To2 -- {0..*} GeoCartesian3D;
  Precision: MANDATORY (
    exact,
    measure_based);
  ShiftAxis1: MANDATORY -10000.000 .. 10000.000 [INTERLIS.m];
  ShiftAxis2: MANDATORY -10000.000 .. 10000.000 [INTERLIS.m];
  ShiftAxis3: MANDATORY -10000.000 .. 10000.000 [INTERLIS.m];
  RotationAxis1: Angle_DMS_90;
  RotationAxis2: Angle_DMS_90;
  RotationAxis3: Angle_DMS_90;
  NewScale: 0.000001 .. 1000000.000000;
END BidirectGeoCartesian3D;

ASSOCIATION BidirectGeoEllipsoidal =
  From4 -- {0..*} GeoEllipsoidal;
  To4 -- {0..*} GeoEllipsoidal;
END BidirectGeoEllipsoidal;

ASSOCIATION MapProjection (ABSTRACT) =
  From5 -- {0..*} GeoEllipsoidal;
  To5 -- {0..*} GeoCartesian2D;
  FromCo1_FundPt: MANDATORY Angle_DMS_90;
  FromCo2_FundPt: MANDATORY Angle_DMS_90;
  ToCoord1_FundPt: MANDATORY -10000000 .. +10000000 [INTERLIS.m];
  ToCoord2_FundPt: MANDATORY -10000000 .. +10000000 [INTERLIS.m];
END MapProjection;

ASSOCIATION TransverseMercator EXTENDS MapProjection =
END TransverseMercator;

ASSOCIATION SwissProjection EXTENDS MapProjection =
  IntermFundP1: MANDATORY Angle_DMS_90;
  IntermFundP2: MANDATORY Angle_DMS_90;
END SwissProjection;

ASSOCIATION Mercator EXTENDS MapProjection =
END Mercator;

ASSOCIATION ObliqueMercator EXTENDS MapProjection =
END ObliqueMercator;

ASSOCIATION Lambert EXTENDS MapProjection =
END Lambert;

ASSOCIATION Polyconic EXTENDS MapProjection =
END Polyconic;

ASSOCIATION Albus EXTENDS MapProjection =
END Albus;

ASSOCIATION Azimutal EXTENDS MapProjection =
END Azimutal;

ASSOCIATION Stereographic EXTENDS MapProjection =
END Stereographic;
```

```
    ASSOCIATION HeightConversion =
      FromHeight -- {0..*} GeoHeight;
      ToHeight -- {0..*} GeoHeight;
      Definition: TEXT*70;
    END HeightConversion;

  END CoordsysTopic;

END CoordSys.
```

The file MiniCoordSysData, whose names might occur in MetadataBasketDef, contains the following data in the INTERLIS 2-transfer format.

```
<!-- File MiniCoordSysData.xtf 2014-08-05 (http://www.interlis.ch/models/refhb24) -
->

<?xml version="1.0" encoding="UTF-8"?>
<ili:transfer xmlns:ili="http://www.interlis.ch/xtf/2.4/INTERLIS"
  xmlns:geom="http://www.interlis.ch/geometry/1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.interlis.ch/xtf/2.4/CoordSys">
  <ili:headersection>
    <ili:models>
      <ili:model>CoordSys</ili:model>
    </ili:models>
    <ili:sender>KOGIS</ili:sender>

    <ili:comment>example dataset ili2 refmanual appendix K</ili:comment>

  </ili:headersection>

  <ili:datasection>
    <CoordsysTopic ili:bid="BCoordSys">
      <Ellipsoid ili:tid="BCoordSys.Bessel">
        <ili:Name>Bessel</ili:Name>
        <EllipsoidAlias>Bessel (1841)</EllipsoidAlias>
        <SemiMajorAxis>6377397.1550</SemiMajorAxis>
        <InverseFlattening>299.1528128</InverseFlattening>
        <Remarks>Documentation swisstopo 19031266</Remarks>
      </Ellipsoid>

      <Ellipsoid ili:tid="BCoordSys.ETRS">
        <ili:Name>ETRS</ili:Name>
        <EllipsoidAlias>ETRS 89</EllipsoidAlias>
        <SemiMajorAxis>6378137.000</SemiMajorAxis>
        <InverseFlattening>298.2572235</InverseFlattening>
        <Remarks>EUREF documentation</Remarks>
      </Ellipsoid>

      <GravityModel ili:tid="BCoordSys.SwissGravityNetwork2004">
        <ili:Name>SwissGravityNetwork2004</ili:Name>
        <Definition>See documentation swisstopo Landesschwerenetz</Definition>
      </GravityModel>

      <GravityModel ili:tid="BCoordSys.UEGN2002">
        <ili:Name>UEGN2002</ili:Name>
        <Definition>EGG07 documentation IAG Symposium Series Vol 133</Definition>
      </GravityModel>

      <GeoidModel ili:tid="BCoordSys.CHGeoid2004">
        <ili:Name>CHGeoid2004</ili:Name>
        <Definition>See new Swiss Geoid swisstopo</Definition>
      </GeoidModel>

      <GeoidModel ili:tid="BCoordSys.EGG">
        <ili:Name>EGG</ili:Name>
        <Definition>EGG07 documentation IAG Symposium Series Vol 133</Definition>
```

```
</GeoidModel>

<GeoHeight ili:tid="BCoordSys.SwissOrthometricAlt">
  <ili:Name>SwissOrthometricAlt</ili:Name>
  <ili:Axis>
    <LengthAXIS>
      <ShortName>h</ShortName>
      <Description>Swiss Orthometric Altitude</Description>
    </LengthAXIS>
  </ili:Axis>
  <System>orthometric</System>
  <ReferenceHeight>373.600</ReferenceHeight>
  <ReferenceHeightDescr>Pierre du Niton</ReferenceHeightDescr>
  <EllipsoidRef ili:ref="BCoordSys.Bessel"/>
  <GeoidRef ili:ref="BCoordSys.CHGeoid2004"/>
  <GravityRef ili:ref="BCoordSys.SwissGravityNetwork2004"/>
</GeoHeight>

<GeoHeight ili:tid="BCoordSys.SwissEllipsoidalAlt">
  <ili:Name>SwissEllipsoidalAlt</ili:Name>
  <ili:Axis>
    <LengthAXIS>
      <ShortName>H</ShortName>
      <Description>Swiss Ellipsoidal Altitude</Description>
    </LengthAXIS>
  </ili:Axis>
  <System>ellipsoidal</System>
  <ReferenceHeight>0.000</ReferenceHeight>
  <ReferenceHeightDescr>Sea level</ReferenceHeightDescr>
  <EllipsoidRef ili:ref="BCoordSys.Bessel"/>
  <GeoidRef ili:ref="BCoordSys.CHGeoid2004"/>
  <GravityRef ili:ref="BCoordSys.SwissGravityNetwork2004"/>
</GeoHeight>

<GeoHeight ili:tid="BCoordSys.SwissUsualAlt">
  <ili:Name>SwissUsualAlt</ili:Name>
  <ili:Axis>
    <LengthAXIS>
      <ShortName>H</ShortName>
      <Description>Swiss Usual Altitude</Description>
    </LengthAXIS>
  </ili:Axis>
  <System>other</System>
  <ReferenceHeight>373.600</ReferenceHeight>
  <ReferenceHeightDescr>Pierre du Niton</ReferenceHeightDescr>
  <EllipsoidRef ili:ref="BCoordSys.Bessel"/>
  <GeoidRef ili:ref="BCoordSys.CHGeoid2004"/>
  <GravityRef ili:ref="BCoordSys.SwissGravityNetwork2004"/>
</GeoHeight>

<GeoHeight ili:="BCoordSys.EuropeanNormalAlt">
  <ili:Name>EuropeanNormalAlt</ili:Name>
  <ili:Axis>
    <LengthAXIS>
      <ShortName>H</ShortName>
      <Description>European Normal Altitude</Description>
    </LengthAXIS>
  </ili:Axis>
  <System>normal</System>
  <ReferenceHeight>0</ReferenceHeight>
  <ReferenceHeightDescr>Normaal Amsterdam Peil</ReferenceHeightDescr>
  <EllipsoidRef ili:ref="BCoordSys.ETRS"/>
  <GeoidRef ili:ref="BCoordSys.EGG"/>
  <GravityRef ili:ref="BCoordSys.UEGN2002"/>
</GeoHeight>

<GeoCartesian2D ili:tid="BCoordSys.COORD2">
  <ili:Name>COORD2</ili:Name>
```

```
      <ili:Axis>
        <LengthAXIS>
          <ShortName>X</ShortName>
          <Description>X-axis</Description>
        </LengthAXIS>
      </ili:Axis>
      <ili:Axis>
      <LengthAXIS>
          <ShortName>Y</ShortName>
          <Description>Y-axis</Description>
        </LengthAXIS>
      </ili:Axis>
      <Definition>Mathematical Cartesian 2D Refsystem</Definition>
    </GeoCartesian2D>

    <GeoCartesian2D ili:tid="BCoordSys.CHLV03">
      <ili:Name>CHLV03</ili:Name>
      <ili:Axis>
        <LengthAXIS>
          <ShortName>Y</ShortName>
          <Description>East-value</Description>
        </LengthAXIS>
      </ili:Axis>
      <ili:Axis>
        <LengthAXIS>
          <ShortName>X</ShortName>
          <Description>North-value</Description>
        </LengthAXIS>
      </ili:Axis>
      <Definition>Swiss Geodetic Cartesian 2D Refsystem 1903</Definition>
    </GeoCartesian2D>

    <GeoCartesian2D ili:tid="BCoordSys.CHLV95">
      <ili:Name>CHLV95</ili:Name>
      <ili:Axis>
        <LengthAXIS>
          <ShortName>Y</ShortName>
          <Description>East-value</Description>
        </LengthAXIS>
      </ili:Axis>
      <ili:Axis>
        <LengthAXIS>
          <ShortName>X</ShortName>
          <Description>North-value</Description>
        </LengthAXIS>
      </ili:Axis>
      <Definition>Swiss Geodetic Cartesian 2D Refsystem 1995</Definition>
    </GeoCartesian2D>

    <GeoCartesian3D ili:tid="BCoordSys.COORD3">
      <ili:Name>COORD3</ili:Name>
      <ili:Axis>
        <LengthAXIS>
          <ShortName>X</ShortName>
          <Description>X-axis</Description>
        </LengthAXIS>
      </ili:Axis>
      <ili:Axis>
        <LengthAXIS>
          <ShortName>Y</ShortName>
          <Description>Y-axis</Description>
        </LengthAXIS>
      </ili:Axis>
      <ili:Axis>
        <LengthAXIS>
          <ShortName>Z</ShortName>
          <Description>Z-axis</Description>
        </LengthAXIS>
```

```
    </ili:Axis>
    <Definition>Mathematical Cartesian 3D Refsystem</Definition>
  </GeoCartesian3D>

  <GeoCartesian3D ili:tid="BCoordSys.CH1903">
    <ili:Name>CH1903</ili:Name>
    <ili:Axis>
      <LengthAXIS>
        <ShortName>XC</ShortName>
        <Description>Equator Greenwich</Description>
      </LengthAXIS>
    </ili:Axis>
    <ili:Axis>
      <LengthAXIS>
        <ShortName>YC</ShortName>
        <Description>Equator East</Description>
      </LengthAXIS>
    </ili:Axis>
    <ili:Axis>
      <LengthAXIS>
        <ShortName>ZC</ShortName>
        <Description>North</Description>
      </LengthAXIS>
    </ili:Axis>
    <Definition>Swiss Geodetic Cartesian 3D Refsystem 1903</Definition>
  </GeoCartesian3D>

  <GeoCartesian3D ili:tid="BCoordSys.CH1903+">
    <ili:Name>CH1903+</ili:Name>
    <ili:Axis>
      <LengthAXIS>
        <ShortName>XC</ShortName>
        <Description>Equator Greenwich</Description>
      </LengthAXIS>
    </ili:Axis>
    <ili:Axis>
      <LengthAXIS>
        <ShortName>YC</ShortName>
        <Description>Equator East</Description>
      </LengthAXIS>
    </ili:Axis>
    <ili:Axis>
      <LengthAXIS>
        <ShortName>ZC</ShortName>
        <Description>North</Description>
      </LengthAXIS>
    </ili:Axis>
    <Definition>Swiss Geodetic Cartesian 3D Refsystem 1995</Definition>
  </GeoCartesian3D>

  <GeoCartesian3D ili:tid="BCoordSys.WGS84">
    <ili:Name>WGS84</ili:Name>
    <ili:Axis>
      <LengthAXIS>
        <ShortName>XW</ShortName>
        <Description>Equator Greenwich</Description>
      </LengthAXIS>
    </ili:Axis>
    <ili:Axis>
      <LengthAXIS>
        <ShortName>YW</ShortName>
        <Description>Equator East</Description>
      </LengthAXIS>
    </ili:Axis>
    <ili:Axis>
      <LengthAXIS>
        <ShortName>ZW</ShortName>
        <Description>North</Description>
```

```
    </LengthAXIS>
  </ili:Axis>
  <Definition>World Geodetic System 1984</Definition>
</GeoCartesian3D>

<GeoCartesian3D ili:tid="BCoordSys.ETRS89">
  <ili:Name>ETRS89</ili:Name>
  <ili:Axis>
    <LengthAXIS>
      <ShortName>XW</ShortName>
      <Description>Equator Greenwich</Description>
    </LengthAXIS>
  </ili:Axis>
  <ili:Axis>
    <LengthAXIS>
      <ShortName>YW</ShortName>
      <Description>Equator East</Description>
    </LengthAXIS>
  </ili:Axis>
  <ili:Axis>
    <LengthAXIS>
      <ShortName>ZW</ShortName>
      <Description>North</Description>
    </LengthAXIS>
  </ili:Axis>
  <Definition>European Terrestrial Reference System 1989</Definition>
</GeoCartesian3D>

<GeoEllipsoidal ili:tid="BCoordSys.Switzerland">
  <ili:Name>Switzerland</ili:Name>
  <ili:Axis>
    <AngleAXIS>
      <ShortName>Lat</ShortName>
      <Description>Latitude</Description>
    </AngleAXIS>
  </ili:Axis>
  <ili:Axis>
    <AngleAXIS>
      <ShortName>Long</ShortName>
      <Description>Longitude</Description>
    </AngleAXIS>
  </ili:Axis>
  <Definition>Coordinates on the Swiss Ellipsoid 1903</Definition>
  <EllipsoidRef ili:ref="BCoordSys.Bessel"/>
</GeoEllipsoidal>

<ToGeoEllipsoidal>
  <From ili:ref="BCoordSys.CH1903"></From>
  <To ili:ref="BCoordSys.Switzerland"></To>
  <ToHeight ili:ref="BCoordSys.SwissEllipsoidalAlt"></ToHeight>
</ToGeoEllipsoidal>

<ToGeoCartesian3D>
  <From2 ili:ref="BCoordSys.Switzerland"></From2>
  <FromHeight ili:ref="BCoordSys.SwissEllipsoidalAlt"></FromHeight>
  <To3 ili:ref="BCoordSys.CH1903"></To3>
</ToGeoCartesian3D>

<BidirectGeoCartesian3D>
  <From ili:ref="BCoordSys.CH1903"></From>
  <To2 ili:ref="BCoordSys.WGS84"></To2>
  <Precision>measure_based</Precision>
  <ShiftAxis1>674.374</ShiftAxis1>
  <ShiftAxis2>15.056</ShiftAxis2>
  <ShiftAxis3>405.346</ShiftAxis3>
  <RotationAxis1>0:0:0</RotationAxis1>
  <RotationAxis2>0:0:0</RotationAxis2>
  <RotationAxis3>0:0:0</RotationAxis3>
```

```
    <NewScale>1.00</NewScale>
  </BidirectGeoCartesian3D>

  <BidirectGeoCartesian3D>
    <From ili:ref="BCoordSys.WGS84"></From>
    <To2 ili:ref="BCoordSys.CH1903"></To2>
    <Precision>measure_based</Precision>
    <ShiftAxis1>-674.374</ShiftAxis1>
    <ShiftAxis2>-15.056</ShiftAxis2>
    <ShiftAxis3>-405.346</ShiftAxis3>
    <RotationAxis1>-0:0:0</RotationAxis1>
    <RotationAxis2>-0:0:0</RotationAxis2>
    <RotationAxis3>-0:0:0</RotationAxis3>
    <NewScale>1.0</NewScale>
  </BidirectGeoCartesian3D>

  <BidirectGeoCartesian3D>
    <From ili:ref="BCoordSys.CH1903+"></From>
    <To2 ili:ref="BCoordSys.WGS84"></To2>
    <Precision>measure_based</Precision>
    <ShiftAxis1>674.374</ShiftAxis1>
    <ShiftAxis2>15.056</ShiftAxis2>
    <ShiftAxis3>405.346</ShiftAxis3>
    <RotationAxis1>0:0:0</RotationAxis1>
    <RotationAxis2>0:0:0</RotationAxis2>
    <RotationAxis3>0:0:0</RotationAxis3>
    <NewScale>1.00</NewScale>
  </BidirectGeoCartesian3D>

  <BidirectGeoCartesian3D>
    <From ili:ref="BCoordSys.WGS84"></From>
    <To2 ili:ref="BCoordSys.CH1903+"></To2>
    <Precision>measure_based</Precision>
    <ShiftAxis1>-674.374</ShiftAxis1>
    <ShiftAxis2>-15.056</ShiftAxis2>
    <ShiftAxis3>-405.346</ShiftAxis3>
    <RotationAxis1>-0:0:0</RotationAxis1>
    <RotationAxis2>-0:0:0</RotationAxis2>
    <RotationAxis3>-0:0:0</RotationAxis3>
    <NewScale>1.0</NewScale>
  </BidirectGeoCartesian3D>

  <BidirectGeoCartesian3D>
    <From ili:ref="BCoordSys.CH1903"></From>
    <To2 ili:ref="BCoordSys.ETRS89"></To2>
    <Precision>measure_based</Precision>
    <ShiftAxis1>674.374</ShiftAxis1>
    <ShiftAxis2>15.056</ShiftAxis2>
    <ShiftAxis3>405.346</ShiftAxis3>
    <RotationAxis1>0:0:0</RotationAxis1>
    <RotationAxis2>0:0:0</RotationAxis2>
    <RotationAxis3>0:0:0</RotationAxis3>
    <NewScale>1.00</NewScale>
  </BidirectGeoCartesian3D>

  <BidirectGeoCartesian3D>
    <From ili:ref="BCoordSys.ETRS89"></From>
    <To2 ili:ref="BCoordSys.CH1903"></To2>
    <Precision>measure_based</Precision>
    <ShiftAxis1>-674.374</ShiftAxis1>
    <ShiftAxis2>-15.056</ShiftAxis2>
    <ShiftAxis3>-405.346</ShiftAxis3>
    <RotationAxis1>-0:0:0</RotationAxis1>
    <RotationAxis2>-0:0:0</RotationAxis2>
    <RotationAxis3>-0:0:0</RotationAxis3>
    <NewScale>1.0</NewScale>
  </BidirectGeoCartesian3D>
```

```
        <BidirectGeoCartesian3D>
          <From ili:ref="BCoordSys.CH1903+"></From>
          <To2 ili:ref="BCoordSys.ETRS89"></To2>
          <Precision>measure_based</Precision>
          <ShiftAxis1>674.374</ShiftAxis1>
          <ShiftAxis2>15.056</ShiftAxis2>
          <ShiftAxis3>405.346</ShiftAxis3>
          <RotationAxis1>0:0:0</RotationAxis1>
          <RotationAxis2>0:0:0</RotationAxis2>
          <RotationAxis3>0:0:0</RotationAxis3>
          <NewScale>1.00</NewScale>
        </BidirectGeoCartesian3D>

        <BidirectGeoCartesian3D>
          <From ili:ref="BCoordSys.ETRS89"></From>
          <To2 ili:ref="BCoordSys.CH1903+"></To2>
          <Precision>measure_based</Precision>
          <ShiftAxis1>-674.374</ShiftAxis1>
          <ShiftAxis2>-15.056</ShiftAxis2>
          <ShiftAxis3>-405.346</ShiftAxis3>
          <RotationAxis1>-0:0:0</RotationAxis1>
          <RotationAxis2>-0:0:0</RotationAxis2>
          <RotationAxis3>-0:0:0</RotationAxis3>
          <NewScale>1.0</NewScale>
        </BidirectGeoCartesian3D>

        <TransverseMercator>
          <From5 ili:ref="BCoordSys.Switzerland"></From5>
          <To5 ili:ref="BCoordSys.CHLV03"></To5>
          <FromCo1_FundPt>46:57:08.66</FromCo1_FundPt>
          <FromCo2_FundPt>7:26:22.50</FromCo2_FundPt>
          <ToCoord1_FundPt>600000</ToCoord1_FundPt>
          <ToCoord2_FundPt>200000</ToCoord2_FundPt>
        </TransverseMercator>

        <TransverseMercator>
          <From5 ili:ref="BCoordSys.Switzerland"></From5>
          <To5 ili:ref="BCoordSys.CHLV95"></To5>
          <FromCo1_FundPt>46:57:08.66</FromCo1_FundPt>
          <FromCo2_FundPt>7:26:22.50</FromCo2_FundPt>
          <ToCoord1_FundPt>2600000</ToCoord1_FundPt>
          <ToCoord2_FundPt>1200000</ToCoord2_FundPt>
        </TransverseMercator>

        <HeightConversion>
          <FromHeight ili:ref="BCoordSys.SwissEllipsoidalAlt"></FromHeight>
          <ToHeight ili:ref="BCoordSys.SwissOrthometricAlt"></ToHeight>
        </HeightConversion>

        <HeightConversion>
          <FromHeight ili:ref="BCoordSys.SwissOrthometricAlt"></FromHeight>
          <ToHeight ili:ref="BCoordSys.SwissEllipsoidalAlt"></ToHeight>
        </HeightConversion>
      </CoordsysTopic>
    </ili:datasection>
  </ili:transfer>
```

Example

What information within an application model (resp. application schema) is needed in order to identify unequivocally the coordinate system employed, resp. the coordinate reference system?

```
MODEL Example (en) AT "http://www.interlis.ch/"
  VERSION "2014-08-05" =

  IMPORTS CoordSys;
```

```
    REFSYSTEM BASKET BCoordSys ~ CoordSys.CoordsysTopic
      OBJECTS OF GeoCartesian2D: CHLV03
      OBJECTS OF GeoHeight: SwissNormalAlt;

    DOMAIN
      LCoord = COORD
        480000.000 .. 850000.000 [INTERLIS.m] {CHLV03[1]},
         60000.000 .. 320000.000 [INTERLIS.m] {CHLV03[2]},
        ROTATION 2 -> 1;
      Height = COORD
        -200.000 .. 5000.000 [INTERLIS.m] {SwissNormalAlt[1]};
      HCoord = COORD
        480000.000 .. 850000.000 [INTERLIS.m] {CHLV03[1]},
         60000.000 .. 320000.000 [INTERLIS.m] {CHLV03[2]},
          -200.000 ..    5000.000 [INTERLIS.m] {SwissNormalAlt[1]},
        ROTATION 2 -> 1;


    TOPIC T =

      CLASS ControlPoint =
        Name: TEXT*20;
        Position: LCoord;
      END ControlPoint;

    END T;

  END Example.
```

Annex L (standard extension suggestion) – Symbology models

Note

The following specification is not a normative component of INTERLIS. This is a standard extension suggestion based upon the INTERLIS Version 2-Reference Manual in the sense of a recommendation. However we intend to put it up for discussion and possibly convert it into a more definite regulation. Consult the corresponding INTERLIS 2-user manuals for examples of application.

Abstract symbology model

Description of the abstract symbology model.

```
!! File AbstractSymbology.ili Release 2014-07-09

INTERLIS 2.4;

SYMBOLOGY MODEL AbstractSymbology (en)
  AT "http://www.interlis.ch/models/refhb24"
  VERSION "2014-07-09" =

  UNIT
    Millimeter [mm] = 0.001 [INTERLIS.m];
    Angle_Degree = 180 / PI [INTERLIS.rad];

  DOMAIN
    Style_COORD2 (ABSTRACT)    = COORD NUMERIC, NUMERIC;
    Style_COORD3 (ABSTRACT)    = COORD NUMERIC, NUMERIC, NUMERIC;
    Style_POLYLINE (ABSTRACT) = POLYLINE WITH (STRAIGHTS, ARCS)
                                 VERTEX Style_COORD2;  !! {Planar}?
    Style_SURFACE (ABSTRACT)  = SURFACE WITH (STRAIGHTS, ARCS)
                                 VERTEX Style_COORD2;
    Style_INT (ABSTRACT)       = NUMERIC; !! [Units?]
    Style_FLOAT (ABSTRACT)     = NUMERIC; !! [Units?]
    Style_ANGLE (ABSTRACT)     = 0.000 .. 359.999 CIRCULAR [Angle_Degree]
                                 COUNTERCLOCKWISE;  !! RefSystem?


  TOPIC Signs =

    !! Graphic interface

    CLASS TextSign (ABSTRACT) EXTENDS INTERLIS.SIGN =
    PARAMETER
      Txt      : MANDATORY TEXT;
      Geometry : MANDATORY Style_COORD2;
      Rotation : Style_ANGLE; !! Default 0.0
      HAli     : HALIGNMENT;  !! Default Center
      VAli     : VALIGNMENT;  !! Default Half
    END TextSign;

    CLASS SymbolSign (ABSTRACT) EXTENDS INTERLIS.SIGN =
    PARAMETER
      Geometry : MANDATORY Style_COORD2;
      Scale    : Style_FLOAT;
      Rotation : Style_ANGLE; !! Default 0.0
    END SymbolSign;

    CLASS PolylineSign (ABSTRACT) EXTENDS INTERLIS.SIGN =
    PARAMETER
      Geometry : MANDATORY Style_POLYLINE;
    END PolylineSign;
```

```
    CLASS SurfaceSign (ABSTRACT) EXTENDS INTERLIS.SIGN =
    PARAMETER
      Geometry : MANDATORY Style_SURFACE;
    END SurfaceSign;

  END Signs;

END AbstractSymbology.
```

Standard symbology model

Description of the extended standard symbology model built upon its abstract version.

```
!! File StandardSymbology.ili Release 2016-01-21

INTERLIS 2.4;

SYMBOLOGY MODEL StandardSymbology (en)
  AT "http://www.interlis.ch/models/refhb24"
  VERSION "2016-01-21" =

  !! Extended symbology model with symbol libraries and priorities.

  IMPORTS AbstractSymbology;

  UNIT
    Angle_Degree = 180 / PI [INTERLIS.rad];

  DOMAIN
    SS_Priority = 0 .. 9999;
    SS_Float    = -2000000000.000 .. 2000000000.000;
    SS_Angle    = 0.000 .. 359.999
                  CIRCULAR [Angle_Degree] COUNTERCLOCKWISE;
    SS_Coord2   = COORD -2000000000.000 .. 2000000000.000 [INTERLIS.m],
                        -2000000000.000 .. 2000000000.000 [INTERLIS.m],
                  ROTATION 2 -> 1;
    SS_Polyline = POLYLINE WITH (STRAIGHTS, ARCS)
                  VERTEX SS_Coord2;
    SS_Surface  = SURFACE WITH (STRAIGHTS, ARCS)
                  VERTEX SS_Coord2 WITHOUT OVERLAPS > 0.001;


  TOPIC StandardSigns EXTENDS AbstractSymbology.Signs =

    !! StandardSigns contains symbol libraries and symbol interfaces.
    !! The libraries (colors, fonts/symbols and line patterns) form the
    !! base for the construction of concrete symbols. The symbol interfaces
    !! extend the symbol interfaces of AbstractSymbology by priorites.

    !! Library section
    !! +++++++++++++++

    !! Color library
    !! =============
    !! Colors are defined by LCh values with transparency.

    CLASS Color =
      Name: TEXT*40; !! name of color, i.e. "light green"
      L: MANDATORY 0.0 .. 100.0; !! Luminance
      C: MANDATORY 0.0 .. 181.1; !! Chroma
      H: MANDATORY 0.0 .. 359.9 CIRCULAR [Angle_Degree] COUNTERCLOCKWISE; !! Hue
      T: MANDATORY 0.000 .. 1.000; !! Transparency: 0=totally transparent, 1=opaque
    END Color;

    !! Polyline attributes
    !! +++++++++++++++++++
    !! Presentation parameters for simple continuous lines. Polyline attributes
    !! are used by all other polyline definitions (see below).
```

```
CLASS PolylineAttrs =
  Width        : SS_Float;
  Join         : ( !! connection form for line segments
    bevel,
    round,
    miter
  );
  MiterLimit   : 1.0 .. 1000.0; !! only for Join = miter
  Caps         : (  !! termination form at end of line
    round,
    butt
  );
END PolylineAttrs;

!! Font- and symbol library
!! ========================
!! Symbols are a collection of lines and surfaces. Symbols are
!! organized in fonts. A font can be either a text font or a symbol
!! font. If the font is a text font (Type = #text), every symbol
!! (Character) has an UCS4 code (Unicode) and a spacing parameter assigned.

STRUCTURE FontSymbol_Geometry (ABSTRACT) =
  !! Basic structure for uniform treatment of all symbol geometries.
END FontSymbol_Geometry;

STRUCTURE FontSymbol_Polyline EXTENDS FontSymbol_Geometry =
  Color        : REFERENCE TO Color; !! only for symbols
  LineAttrs    : REFERENCE TO PolylineAttrs;
  Geometry     : MANDATORY SS_Polyline;
END FontSymbol_Polyline;

STRUCTURE FontSymbol_Surface EXTENDS FontSymbol_Geometry =
  FillColor    : REFERENCE TO Color; !! only for symbols
  Geometry     : MANDATORY SS_Surface;
  !! Remark: Has no line symbology, because the boundary is *not* part
  !! of the surface. With FillColor you define only the color of the
  !! surface filling.
END FontSymbol_Surface;

CLASS FontSymbol =
  !! All font symbols are defined for size 1.0 and scale 1.0.
  !! The value is measured in user units (i.e. normally [m]).
  Name         : TEXT*40; !! Symbol name, if known
  UCS4         : 0 .. 4000000000; !! only for text symbols (characters)
  Spacing      : SS_Float; !! only for text symbols (characters)
  Geometry     : LIST OF FontSymbol_Geometry
                 RESTRICTION (FontSymbol_Polyline; FontSymbol_Surface);
END FontSymbol;

CLASS Font =
  Name         : MANDATORY TEXT*40; !! Font name or name of external font
  Internal     : MANDATORY BOOLEAN; !! Internal or external font
                                    !! Only for internal fonts the geometric
                                    !! definitions of the symbols is contained
                                    !! in FontSymbol.
  Type         : MANDATORY (
    symbol,
    text
  );
  BottomBase   : SS_Float; !! Only for text fonts, measured relative to text
                           !! height 1.0
END Font;

ASSOCIATION FontAssoc =
  Font -<#> {1} Font;
  Symbol -- {0..*} FontSymbol;
END FontAssoc;
```

```
!! Line symbology library
!! ======================
!! With the line sybology library the user can define continuous, dashed or
!! patterned lines. It is also possible to define multi line symbologies.
!! Each line in a multi line symbology can be continuous, dashed or patterned
!! for itself. The offset indicates the distance from the middle axis. All
!! are stored in the library relative to the width 1.0. The width can be over
!! written by the symbology parameter Width in the symbology interface. For
!! continuous lines the Width parameter defines the total width of the line,
!! for multi lines the parameter Width scales the attribute value offset.

CLASS LineStyle (ABSTRACT) =
  Name          : MANDATORY TEXT*40;
END LineStyle;

CLASS LineStyle_Solid EXTENDS LineStyle =
END LineStyle_Solid;

ASSOCIATION LineStyle_SolidColorAssoc =
  Color -- {0..1} Color;
  LineStyle -- {0..*} LineStyle_Solid;
END LineStyle_SolidColorAssoc;

ASSOCIATION LineStyle_SolidPolylineAttrsAssoc =
  LineAttrs -- {0..1} PolylineAttrs;
  LineStyle -- {0..*} LineStyle_Solid;
END LineStyle_SolidPolylineAttrsAssoc;

STRUCTURE DashRec =
  DLength       : SS_Float; !! Length of dash
END DashRec;

CLASS LineStyle_Dashed EXTENDS LineStyle =
  Dashes        : LIST OF DashRec; !! 1. dash is continuous
                                   !! 2. dash is not visible
                                   !! 3. dash is continuous
                                   !! etc.
END LineStyle_Dashed;

ASSOCIATION LineStyle_DashedColorAssoc =
  Color -- {0..1} Color;
  LineStyle_Dashed -- {0..*} LineStyle_Dashed;
END LineStyle_DashedColorAssoc;

ASSOCIATION LineStyle_DashedLineAttrsAssoc =
  LineAttrs -- {0..1} PolylineAttrs;
  LineStyle_Dashed -- {0..*} LineStyle_Dashed;
END LineStyle_DashedLineAttrsAssoc;

STRUCTURE Pattern_Symbol =
  FontSymbRef : MANDATORY REFERENCE TO FontSymbol;
  ColorRef      : REFERENCE TO Color;
  Weight        : SS_Float; !! Width for symbol lines
  Scale         : SS_Float; !! Default: 1.0
  Dist          : MANDATORY SS_Float; !! Distance along polyline
  Offset        : MANDATORY SS_Float; !! Vertical distance to polyline axis
END Pattern_Symbol;

CLASS LineStyle_Pattern EXTENDS LineStyle =
  PLength       : MANDATORY SS_Float;
  Symbols       : LIST OF Pattern_Symbol;
  !! after PLength the pattern is repeated
END LineStyle_Pattern;

!! Symbology interface
!! +++++++++++++++++++
```

```
!! Text interface
!! ==============

CLASS TextSign (EXTENDED) =
  Height      : MANDATORY SS_Float;
  Weight      : SS_Float; !! line width for line fonts
  Slanted     : BOOLEAN;
  Underlined  : BOOLEAN;
  Striked     : BOOLEAN;
  ClipBox     : SS_Float; !! Defines a rectangular surface around the text
                          !! with distance ClipBox from text.
PARAMETER
  Priority    : MANDATORY SS_Priority;
END TextSign;

ASSOCIATION TextSignFontAssoc =
  Font -<#> {1} Font;
  TextSign -- {0..*} TextSign;
MANDATORY CONSTRAINT
  Font -> Type == #text;
END TextSignFontAssoc;

ASSOCIATION TextSignColorAssoc =
  Color -- {0..1} Color;
  TextSign -- {0..*} TextSign;
END TextSignColorAssoc;

ASSOCIATION TextSignClipFontAssoc =
  ClipFont -- {0..1} Font;
  TextSign2 -- {0..*} TextSign;
END TextSignClipFontAssoc;

!! Symbol interface
!! ================

CLASS SymbolSign (EXTENDED) =
  Scale       : SS_Float;
  Rotation    : SS_Angle;
PARAMETER
  Priority    : MANDATORY SS_Priority;
END SymbolSign;

ASSOCIATION SymbolSignSymbolAssoc =
  Symbol -- {1} FontSymbol;
  SymbolSign -- {0..*} SymbolSign;
END SymbolSignSymbolAssoc;

ASSOCIATION SymbolSignClipSymbolAssoc =
  ClipSymbol -- {0..1} FontSymbol;
  SymbolSign2 -- {0..*} SymbolSign;
END SymbolSignClipSymbolAssoc;

ASSOCIATION SymbolSignColorAssoc =
  Color -- {0..1} Color;
  SymbolSign -- {0..*} SymbolSign;
END SymbolSignColorAssoc;

!! Polyline interface
!! ==================

CLASS PolylineSign (EXTENDED) =
  !! The parameter Width of the interface influences the width *and*
  !! the scale of start- and endsymbols.
PARAMETER
  Priority    : MANDATORY SS_Priority;
  Width       : SS_Float; !! Width of line symbology, default = 1.0
END PolylineSign;
```

```
    ASSOCIATION PolylineSignLineStyleAssoc =
      Style -- {1} LineStyle;
      PolylineSign -- {0..*} PolylineSign;
    ATTRIBUTE
      Offset      : SS_Float; !! Default 0.0
    END PolylineSignLineStyleAssoc;

    ASSOCIATION PolylineSignColorAssoc =
      Color -- {0..1} Color;
      PolylineSign -- {0..*} PolylineSign;
    END PolylineSignColorAssoc;

    ASSOCIATION PolylineSignClipStyleAssoc =
      ClipStyle -- {0..1} LineStyle; !! Used as a mask for clipping
      PolylineSign2 -- {0..*} PolylineSign;
    END PolylineSignClipStyleAssoc;

    ASSOCIATION PolylineSignStartSymbolAssoc =
      StartSymbol -- {0..1} SymbolSign; !! Symbol at start of line in opposite
                                        !! direction of line
      PolylineSign -- {0..*} PolylineSign;
    END PolylineSignStartSymbolAssoc;

    ASSOCIATION PolylineSignEndSymbolAssoc =
      EndSymbol -- {0..1} SymbolSign; !! Symbol at end of line in same
                                      !! direction as line
      PolylineSign3 -- {0..*} PolylineSign;
    END PolylineSignEndSymbolAssoc;

    !! Surface interface
    !! =================

    CLASS SurfaceSign (EXTENDED) =
      Clip        : (
        inside,
        outside
      );
      HatchOffset : SS_Float;
    PARAMETER
      Priority    : MANDATORY SS_Priority;
      HatchAng    : SS_Angle;  !! Default 0.0
      HatchOrg    : SS_Coord2; !! Default 0.0/0.0, Anchor point for hatching
                               !! or filling
    END SurfaceSign;

    ASSOCIATION SurfaceSignColorAssoc =
      FillColor -- {0..1} Color; !! Fill color
      SurfaceSign -- {0..*} SurfaceSign;
    END SurfaceSignColorAssoc;

    ASSOCIATION SurfaceSignBorderAssoc =
      Border -- {0..1} PolylineSign; !! Border symbology
      SurfaceSign -- {0..*} SurfaceSign;
    END SurfaceSignBorderAssoc;

    ASSOCIATION SurfaceSignHatchSymbAssoc =
      HatchSymb -- {0..1} PolylineSign; !! Hatch symbology
      SurfaceSign2 -- {0..*} SurfaceSign;
    END SurfaceSignHatchSymbAssoc;

  END StandardSigns;

END StandardSymbology.
```

Example

Cf. appendix E *A small example Roads*.

Annex M (informative) – Glossary

Common abbreviations, abbreviation of technical terminology see definitions

Abbr.	Abbreviation
Art.	Article (in legal texts)
Par.	Paragraph (in legal texts)
Def.	Definition
de	deutsch (German)
en	English
fr	français (French)
Syn.	Synonym
RefMan	The note RefMan e.g. RefMan 2.6.4 signifies that in paragraph 2.6.4 of this INTERLIS Version 2-Reference Manual (Standard eCH-0031) further information concerning this term can be found.
→ A	A is a term that has been defined in this glossary.

Definitions

Abstract class

→ Class, which cannot contain → objects.

Note: An abstract class is always incomplete and forms the base for → subclasses (i.e. → specializations) whose object set then need not be void.

Aggregation

Directed → proper relationship between a superior → class and an inferior → class. Several parts (sub-objects) of an inferior → class are assigned to an entity (meta object of the superior → class). It is also possible to assign several entities to a part. When copying an entity all assigned parts are copied as well. When deleting an entity assigned parts may continue existing.

Note 1: By means of an aggregation the → relationship between an entity and its parts is described (e.g. car/motor). The → role of the → subclass can be termed with "is-part-of".

Note 2: In → INTERLIS 2 an aggregation is indicated analogous to the → UML class diagram notation with a (void) rhombus (-<>).

Note 3: See also → composition.

Amendment

Consistency-saving → operation on a → database.

Area

→ Planar general surface of an → area division.

Syn. area object.

Area division

Set of → planar general surfaces which do not have any → points or only boundary points in common.

Area object

Syn. for → area.

Argument

→ Value of a → parameter.

Association

Proper relationship, which does not restrict the independence of the → classes concerned. Assigned → objects can, independently of each other, be copied or deleted.

Note 1: In → INTERLIS 2 the → association class is available for describing an association.

Note 2: See also → reference attribute, → aggregation and → composition.

Association class

→ Class element to describe an → association, → aggregation or → composition.

Attribute

Data (elements) corresponding to a specific characteristic of → objects of a → class and of → structure elements of a → structure (cf. RefMan 2.6.4). Each attribute has an attribute-name and a → domain.

Syn. property (en).

Note: Each → object of a → class likewise contains a → data element of an attribute with an individual → value. Graphically an attribute corresponds to the column of a → table.

Attribute specialization

Restriction of the → domain of an → attribute.

Note: Attribute specialization is also employed when defining → inheritance relationships.

Basic class

Ambiguous syn. for → super class and → view base class.

Basic data type

Predefined → value domain such as TEXT or BOOLEAN (cf. RefMan 2.8).

Basic view

→ View whose → objects contribute to the set-up of a new → view.

Basket

Collection of → objects that belong to a → topic or to its → extensions.

Bi-directional association

Def. cf. → association.

Boundary of a surface

Set of all boundary points of a → surface.

Cardinality

Number of → objects of → class B (resp. A) which can be assigned to an → object of → class A (resp. B) through the → relationship between the → classes A and B.

Syn. multiplicity.

Note: In → UML the term → multiplicity is also employed; in which case "cardinality" means the concrete number of → object relationships between → object instances.

Cartesian coordinate system

→ Coordinate system of the Euclidic → space whose axes are straights set perpendicularly and in pairs.

Cartographic sign system

Set of graphical representation possibilities for → graphic symbols.

Note 1: A concrete → graphic element shown on screen or printed on paper is the result of a multi-level process whereby → objects are selected (selection), then represented on → graphic symbols (mapping) and assembled, graphically rendered (rendering) and represented (display).

Note 2: In → INTERLIS 2 the first two levels are determined by means of a → representation description, all other levels are subject to the implementation of each system resp. "driver", in some cases certain graphic standards exist such as PostScript, HPGL, OpenGL, Java2D, SVG.

Change database

Temporary → database with whose → objects → amendments can be executed. A change database receives its → objects from a → primary database and returns them after their processing (→ update).

Note: A change database may operate on the same → system as the → primary database (internal change database) or on another → system (external change database).

Class

Set of → objects with the same properties and → operations. Each property is described by an → attribute, each → operation by a → signature.

Syn. object class, set of entities, object type, feature type, feature.

Note 1: A class described with → INTERLIS 2 corresponds to a UML-class with nothing but "public", i.e. visible → attributes.

Note 2: Cf. → super class, → subclass, → table as well as → class element.

Note 3: Classes need not necessarily contain → objects. If they do contain → objects we speak of → concrete classes, if not of → abstract classes.

Class diagram

Graphic representation of → classes and their → relationships.

Class element

→ Modeling element "of the modeling level class". To be exact: class elements are called → class, → structure, → association class, → view, → view projection and → graphic definition.

Class interface

Function call of a part or of the entirety of the → operations of a → class.

Syn. program interface, software interface, interface.

Note 1: One → class may have several class interfaces. For each of them a separate → interface class can be defined. The → conceptual schema of an → interface class consists only of→ signatures.

Note 2: See also → user interface and → data interface.

Class specialization

Restrictions of a → class through additional → attributes, → relationships, → consistency restraints or → attribute specialization.

Note: Class specializations are used for defining → inheritance relationships.

Complete data transfer

→ Data transfer of a complete → database state from → sender (source) to → receiver (target).

Comportment rule

Conditions under which on the one hand → messages from a sender system will be received, on the other hand a → message containing output arguments of the → class interface will be retransferred to the sender system of the → message including a call of a → class interface.

Composition

Directed → proper relationship between a superior → class and a subordinate → class. Several parts (sub-objects of the subordinate → class) are assigned to an entirety (super-object of the superior → class), while there is a maximum of one entirety that can be assigned to one part. When copying an entirety all assigned parts are copied at the same time. Likewise when deleting an entirety all assigned parts are also deleted.

Note 1: All parts are dependent; they inalterably form part of the entirety. Thus all → classes involved do not lead equal → relationships, but form a consists-of hierarchy.

Note 2: In → INTERLIS 2 a composition is defined as an → association class.

Note 3: Cf. → structure attribute.

Conceptual schema

Def. cf. → data schema (note 2).

Syn. conceptual → data schema.

Concrete class

→ Class which can contain → objects.

Note: Cf. → abstract class.

Consistency constraints

Restrictions all → objects must comply with.

Syn. condition, limiting condition, assurance, constraint (en).

Note: Certain consistency constraints are predefined in → INTERLIS 2. Other consistency constraints can be formally defined by means of → functions, → logical expressions or rules and are subject to a → contract.

Constraint attribute

→ Attribute, for which a → consistency constraint has been defined.

Constraint class

→ Class, for which a → consistency constraint has been defined.

Constraint

Syn. for → consistency constraint.

Contract

Agreement with software-tool suppliers.

Note: Contracts are e.g. required, if in INTERLIS 2 data models not predefined → functions, → symbology models or not predefined → line form types are used.

Conversion

→ Mapping of a → coordinate system (resp. of its → space) to another → coordinate system (resp. to its → space), strictly defined by formulas and their → parameters.

Note: The term conversion occasionally is used for the reformatting of → transfer files.

Coordinate reference system

Syn. for → reference system.

Coordinate system

Base of an Euclidic vector space, resp. original base of the assigned Euclidic vector space when dealing with map homomorphism of a diversity (for details see vector analysis).

Note: From the viewpoint of data a coordinate system is defined by its axes which either are straights (in → INTERLIS 2 so-called LengthAXIS) or elliptic arcs (so-called AngleAXIS) depending on the type of → space they permit to measure.

Corner

Not smooth part of a → line string.

CSL

Abbr. for Conceptual Schema Language.

Curve segment

Subset of the → space, it is the image set of a smooth and injective → mapping of an interval of the numerical straight line.

Syn. line segment.

Data abstraction

Abstracting (amongst others omitting) of unimportant details via data.

Note 1: Separating What? (→ class interface → type) from How? (→ class, concrete implementation). → generalization and → specialization are possible principles of abstraction.

Note 2: The actual realization of the → operations and the inner structure of the → object or → structure element are hidden, i.e. characteristics are considered in an abstract way and no attention is being paid to the actual implementation.

Data catalogue

Syn. for → object catalogue.

Data description

Syn. for → data schema and → data model.

Data description language (DDL)

Formal language for the exact description of data structures.

Syn. Conceptual Schema Language (CSL).

Data element

Def. cf. Informatics, cf. → domain.

Data interface

Program for the reformatting of → transfer files or → protocol for the → data transfer.

Syn. interface.

Note: See also → class interface and → user interface.

Data model

Exact description of a data structure (so-called conceptual → data schema), which is a complete and self contained unit. From the hierarchy point of view a d. is the highest → modeling element.

Syn. model, data description.

Note 1: Beware! In database theory data model is a common synonym for conceptual formalism (i.e. a data model is considered as a → method for the creating of a → conceptual schema).

Note 2: A data model consists of at least one → topic.

Note 3: In → INTERLIS 2 described by the key word MODEL. The → package, which corresponds to this data model, is above all other → packages, which correspond to the → topics of a data model.

Data schema

Description of content and organization of data characterizing a user-specific facet of reality, as well as rules governing these and of → operations which can be executed with such data.

Syn. data description, schema, conceptual schema, ontology.

Note 1: Plural: data schemas.

Note 2: Depending on the abstraction level at which the data are described, we distinguish between the → conceptual schema, the logical schema and the physical schema. When formulating a data schema we dispose of appropriate → data description languages.

Note 3: When dealing with → databases the logical schema formulated in accordance with the → conceptual schema and the system specific possibilities of organization, is also called internal schema. Logical as well as physical schemas of peripheral instruments or exchange files are often called external schemas or format schemas.

Data transfer

Transfer of data from one → database A to another → database Z. A is known as primary system, source, → sender, sender system, Z as → target system, → receiver. The delivery of data to be transferred by → system A is also called export; its acceptance by → system Z is called import.

Syn. transfer, data transmission.

Data transfer mechanism

(Conceptual) → data description language and (physical) → transfer format as well as rules governing the derivation of such a → transfer format of a data structure that is described by means of a → data description language.

Data type

Syn. for → domain.

Database

Logical administration unit for the treatment and long-term memorization of → objects.

Abbr. DB.

Note: It is possible to run several databases on one → system. It is also conceivable that one database has been divided into several → systems.

Database state

Totality of all data and → relationships of a → database at one given moment. Each database state has its name.

Note: By means of one or several → amendments a → database is transferred from one database state to the next (→ update).

Date

Ambiguous syn. for → geodetical date and indication of time (e.g. 2002-06-25).

Date transformation

→ Transformation of a → geodetical date (resp. of the → space defined thereby) on another → geodetical date (resp. its → space).

Derived attribute

→ Attribute, whose → domain is calculated by means of a function regulation (→ logical expression, calculation).

Note 1: Derived attributes cannot be altered.

Note 2: In → INTERLIS 2 the function regulation is defined via a → function.

Directed relationship

→ Aggregation or → composition or → reference attribute or → inheritance relationship.

Domain

Set of homogeneous → data elements. A → data element of a domain is called → value.

Syn. data type.

Note 1: Cf. → basic data type.

Note 2: A value can also consist of → structure elements of a → sub-structure.

Domain of a name

→ Namespace of the → name category of this name corresponding to the → modeling element, in which this name is defined.

Note 1: Within the domain of a name each name may only have one definition/meaning. However the same name can be defined once within the → namespace of every → name category of the same → modeling element.

Note 2: The domain of a name is part of the → visibility domain of a name.

Drawing rule

Language element of a → graphic definition. A drawing rule assigns a → graphic symbol to the → objects) of a → class and determines the corresponding graphic symbol arguments according to the attribute values (i.e. data) of the → objects.

Syn. symbol attribute.

Element

Fundamental idea of the set theory. A set consists of elements.

Syn. instance.

Note: See also → modeling element or → graphic element.

Ellipsoid coordinate system

→ Coordinate system on the 2-dimensional boundary surface of a 3-dimensional (rotation-) ellipsoid.

Ellipsoid height

Euclidic distance of a point measured from the ellipsoid along the normal line to surface through this point.

End point of a curve segment

Picture of the other interval end point with the → mapping which defines the → curve segment.

Entity

Syn. for → object.

Expression

Syn. for → logical expression.

Extension

Syn. for → specialization.

Feature

Syn. for → object, resp. often also for → class.

Feature type

Syn. for → class.

File

Def. cf. informatics.

Force

Link between parts (sub-objects of the subordinate → class) and the entirety (super object of the superior → class) in a → proper relationship.

Function

→ Mapping of → value domains of input-parameters into a → value domain of an output-parameter by means of a calculation-rule (→ parameter).

Note: In → INTERLIS 2 certain functions are predefined, others are subject to a → contract.

General identification

→ Identification for all (modeled) → objects of a → transfer community.

Note: See also → object identification.

General surface

→ Surface with an additional finite number of → singular points however with a continuous → interior of the surface.

Generalization

→ Role of the → super class in an → inheritance relationship.

Note 1: Generalization is occasionally used as a synonym for → inheritance (even though it actually means the opposite direction).

Note 2: In cartography generalization describes all activities due to the scaled and reduced → mapping of real-world → objects.

Geodetical date

3-dimensional → Cartesian coordinate system, whose axes have a fixed position and orientation as to the center of gravity and the rotation axis of the earth.

Syn. geodetical reference system.

Geodetical reference system

Syn. for → geodetical date.

Geoide

Equipotential surface of the field of gravity.

Note: A geoide supplies a physical earth model that adjusts to the gravity field of the earth. It is of irregular form since it takes into consideration the irregular mass distribution of the earth. It has to be imagined as if the average ocean surface were to continue beneath the continents.

GIS

Abbr. for geo-information system or geographical information system.

Graphic definition

→ Class element of a → graphic topic, i.e. each → graphic topic of a → graphic description is a collection of graphic definitions (not of → classes!). Each graphic definition belongs to a → class (BASED ON) of the corresponding data-topic, assigns by means of → drawing rules → one or several → graphic symbols to objects of this → class and determines the → arguments of the → graphic symbol according to the data of the → objects.

Note: The data of the → graphic symbols, i.e. their names and graphic visualization are comprised in a → symbol library described in the corresponding → symbology model.

Graphic description

Syn. for → representation description.

Graphic element

Graphic description of an → object taking into consideration its 2-dimensional geometry and further → attributes of this → object, after possibly necessary processing ready for output by a suitable peripheral device.

Syn. graphic object.

Graphic model

Syn. for → graphic description.

Graphic object

Syn. for → graphic element.

Graphic parameter

Syn. for → parameter of a → graphic symbol.

Graphic symbol

Data for the graphic representation of an → object still independent of 2-dimensional geometry and further attribute values of this → object. A → graphic parameter is called → parameter of graphic symbol.

Syn. symbol, style.

Note 1: There are four types of graphic symbols: (1) text, resp. text symbol (sometimes called text label or simply label), (2) point symbol (sometimes also called point sign or simply → symbol or pictogram), (3) line symbol and (4) (single) surface symbol.

Note 2: In → INTERLIS 2 the data structure and possible → parameters of a graphic symbol are specified within a → symbology model and the corresponding data are stored within a → symbol library. A graphic symbol is referenced via its graphic symbol-name within a → graphic definition. Thereby corresponding → arguments for eventual → parameters have to be defined.

Graphic topic

Def. cf. → representation description.

Gravity model

Description of the gravity field of the earth.

Height

Either → ellipsoid height or → normal height or → orthometrical height.

Help line

Linear → graphic element which links two → graphic elements or one → graphic element and a label.

Note: A typical case of a help line is the representation of a join from a line or surface symbol to a label or its corresponding measurement line.

IDDL

Abbr. for → INTERLIS Data Description Language (IDDL).

Identification

→ Attribute or combination of attributes whose → value unequivocally determines an → object within its → class.

Abbr. ID.

Syn. identifier, identity.

Note: Within an INTERLIS-transfer file each → object is assigned an identification in addition to the attribute values described in the → data schema, thus being unequivocally identified within the → transfer file. This is a so-called → transfer identification (→ TID). If such a → TID is a → general and → stable identification, then it is called an → object identification (→ OID).

Identifier

Syn. for → identification.

Identity

Syn. for → identification.

ILI

Abbr. for → INTERLIS.

Note: Also common as data name extension of → files which contain a → data schema conceived in → INTERLIS (version 1 and 2).

Implemented class

Executable software module with → operations realized as → methods.

Incremental data transfer

→ Data transfer of the difference between two → states of database from a → sender to a → target system.

Incremental update

→ Complete or → incremental → data transfer of a → database state of the → primary database to a → secondary database.

Note 1: An incremental update always proceeds sequentially, i.e. one → secondary database will never have to receive several incremental updates at the same time.

Note 2: Cf. → synchronization.

Information layer

Non-void set of → topics.

Inheritance

→ Method for the definition of → inheritance relationships between → super classes and → subclasses. These → methods are → class specialization and → attribute → specialization.

Note 1: → Subclasses correspond to the same idea; they have the same properties as their → super classes that they specialize.

Note 2: We distinguish between → single inheritance and → multiple inheritance. In the case of a → single inheritance (de: Einfachvererbung; fr: héritage singulaire) one → subclass inherits → only from one direct → super class. In the case of a → multiple inheritance one → class inherits from several → super classes.

Note 3: → INTERLIS 2 only admits simple inheritance (such as Java).

Inheritance relationship

→ Directed → relationship between a superior → class, called → super class, and a subordinate → class, called → subclass, defined by → inheritance. The → role of the → super class is called → generalization; the → role of the → subclass is called specialization.

Note 1: The → objects of the → super class are → generalizations of the → objects of the → subclass. The → objects of the → subclass are restrictions (→ specializations, → extensions) of the → objects of the → super class.

Note 2: The inheritance relationship is a subset relationship, the → objects of the → subclass form a subset of the → objects of the → super class. Thus in the case of → objects of the → subclass we do not deal with new → objects, but with a part or subdivision of the → objects of the →

super class. Both → objects of an object pair of the inheritance relationship possess the same → OID.

Inner boundary

Subset of the → edge of a → planar surface, it is an interior → simple closed line string.

Instance

Syn. for → element (concrete specimen) of a set (abstraction).

Note: Examples of an instance: A → value is an instance of a → data type. An → object is an instance of a → class. A → basket is an instance of a → topic. An object pair is an instance of an → association class.

Interface

Ambiguous syn. for → class interface, → user interface and → data interface.

Interface class

Def. cf. → class interface.

Syn. class of class interface.

Interior of a surface

Set of all inner points of a → surface.

INTERLIS 2

→ Data transfer mechanism for geodata consisting of → INTERLIS Data Description Language (→ IDDL) and the INTERLIS-XML transfer format (IXML) as well as rules for the derivation of IXML for a data structure described with → IDDL. → IDDL, IXML and conversion rules are defined in RefMan 2 or 3.

Abbr. for "INTER land information systems" (i.e. between → GIS).

INTERLIS-Compiler

Program that derives the description of the corresponding INTERLIS → transfer format from a → data schema in → IDDL. At the same time the syntactic correctness of the data schema is examined (so-called parsing), cf. RefMan appendix A.

INTERLIS Data Description Language (IDDL)

(Conceptual) → data description language of the → data transfer mechanism INTERLIS.

Note: A → data schema described in → IDDL can be memorized as a text file. For such schema files it is common to add the abbreviation "ILI" to the file name. Example: The schema file of the database set of Cadastral Surveying is thus called DM01AV.ILI.

Layer

Within the scope of CAD a common term for the collection of graphic data of a certain → type. Occasionally used in GIS for → topic.

Layout of the plan

Description of plan by means of → meta data title, → legend, producer description, issue date, definition of sign type and graphic representation of further → elements such as grid intersections and north direction.

Syn. layout of the map.

Legend

Labeling and explanation of a map, resp. plan and the → graphic symbol employed therein.

Note: Cf. → graphic description as well as → symbol library.

Line form type

Form of curves that make up a line string (straights, arcs and other connecting geometries). For the definition of object geometries supported by → INTERLIS 2, see RefMan 2.8.12 and 2.8.13.

Line segment

Syn. for → curve segment.

Line string

Subset of the → space, image set of a continuous and partially smooth (but not necessarily injective) → mapping (the so-called assigned → mapping) and which only features a finite number of not-smooth parts (so-called → corners).

Logical expression

Predicates joined by means of Boolean operators.

Syn. expression.

Map frame

Confines the limits within which the contents of a plan are represented.

Note: Towards the exterior edge it is possible to define graded covering regions.

Map projection

→ Conversion of an elliptic or spherical → space into an Euclidic → plane.

Map symbol

Syn. for → graphic symbol.

Mapping

(From space A, defined by a → coordinate system in another space Z, defined by a second → coordinate system:) Regulation which assigns exactly one point z of Z to each point a of A.

Note: Special cases of mappings are → transformation and → conversion.

Message

Data containing calls for → class interfaces including → arguments for the input resp. data containing → arguments for the output of → class interfaces.

Metadata

Data dealing with data.

Note: Especially in geo-informatics metadata is data that indicate amongst others object-descriptions in colloquial language, of → objects, organization, space reference, quality, availability and origin, etc.

Metamodel

→ Data model of → metadata.

Metaobject

→ Object, whose subject of the real world is a set of → objects.

Note 1: Thus a metaobject consists of → metadata. Metaobjects exist for individual → objects and/or for all → objects of a → modeling element.

Note 2: → Metadata for the → values of individual → attributes of → objects are additional → attributes of the → class of these → objects.

Metaobject-names

→ Name category that solely consists of names of → metaobjects.

Method

Implementation of an → operation by a series of instructions (i.e. by a program).

Note: ambiguous term, often used as synonym for → operation.

Model

Syn for → data model.

Note: The object-oriented modeling distinguishes between object models (as synonym for the part of a → data schema which describes content and organization of data) and models of behavior (as synonym for the part of a → data schema which describes → operations that can be executed with the data).

Model driven approach

Approach which leads from user-specific detail of reality via a → conceptual schema to data and programs for their processing.

Syn. model driven architecture (en).

Abbr. MDA (en).

Note 1: There are four phases to the model driven approach which lead to the following results: (1) Description of the real world detail in colloquial language, (2) conceptual, (3) logical, (4) physical → data schema. Both phases (1) and (2) and their results are system-independent.

Note 2: For the establishment of a → conceptual schema tools such as → UML and → INTERLIS 2 will be used. → INTERLIS 2 also supplies coding rules which permit the deriving of physical → data schema of a → transfer file (the → transfer format) from a → conceptual schema (in → INTERLIS 2 → CSL).

Note 3: One of the main advantages of a model driven approach consists in the precise wording, above all of the → conceptual schema, which then permits communication about and comprehension of data structure between experts.

Model driven method

Syn. for → model driven approach.

Model driven protocol

→ Protocol whose → class interfaces and → messages are described by means of a (system-independent) → conceptual schema.

Modeling element

Special → schema element. There are three modeling elements, namely → data model, → topic and → class element.

Note: Modeling element and → name category define the → namespace.

Multiple inheritance

→ Inheritance relationship that assigns more than one → super class to one → subclass.

Note: Multiple inheritance is not provided in → INTERLIS 2.

Multiplicity

Syn. for → cardinality.

Name category

Subset of the names of a conceptual → data schema. There are three name categories, namely → type names, → part names and → metaobject names.

Note: Name category and → modeling element define the → namespace.

Namespace

Set of (unequivocal) names of a → name category in a → modeling element.

Note: The namespace is of importance when determining the → domain and the → visibility domain of a name.

Normal height (of a point)

Distance between the point and the quasi-geoid.

Note: The normal height is a rigorous height with respect to potential theory. The mean normal gravity is taken into account.

Object

Data of a real-world object together with the → operations that can be executed with these data and with an → object identification.

Syn. entity, tuple, object instance, feature, feature instance.

Note 1: Cf. → instance, → class.

Note 2: As opposed to a → value, an object possesses an → identity, exists in time and → space, can be altered while keeping its → identity and by means of a reference it can be of common use. An object is concrete. It is tightly connected with the existence of real things.

Note 3: In object-oriented literature we find the following flowery definition of the term object: A concrete existing unit with its own (unchangeable) → identity and defined limits (in the figurative sense of the word) which encapsulate state and characteristics. Its state is represented by → attributes and → relationships, its characteristics by → operations. Each object belongs exactly to one → class. The defined structure of their → attributes, as well as their characteristics, applies likewise to all objects of one → class. However the → values of the → attributes are specific for each object.

Object catalogue

Informal enumeration of → classes with colloquial definitions (name and description of the → class) of all data objects relevant for one utilization.

Abbr. OC.

Syn. data catalogue.

Note 1: An object cataloque comprises indications concerning degree of detailed description quality requirements (mainly geometrical quality) as well as rules for recording.

Note 2: An object catalogue is a preliminary and a complement of the conceptual → data model.

Object class

Syn. for → class.

Object identification

→ General and → stable identification.

Syn. object identifier, object identity.

Note 1: Usually the object identification is only altered by a → system and not by a user. An object identification is a property that distinguishes one → object from all others, even though it may possess the same attribute values.

Note 2: Cf. → transfer identification.

Note 3: In appendix F of the INTERLIS Version 2-Reference Manual you will find a suggestion for an object identification.

Object identifier

Syn. for → object identification.

Object identity

Syn. for → object identification.

Object instance

Syn. for → object.

Object relationship

Two → objects assigned to each other by a → relationship between the → classes they belong to.

Syn. link.

Object type

Syn. for → class.

Official height

Sum of all leveling measurements (height differences) along a leveling run of one point of → height 0 to a point with wanted G.

OID

Abbr. for → object identification.

Onesided relationship

Syn. for → reference attribute.

Ontology

Syn. for → data schema.

Note 1: Ontology is an "explicit formal specification of a shared conceptualization", i.e. in graphic terms, a repository of concepts.

Note 2: Ontologies use UML/OCL or their own languages such as DAM/OIL (DARPA Agent Markup Language + Ontology Interchange Language). Typically ontologies consist of → conceptual data schema, a taxonomic hierarchy of → classes (vocabulary, thesaurus) and axioms, which restricts possible interpretations of the defined terms (in most cases with a logic-language). (In the future) ontologies should be used as higher abstractions of → data schemas for the specification of software and for the communication between individuals.

Operation

→ Mapping from the attribute domains of a → class and/or from → domains of input-parameters into the → domain of an output-parameter.

Note 1: The implementation of an operation by means of a series of instructions (i.e. by a program) is called → method.

Note 2: The description of an operation is called → signature and consists of operation names and description of the → parameters.

Optional

Not compulsory, need not exist or be applicable. Contrary: mandatory.

Note 1: → Attributes are optional, unless it is stated that they are to be mandatory. For mandatory → attributes we dispose of the keyword MANDATORY in → IDDL.

Note 2: In → IDDL the term "not mandatory" refers to the → transfer file.

Orthometric height

Length of curve of the (curved) perpendicular between → geoid and point.

Outer boundary

Subset of the → edge of a → planar surface, the outermost → simple closed line string.

Package

Element of the UML-language for the description of → models, → topics and parts of topics.

Note 1: A package defines a → namespace, i.e. within a package the names of the → schema elements contained must be unequivocal. Each designated → schema element can be referenced in a different package, but it belongs to exactly one (source-) package.

Note 2: In the case of → UML packages themselves can contain other packages. The top package contains the entire system according to the → data model of → INTERLIS 2.

Parameter

Data (elements), whose → values are transmitted to a → function, an → operation or a → meta object and/or have been returned by → functions or → operations. Each parameter is supplied with a name, a → domain and - where → functions or → operations are concerned - a transfer direction (in, out, inout). The concrete → value of a parameter is called → argument.

Note 1: Cf. → run time parameter.

Note 2: By means of parameters we describe those properties of → meta objects which do not concern the → meta object itself, but its use within the application.

Part names

→ Name category consisting of names of → run time parameters, → attributes, → drawing rules, → parameters, → roles, → relationship access and → basic views.

Path

Series of names of → attributes and/or → classes and/or → roles of → association classes, which define an → object or the → value of an → attribute which are to be processed by a → logical expression.

Planar curve segment

→ Curve segment which is a subset of a → plane.

Planar general surface

→ General surface which is a subset of a → plane.

Planar surface

→ Surface which is a subset of a → plane.

Plane

2-dimensional sub-space of a → space.

Point

(Set) element of the → space (considered as a set).

Polymorphism of objects

Wherever → objects of a → super class are expected it is also possible to have → objects of an → extension.

Syn. polymorphy, polymorphism.

Note 1: See also → polymorphism of operations.

Note 2: In → INTERLIS 2 we refer mainly to polymorphism of objects.

Polymorphism of operations

Based upon the → signature it is conceivable that → objects of different → classes respond to identical operation names (messages), i.e. they are processed by → operations with identical names.

Syn. polymorphy, polymorphism.

Note 1: See also → polymorphism of objects.

Note 2: In INTERLIS 2 mainly → polymorphism of objects is applied.

Primary database

→ Database that deals with long-term administration of → objects of certain → topics of a certain field.

Program

Syn. for → method.

Program interface

Syn. for → class interface.

Examples: Java-API or Open Database Connectivity (ODBC).

Program system

Entirety of all → methods of → classes necessary for the processing of an application by means of electronic data processing.

Propeller set

Union of a finite number of triangular surfaces which have exactly one → point in common, the centre.

Proper relationship

Def. cf. → relationship.

Property

Syn. for → attribute.

Protocol

Entirety of all → class interfaces → messages and → comportment rules of a set of → systems which contribute to the solution of an application task.

Receiver

Def. cf. → data transfer.

Syn. target system.

Reference attribute

→ Relationship which is only known to the first → object of each object pair of the → relationship.

Syn. onesided → relationship.

Reference system

→ Coordinate system, appearing at the end of a series of → coordinate systems and → conversions where exactly one → geodetical date occurs and which stands at the start of the series.

Referential integrity

Rule, which determines what is to happen with an → object relationship, resp. with the → objects concerned, if the → objects involved or the → relationship itself is deleted.

Relationship

Set of object pairs (resp. in the general case of object-n-tuples also known as → relationship objects). The first → object of a pair belongs to a first → class A, the second object to a second → class B. The attribution of → objects to such pairs shall be predefined; hence it must only be described, i.e. modeled. We distinguish between → proper relationship (that is → association, → aggregation, → composition), → inheritance relationship and → reference attribute.

Note 1: As proved by the view concept, it is on the other hand also possible to calculate such assignments by means of algorithms, e.g. based upon attribute values.

Note 2: Cf. → object relationship.

Note 3: For a proper relationship both → force and → cardinality are defined.

Relationship access

Conditions and possibilities to refer to → relationship objects and by these means also → objects of (ordinary) → classes via → paths.

Relationship object

Def. cf. → relationship.

Replicate

To copy whereby the copied → object may not be altered independently of the original.

Note: Term mainly used in connection with → incremental update.

Representation description

→ Conceptual schema, which describes the assignment of → graphic symbols to → objects and consists of graphic → topics. The → objects can be selected in a → view.

Syn. graphic model, graphic description.

Note 1: A representation description in → INTERLIS 2 consists of graphic → topics each of which corresponds to a data topic (DEPENDS ON). A graphic → topic is a collection of → graphic definitions (not of → classes!).

Note 2: The representation description itself can also contain → data schemas (e.g. → classes which describe text positions).

Role

Significance of the → objects of a → class within a → relationship.

Note: In a → proper relationship the role of each → class involved is described by its name, its → force and its → cardinality. A → reference attribute describes the role of the → class with this → attribute. Within an → inheritance relationship roles are implicitly defined.

Run time parameter

→ Parameter whose → value is supplied at run time by a treatment, evaluation or representation system.

Note: Examples are representation scale, → date.

Schema

Syn. for → data schema.

Schema element

Partial schema of a conceptual → data schema that possesses a name.

Note: All → modeling elements are schema elements.

Secondary database

Copy of the → database state of a → primary database.

Note: Usually a secondary database cannot be found on the same → system as the → primary database.

Sender

Def. cf. → data transfer.

Set of entities

Syn. for → class.

Sign

Letter or digit or blank or punctuation mark or symbol.

Signature

Describing the call of an → operation, consisting of the name of the → operation, its → data types and possible names of their → parameters and possibly indication of a return-data type.

Simple closed line string

→ Line string whose assigned → mapping is injective, with the exception of its → start point and → end point which coincide.

Simple inheritance

Def. cf. → inheritance.

Simple line string

→ Line string whose assigned → mapping is also injective.

Singular point

→ Point which, together with its environment can be deformed into a planar → propeller set, the point itself being at its centre.

SN

Abbr. for Swiss → Norm.

Software interface

Syn. for → class interface.

Space

3-dimensional Euclidic space.

Specialization

→ Role of the → subclass of an → inheritance relationship, often also synonym for → inheritance.

Syn. extension.

Note 1: Cf. → class specialization und → attribute specialization.

Note 2: Since more text will be necessary for the description of a → class or → attribute specialization than for → the super class or the original attribute, we often rather speak of → extension than of specialization.

Stable identification

→ Identification which is independent of time, i.e. it cannot be altered during the life cycle of an → object. Once an → object has been deleted, its stable identification no longer can be used.

Note: Cf. → object identification.

Standard

A 'de jure' standard (or short standard) is a technical regulation laid down by national or international committees for standardization. A 'de facto' standard is a generally acknowledged and majority used technical regulation, but less binding than a 'de jure' standard.

Note 1: A law is a regulation superior to both 'de jure' and 'de facto' standards.

Note 2: German synonym for 'de facto' standard is "standard". In English standard is used for 'de facto' or 'de jure' standards.

Starting point of a curve segment

Representation of one of the interval end points when establishing the → mapping defining the → curve segment.

String

Succession (i.e. ordered set) of → signs.

Structure

Set of → structure elements with the same properties and → operations. Only such → operations are permitted which will not alter the data of the → structure elements. Each property is described by an → attribute, each → operation by its → signature.

Note 1: Structures occur either within LIST- or BAG-attributes (→ substructure) or exist only temporarily as the result of → functions.

Note 2: Cf. → class element.

Structure attribute

→ Attribute with the INTERLIS 2-data type BAG or LIST.

Note: As opposed to the definition of a → composition by means of the → association class, with a structure attribute → structure elements cannot be referenced, i.e. outside of the → object to whose structure attribute value they belong to, they have no identity.

Structure element

Data of an object of the real world with → operations that could be executed with these data, however without permission to alter them, and without → object identification.

Note: A structure element is the → instance of a → structure.

Structured domain

INTERLIS 2 language element for the description of compound → attributes such as → date or time.

Subclass

Def. cf. → inheritance relationship.

Substructure

→ Domain that has been defined by means of a → structure.

Note: Cf. → structure attribute.

Super class

Def. cf. → inheritance relationship.

Surface

Union *F* of a finite number of → surface elements which is continuous and complies with the following condition: for every → point P of the surface there exists an environment which can be deformed into a planar polygon (i.e. permits homeomorph mapping). If in the course of such a deformation → point P should be placed in the boundary of the polygon, it becomes a boundary point of F, otherwise it remains an inner point of F.

Surface element

A surface element is a subset of the → space, which is the image set of a smooth and injective → mapping of a planar regular polygon.

Symbol

Ambiguous syn. for → graphic symbol, language symbol or semiotic symbol.

Symbology

Subset of elements of a → cartographic sign system consisting of → graphic symbols, fonts, diagrams, half-tones.

Note: Cf. → symbol library.

Symbol attribute

Syn. for → drawing rule.

Symbol library

Collection of → graphic symbols, structured according to a → symbology model.

Note 1: A symbol library always is a → basket, i.e. an XML-file.

Note 2: In most cases a symbol library is a concrete, user-specific collection of → graphic symbols.

Symbology model

→ Conceptual → schema which describes the data structure of → graphic symbols and their → parameters.

Note 1: Symbology models always demand → contracts.

Note 2: In appendix L of the INTERLIS Version 2-Reference Manual you will find a suggestion for an extended symbology model.

Note 3: Cf. → symbol library.

Symbol object

Syn. for → graphic symbol.

Synchronization

Automatic and regular adjustment of the → database states of two → databases.

System

Totality of all components (hardware and software) forming a data processing-system and being put to a certain use.

Table

→ Class for whose → objects it is impossible to explicitly define → operations.

Target system

Syn. for → receiver.

TID

Abbr. for → transfer identification.

Topic

Set of → classes whose data in a certain sense belong together, e.g. they have a relationship, belong to the same data processing authority or possess a similar rhythm of updates. → Instances of topics are → baskets (recipients).

Note 1: In → UML, a → topic is described by a → package beneath a → data model described, with the additional significance that this → package (a) possesses its own → namespace and (b) may depend (be an extension) of other → packages. A UML-package, assigned to a topic, may contain other (nested) → packages.

Note 2: Beware: With → layer, in CAD terms a commonly used expression for "surface", we mean a collection of graphic data. A topic may comprise several (graphic) → layers plus additional structured thematic data.

Transfer

Syn. for → data transfer.

Transfer community

Community of → senders and → receivers who both participate in a → data transfer.

Transfer file

→ File prepared for → data transfer → in an appropriate → transfer format.

Transfer format

System of data fields within a transfer file.

Syn. format.

Transfer identification

Def. cf. → identification.

Abbr. TID.

Transformation

→ Mapping from one → coordinate system (resp. from its → space) to another → coordinate system (resp. to its → space), where the mapping regulation (formula) is based on hypotheses and the → parameters are established by means of mostly statistical analysis of measurements in both → coordinate systems.

Tuple

Syn. for → object.

Type

Ambiguous syn. for → data type (i.e. → domain), → class interface, and → signature.

Type name

→ Name category consisting of → topics → classes, → associations, → views, → graphic definitions, → baskets, → units, → functions, → line form types, → domains, → structures.

UML

Abbr. for Unified Modeling Language.

Def. cf. www.omg.org/.

Unit

Basic element of a measuring scale (examples: meters, seconds).

Update

One or several → amendments on a → primary database. By means of an update the → primary database is transferred from one → database state to the next.

Note: Several → amendments on the → primary database may occur parallel at the same time. In the case of parallel → amendments, the → primary database must guarantee the consistency of the result.

User interface

Graphic interface of a computer program.

Syn. interface, graphic user interface.

Note: See also → class interface and → data interface.

Value

→ Data element of a → domain.

Vertex

Syn. for → corner.

View

→ Class whose → objects are created by combining and selecting (to be exact by → view operations) → objects of other → classes or views.

Note 1: → Objects of a view are not "original" in the sense that they do not directly correspond to a real world object. Thus a view is sort of a virtual → class.

Note 2: Cf. → class element.

View base class

→ Vlass whose → objects participate in the forming of a → view.

View operation

Regulation for the definition of a new → object from the → objects of → view base classes, resp. → basic views.

Note: View operations of → INTERLIS 2 are projections, joins, unions, aggregations and inspections. Subsequently the object set can be restricted by means of selections.

View projection

→ Class whose → objects are determined by complementing → attributes selected from → objects of another → class, → view or view projections. In particular it is possible to define further (virtual) → attributes whose → values are determined by → functions.

Note 1: → Extensions of view projections are possible. However their → objects will always remain subsets of the object-set of the → basic class, → basic view or basic view projection.

Note 2: Cf. → class element.

Visibility domain of a name

Set of all → namespaces out of which the name may be referenced in an unqualified manner. The visibility domain of the name consists of its definition domain and of the → namespaces of its → name category in all → modeling elements that hierarchically are subordinated to the modeling element of its definition domain.

Note: Besides the → namespace of its definition domain a name can be newly defined in each → namespace of its visibility domain. Thus this → namespace will become the new → domain of a name. This new definition domain and its assigned visibility domain "override" part of the original visibility domain in so far as in this subdomain (which forms a subtree of the modeling element-hierarchy) only the new definition/ meaning of the name will apply.

Anhang N (informativ) – Index

A

ABSTRACT 25, 26, 27, 29, 32, 33, 36, 38, 39, 44, 46, 47, 53, 59, 60, 62, 63, 71, 74, 76, 100, 101, 102, 149, 152, 168, 177, 178, 179, 180
ACCORDING 26, **76**, 79
AGGREGATES 26, **67**, 68, 72
Aggregation **73**
AGGREGATION 26, 72, **73**
AlignmentType 39, **43**
ALL 26, 41, 64, 67, 69, 73, 74, 86, 134
AND 26, **66**, 67
ANY 26, 31, **48**, 49, 100
ANYCLASS 26, **34**, 67, 69, 70, 71, 101
ANYSTRUCTURE 26, **34**, 35, 67, 70, 71, 101
ARCS 26, 52, **54**, 100, 146, 147, 177, 178
ArcSegment **91**
AREA 26, **54**, 58, 59, 63, 64, 72, 73, 74, 81, 88, 91, 92, 146, 147
Argument **67**
ArgumentType **69**
AS 26, **29**, 30, 31, 32, 36
ASSOCIATION 26, 32, **35**, 36, 74, 121, 147, 152, 167, 168, 169, 179, 180, 181, 182
AssociationDef 30, **36**
AssociationPath **67**
AssociationRef 34, **36**
AT 26, **28**, 76, 77, 78, 100, 121, 122, 134, 147, 149, 151, 158, 166, 175, 177, 178
Attribute 85, 87, **88**, 90
ATTRIBUTE 26, **32**, 36, 49, 63, 71, 74, 86, 90, 101, 134, 182
AttributeDef 32, **33**, 36, 74
AttributePath 49, 65, **67**, 75, 76
AttributePathConst 40, **49**
AttributePathType 39, **49**
AttributePathTypeValue 88, **90**
AttributeRef **67**
AttributeValue **88**
AttrType **33**
AttrTypeDef 33, 49, 62, 63, 69

B

BAG 8, 9, 26, **33**, 35, 64, 68, 70, 71, 72, 81, 88, 92, 101, 152
BASE 26, **74**
BaseAttrRef **45**
BASED 26, **45**, 46, 75, 76, 78, 102, 134, 135, 151, 158, 166
BaseExtensionDef 71, **73**, 74
BaseType **39**
Basket **84**, 85
BASKET .. 26, **29**, 30, 46, 61, 64, 65, 77, 79, 102, 134, 158, 176
BINARY 26, **49**
BLACKBOX 26, **49**, 89, 90
BlackboxType 39, **49**
BlackboxValue 88, **89**
BOOLEAN . 26, **43**, 70, 71, 74, 89, 100, 101, 151, 152, 179, 181
BooleanType 39, **43**
Boundaries **91**, **92**
BY 26, **74**

C

Cardinality 33, **36**
CARDINALITY 26, **36**
CHARSET 26, **28**, 29
CIRCULAR . 26, 41, 43, 44, 46, 47, 102, 121, 151, 158, 166, 177, 178
CLASS . 9, 24, 25, 26, 31, **32**, 46, 47, 49, 62, 63, 64, 67, 74, 76, 77, 78, 90, 101, 102, 121, 122, 146, 147, 152, 158, 166, 167, 176, 177, 178, 179, 180, 181, 182
ClassConst 40, **49**
ClassDef 25, 28, 30, **32**
ClassOrAssociationRef **34**
ClassOrStructureDef **32**
ClassOrStructureRef **32**, 34, 49
ClassRef **32**, 34, 62, 76
ClassType 39, **49**
ClassTypeValue 88, **90**
CLOCKWISE 26, **44**
ComposedUnit **60**
CondSignParamAssignment 75, **76**
Constant **39**, 67, 76
CONSTRAINT . 26, 64, **65**, 67, 73, 103, 151, 152, 167, 168, 181
ConstraintDef 32, 36, **65**, 71
CONSTRAINTS 26, 39, **65**, 147
ConstraintsDef 30, **65**
CONTEXT 26, **48**
ContextDef 28, **48**
CONTINUOUS 26, **33**, 46, 102, 151, 166
CONTRACTED 8, 26, **28**, 29
ControlPoints **54**
COORD 26, **47**, 48, 54, 76, 77, 81, 88, 90, 101, 121, 147, 176, 177, 178
CoordinateType 39, **47**
CoordValue 88, **90**, 91
COUNTERCLOCKWISE 26, **44**, 158, 177, 178

D

DataSection 83, **84**
DATE 26, **45**
DATETIME 26, **45**
DateTimeType 39, **45**
DateTimeValue 88, **89**
DateValue 88, **89**
Dec **25**, 27, 44, 54, 65
DecConst **44**, 45, 60
DEFERRED 26, **30**
DEFINED 26, **66**
Definitions **30**
DeleteObject 85, **86**
DEPENDS 26, **30**, 34, 77, 134, 147, 158
DERIVED 26, **36**, 74
DerivedUnit **60**
Digit **24**, 25
DIRECTED 26, 52, **54**, 72, 103
DOMAIN ... 26, **39**, 40, 41, 42, 43, 44, 46, 47, 48, 54, 74, 76, 77, 100, 102, 121, 151, 164, 166, 176, 177, 178
DomainDef 28, 30, **39**
DomainRef 29, 30, 32, 33, 34, 36, **39**, 41, 44, 45, 48, 54
DrawingRule 75, **76**

E

EmbeddedLink 85, **87**, 98
EmbeddedLinkStruct **87**
END ... 26, **28**, 30, 32, 35, 36, 46, 47, 53, 62, 63, 64, 65, 67, 71, 72, 73, 74, 76, 77, 78, 79, 101, 102, 103, 121, 122,

134, 135, 146, 147, 150, 151, 152, 158, 166, 167, 168, 169, 176, 177, 178, 179, 180, 181, 182
EnumAssignment **76**
EnumElement **41**
Enumeration **41**
EnumerationConst 40, **41**, 76
EnumerationType 39, **41**
EnumRange **76**
ENUMTREEVAL 26, 69, 70, 101
EnumTreeValueType 39, **41**
ENUMVAL 26, 69, 70, 101
EnumValue **88**, 89
EQUAL 26, **73**
EXISTENCE 26, **65**
ExistenceConstraint 63, **65**
Explanation **26**, 28, 60, 69
Expression 39, 65, 66, 67, 74, 76
EXTENDED ... 25, 26, 27, 29, **32**, 33, 36, 38, 46, 62, 71, 74, 76, 78, 79, 102, 122, 152, 166, 167, 181, 182
EXTENDS 26, 27, **29**, 32, 36, 39, 42, 44, 46, 48, 53, 59, 60, 61, 62, 63, 71, 76, 77, 78, 79, 100, 101, 102, 122, 147, 149, 150, 151, 152, 158, 166, 167, 168, 177, 178, 179, 180
EXTERNAL 16, 26, **34**, 36, 38, 147

F

Factor 33, 36, 66, **67**, 69, 74, 75, 76
FINAL 25, 26, 27, **29**, 32, 33, 36, 39, 40, 41, 42, 43, 53, 61, 62, 71, 74, 76, 100, 102
FIRST 26, **67**, 73, 103
Float **25**
FORM 26, **55**, 91, 100
FORMAT 26, **45**, 46, 102, 151, 166
FormatDef **45**
FormationDef 71, **73**
FormattedConst 40, **45**
FormattedType 39, **45**
FormattedValue 88, **89**
FROM 26, **36**, 74
FUNCTION 26, 60, **69**, 70, 71, 74, 101, 150, 151, 152
FunctionCall **67**
FunctionDef 28, 30, **69**

G

GENERIC 26, **38**, **39**, 47, 54, 85
GenericRef **30**
GENERICS 26, **30**
GlobalUniqueness **65**
GRAPHIC 26, 32, **76**, 78, 79, 134, 135, 158
GraphicDef 30, **76**
GraphicRef **76**

H

HALIGNMENT 26, **42**, **43**, 89, 100, 177
HeaderSection 83, **84**
HexDigit **24**, 25
HIDING 26, **36**, 38

I

IMPORTS 26, **28**, 29, 77, 78, 122, 134, 147, 151, 158, 175, 178
IN 26, 54, **65**, 76, 79
INHERITANCE 26, **45**, 46, 102
InnerBoundary **92**
Inspection 67, 68, 72, **73**
INSPECTION 26, 59, 67, 72, **73**, 134
INTERLIS 7, 8, 10, **11**, 12, 13, 14, 15, 17, 18, 19, 20, 21, 26, 27, 29, 31, 38, 40, 45, 55, 59, 61, 62, 73, 76, 100, 103, 121, 122, 134, 143, 144, 146, 149, 151, 152, 154, 156, 157, 161, 165, 166, 177, 178, 211
INTERLIS2Def **27**
IntersectionDef **54**

J

Join **73**
JOIN 26, 72, **73**, 74

L

LAST 26, **67**
Letter **24**
LINE 26, **55**, 91, 100
LineForm **54**
LineFormSegment **91**
LineFormType **54**
LineFormTypeDef 28, **55**
LineType 39, **54**
Link **85**, 86
LIST 8, 9, 26, **33**, 34, 35, 63, 64, 68, 72, 73, 81, 88, 92, 101, 103, 167, 179, 180
LNBASE 26, **44**
LOCAL 26, 64, **65**
LocalUniqueness **65**

M

MANDATORY . 9, 26, **33**, 34, 39, 53, 62, 64, 65, 67, 73, 77, 98, 101, 102, 103, 121, 122, 151, 152, 158, 166, 167, 168, 177, 178, 179, 180, 181, 182
MandatoryConstraint **65**
MetaDataBasketDef 28, 30, **61**
MetaDataBasketRef **61**
METAOBJECT 26, 31, 40, 61, **62**, 76, 92, 101, 158
MetaObjectRef 44, **61**, 76
Model **84**
MODEL .. 26, **28**, 32, 61, 62, 76, 77, 78, 100, 121, 122, 134, 147, 149, 151, 158, 166, 175, 177, 178
ModelDef 27, **28**
MTEXT 26, 40, 70, 88, 101, 113
MTextValue **88**
MULTIAREA 26, **54**, 88
MULTICOORD 26, **47**, 88, 90
MultiCoordValue 88, **90**
MULTIPOLYLINE 26, 52, **54**, 85, 88, 91
MultiPolylineValue 88, **91**
MULTISURFACE 26, **54**, 58, 85, 88, 92
MultiSurfaceValue 88, **92**

N

Name . **24**, 28, 29, 30, 32, 33, 36, 39, 41, 45, 49, 54, 55, 60, 61, 62, 63, 65, 67, 69, 71, 72, 73, 74, 76
NAME 26, 40, 62, 88, 92, 100, 101
NameSpaceDef **83**
NO 26, 31, **32**, 36
NOINCREMENTALTRANSFER 26, **28**
NOT 26, **66**, 152
NULL 26, 72, **73**
Number **25**
NUMERIC .. 26, 39, 44, 47, 53, 54, 62, 63, 70, 76, 101, 102, 152, 166, 177
NumericConst 39, **45**, 90, 91
NumericType 39, **44**, 47, 48
NumericValue 88, **89**

O

Object **85**
OBJECT 26, 69, 101

ObjectOrAttributePath ... 65, **67**
OBJECTS ... 26, 46, **61**, 63, 64, 69, 70, 71, 101, 102, 134, 158, 176
OF 26, **28**, 29, 33, 41, 46, 49, 50, 61, 62, 63, 64, 65, 67, 68, 69, 70, 71, 72, 73, 74, 76, 78, 79, 81, 84, 86, 88, 92, 101, 102, 103, 134, 135, 147, 152, 158, 167, 176, 179, 180
OID ... 17, 18, 20, 21, 26, 29, 30, 31, 32, 34, 36, **48**, 49, 81, 85, 86, 87, 100, 120, 143, 144, 145
OIDAttributeValue ... **93**
OIDType ... 39, **48**
ON ... 26, **30**, 34, 45, 46, 75, 76, 77, 78, 102, 134, 135, 147, 151, 158, 166
OR ... 26, 35, 36, 48, 65, **66**, 67, 72, 73
ORDERED ... 26, 36, 41, 42, 43, 77, 100
OTHERS ... 26, **41**, 88
OuterBoundary ... **92**
OVERLAPS ... 26, 53, 54, 57, 121, 146, 147, 178

P

PARAMETER 26, 32, 62, **63**, 67, 76, 77, 78, 101, 102, 152, 158, 166, 177, 178, 181, 182
ParameterDef ... 32, **62**
PARENT ... 26, **67**, 68
PathEl ... **67**
PI ... 26, **44**, 121, 150, 166, 177, 178
PlausibilityConstraint ... 63, **65**
POLYLINE ... 26, 52, **54**, 72, 81, 85, 88, 91, 103, 121, 177, 178
PolylineValue ... 88, **91**, 92
PosNumber ... **25**, 36, 40, 44, 45, 47, 67
Predicate ... **66**
Projection ... **73**
PROJECTION ... 26, 72, **73**
Properties **25**, 29, 31, 32, 33, 34, 36, 38, 39, 61, 62, 71, 74, 75, 76

R

REFERENCE ... 9, 26, **34**, 92, 179, 180
ReferenceAttr ... **33**, **34**
ReferenceAttribute ... 88, **92**
RefSys ... **44**, 62
REFSYS ... 26, **47**
REFSYSTEM ... 26, **28**, 46, 61, 62, 63, 101, 102, 151, 166, 176
Relation ... **66**
RenamedViewableRef ... 36, **73**, 74
REQUIRED ... 26, 54, **65**
RestrictedClassOrAssRef ... **34**, 36, 67, 69
RestrictedClassOrStructureRef ... **34**
RestrictedStructureRef ... **34**, 35
RESTRICTION ... 26, **34**, 35, 36, 49, 71, 101, 179
Role ... 85, **87**
RoleDef ... **36**
ROTATION ... 26, **46**, 47, 77, 121, 176, 178
RotationDef ... **47**
RunTimeParameterDef ... 28, **63**

S

Scaling ... **25**
SegmentSequence ... **91**
Selection ... 71, **74**, 76
SET ... 26, 64, **65**
SetConstraint ... **65**
SIGN ... 26, **61**, 76, 77, 79, 134, 158
SignParamAssignment ... 75, **76**
StartSegment ... **91**
STRAIGHTS ... 26, 52, **54**, 100, 121, 146, 147, 177, 178
StraightSegment ... **91**
String ... **24**, 28, 40, 45, 47
STRUCTURE . 26, **32**, 46, 49, 53, 62, 64, 70, 72, 73, 81, 88, 90, 101, 102, 103, 151, 166, 179, 180
StructureDef ... 28, 30, **32**
StructureRef ... **32**, 34, 45
StructureValue ... 88, **90**, 91
SUBDIVISION ... 26, **33**, 46, 102, 151, 166
SURFACE ... 26, **54**, 58, 59, 64, 67, 71, 81, 85, 88, 91, 92, 101, 121, 177, 178
SurfaceValue ... 88, **91**, 92
SYMBOLOGY ... 26, **28**, 61, 76, 78, 158, 177, 178

T

Term ... **66**
Term0 ... **66**
Term1 ... **66**
Term2 ... **66**
TEXT ... 26, 40, 49, 64, 70, 71, 77, 88, 100, 101, 121, 147, 166, 167, 169, 176, 177, 178, 179, 180
TextConst ... **39**, 40
TextType ... 39, **40**, 48
TextValue ... **88**
THATAREA ... 26, **67**, 68, 72
THIS ... 26, **67**
THISAREA ... 26, **67**, 68, 72
TIMEOFDAY ... 26, **45**
TimeValue ... 88, **89**
TO ... 9, 26, **34**, 35, 92, 179, 180
TOPIC .. 26, 29, 32, 48, 65, 71, 77, 78, 79, 81, 85, 101, 121, 122, 134, 146, 147, 152, 158, 166, 176, 177, 178
TopicDef ... 28, **29**
TopicRef ... **29**, 30, 61
Transfer ... 80, 81, 82, **83**, 84, 85, 88, 92
TRANSIENT ... 26, 33, **71**, 74, 81
TRANSLATION ... 26, **28**, 29, 84
Type ... 33, 34, **39**
TYPE ... 26, **28**, 100, 149

U

UNDEFINED ... 26, **39**, 64, 66, 71, 73
Union ... **73**
UNION ... 26, 72, **73**
UNIQUE ... 9, 26, 62, 64, **65**, 101, 146, 147
UniqueEl ... **65**, 73
UniquenessConstraint ... 64, **65**
UNIT .. 26, 44, 46, 59, **60**, 100, 102, 121, 149, 166, 177, 178
UnitDef ... 28, 30, **60**
UnitRef ... 44, 45, **60**
UNQUALIFIED ... 26, **28**, 29
URI ... 26, 28, **40**, 88, 100, 143

V

VALIGNMENT ... 26, **42**, **43**, 89, 100, 177
VERSION ... 26, **28**, 100, 121, 122, 134, 147, 149, 151, 158, 166, 175, 177, 178
VERTEX ... 26, **54**, 121, 146, 147, 177, 178
VIEW ... 26, 29, 32, **71**, 74, 81, 85, 134
ViewableRef ... 49, 65, 67, **73**, 76
ViewAttributes ... 71, **74**
ViewDef ... 30, **71**
ViewRef ... 69, **71**, 72

W

WHEN ... 26, **76**, 79
WHERE ... 26, 64, 65, **74**, 75, 76, 79, 134, 135
WITH ... 26, **54**, 121, 146, 147, 177, 178
WITHOUT ... 26, 53, 54, 57, 121, 146, 147, 178

X

XML 8, **49**
XML-Any **82**, 89, 90
XML-base64Binary **82**, 89, 90
XML-ID **83**, 85, 86, 93
XML-NcName **82**, 83
XML-NormalizedString **82**, 83, 88, 89, 90
XMLNS 26, **28**, 29
XML-String **82**, 83, 88

This index lists reserved words in capitals (cf. chapter 2.2.7 Special symbols and reserved words), the syntax definitions of the descriptive language in standard fonts (cf. chapter 2 Decription language) and the syntax definitions of the transfer in italics (cf. chapter 3 Sequential transfer). The page number in bold tells you where to find the most comprehensive definition of a term within this reference manual.

Annex O – References & Bibliography

None

Annex P – Cooperation & Verification

There is an "INTERLIS 2-core team" that has pushed ahead the continuous development from INTERLIS version 2.3 to version 2.4. Its members are Joseph Dorfschmid (Adasys AG), Claude Eisenhut (Eisenhut Informatik AG), Michael Germann (infoGrips GmbH), Stefan Keller (HSR Technical University Rapperswil), Pirmin Kalberer (Sourcepole AG), Hugo Thalmann (a/m/t software service AG) and Rolf Zürcher (KOGIS) in the role of editor and coordinator.

By financing experts and by supplying basic software tools KOGIS has contributed its share in the development of this present version. We are looking forward to the development of further innovative tools and products based upon INTERLIS.

No standard can be laid down by a group of individuals only; on the contrary it needs the help of many specialists. We wish to express our thanks to all these professionals!

Annex Q – Changes in comparison to the previous version

This present version 2.0 of the standard has been revised contents-wise as stated in the preface under « Extensions of INTERLIS 2.4 in Comparison with INTERLIS 2.3». Adaptations concerning the description language (chapter 2) are of minor consequence, whereas considerable changes are to be reported with regard to sequential transfer (chapter 3). Both normative appendices B and C are completely new.

This document underwent a complete *editorial* revision and is now published according to eCH norms. To list all modifications in detail would go beyond the scope of this appendix.

In comparison to the issue of September 8th, 2016 the following changes have been made:

- correction of syntax rules in DomainDef and FormattedType,
- adding three textual changes in chapter 2.5.2, 2.12 and 2.15,
- removal of XML-attribute "SRS" (Spatial Reference System) in the coding of topics
- adjustment of inconsistences between chapter 3.3.4 and annex B (header section of transfer has to be mandatory).

Annex R – Table of figures

Figure 1: Data transfer between several databases via a common data model (data schema) described in a common data description language. 11
Figure 2: Specializing the modeling of a concept from the federal level, to cantonal (country specific) and local level. 12
Figure 3: Inheritance hierarchy of addresses, persons and buildings. 14
Figure 4: Up-dating of a primary database and subsequent transfer to secondary databases (a double arrow means incremental update). 17
Figure 5: Graphic definitions, on one hand built upon data and views, on the other upon symbols permitting generation of graphics (abstract diagram). 19
Figure 6: The various ranges of application of INTERLIS. A double arrow means that data can be incrementally transferred. 20
Figure 7: Roads - a small example. 21
Table 1: Reserved words in INTERLIS 2. 26
Figure 8: Example of an enumeration. 41
Figure 9: Text orientation horizontally (HALIGNMENT) and vertically (VALIGNMENT). 43
Figure 10: Examples of planar curve segments. 50
Figure 11: Examples of planar sets not being curve segments (a double circle indicates "not smooth" and a double square "not injective"). 50
Figure 12: Examples of planar line strings. 52
Figure 13: Examples of planar sets that are not line strings (the double circle means "not continuous" and the double rhombus "not image of an interval"). 52
Figure 14: Examples of (planar) simple line strings. 52
Figure 15: a) Height parameter (of arrow) may not exceed the given tolerance; b) inadmissible overlap of polylines since another vertex is situated between vertex and intersection; c) inadmissible overlap of polylines since there exists no common vertex. 54
Figure 16: Examples of surface elements. 56
Figure 17: Examples of point sets in space, which are not surface elements (here a double circle means "not smooth"). 56
Figure 18: Examples of surfaces in the space. 56
Figure 19: Examples of planar point sets that are not surfaces (a double circle marks a "singular point"). 56
Figure 20: Planar surface with boundaries and enclaves. 57
Figure 21: a) Examples of planar general surfaces; b) Examples of planar sets that are not general surfaces, because their interior is not connected. But these planar sets can be subdivided into general surfaces ("---" shows the subdivision into surface elements and "===" the subdivision into general surfaces). 57
Figure 22: Different possible subdivisions of the boundary of a general surface. 57
Figure 23: Disallowed boundary configurations for tessellations. 58
Figure 24: Individual surfaces (SURFACE). 58
Figure 25: Tessellation (AREA). 58
Table 2: UCS/Unicode-symbols and their coding to be employed by default in INTERLIS 2. 118
Figure 26: UML-class diagram of data models. 120
Figure 27: Graphic generated from graphic and data descriptions. 142
Figure H.1: Suitability of different colorspaces for the purposes of INTERLIS. 155
Figure H.2: The conversion of XYZ to L*a*b*. 155
Figure H.3: Conversion of the cartesian L*a*b*-space to the polar form L*C*abh*ab (according to [Sangwine/Horne, 1998]). 156

Figure H.4: The colorspace L*C*abh*ab functions with polar coordinates onto L*a*b*. 156
Figure H.5: Calculation of color differences in the Cartesian L*a*b*-space......................... 157
Figure H.6: Cartesian and polar coordinates of a color extremely far away from the zero point (conversion see figure H.3). 157
Figure H.7: Cartesian and polar coordinates of some colors. 159
Figure I.1: How to transform the earth surface into 2D horizontal coordinates. 164