INTERNATIONAL STANDARD

ISO 23952

First edition
2020-07

Automation systems and integration — Quality information framework (QIF) — An integrated model for manufacturing quality information

Reference number
ISO 23952:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted (www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by the Digital Metrology Standards Consortium (DMSC) (as ANSI QIF 3.0 - 2018) and drafted in accordance with its editorial rules. It was assigned to Technical Committee ISO/TC 184, Automation systems and integration, Subcommittee SC 4, Industrial data, and adopted under the "fast-track procedure".

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Contents

Figures

TABLES

Foreword

The Digital Metrology Standards Consortium (DMSC, Inc.) is an American National Standards Institute (ANSI) accredited standards developing organization, as well as an A-Liaison to the International Organization for Standardization (ISO) Technical Committee (TC) 184. The mission of the DMSC is to identify urgently needed digital standards in the field of dimensional metrology, and to promote, foster, and encourage the development and interoperability of these standards, along with related and supporting standards that will benefit the industry as a whole. More information about the DMSC can be found at www.dmsc-inc.org.

The Quality Information Framework (QIF) was developed by domain experts from the manufacturing quality community representing a wide variety of industries and quality measurement needs. Contributors to this version 3.0 of the QIF standard include:

- Capvidia
- Datapixel
- Honeywell Federal Manufacturing and Technology
- John Deere
- Metrosage
- Mitutoyo America
- National Institute of Standards and Technology
- Origin International
- QIF Solutions
- The Manufacturing Technology Centre
- University of North Carolina at Charlotte

This document was written by the QIF Working Group, and given final approval for ANSI review by the DMSC's Quality Measurement Standards (QMS) Committee. More information about DMSC's QIF effort can be found at www.qifstandards.org.

The QIF standard, version 3.0, consists of the following subject areas under the general title *Quality Information Framework (QIF) – An Integrated Model for Manufacturing Quality Information*:

Overview and Fundamental Principles

QIF Library Information Model

QIF Model Based Definition (MBD) Information Model

QIF Plans Information Model

QIF Resources Information Model

QIF Rules Information Model

QIF Results Information Model

QIF Statistics Information Model

The Overview and Library clauses describe the overview and central concepts of the QIF standard. The following clauses describe information models for the six *application areas* of QIF; Model Based Design, Plans, Resources, Rules, Results, and Statistics.

This version of QIF, designated ANSI/QIF 3.0 – 2018, is a revision of the previous QIF standard, version 2.1, which was published in 2015. This document comprises the fourth release of the QIF suite of standards, denoted version 3.0. This QIF version 3.0 document cancels and replaces all documents of version 2.1. QIF version 3.0 is solely a product of the DMSC and its committees and working groups.

Each major release of the QIF standard from 1.0 through 2.1 was composed of eight Parts documents, namely, Part 1 Overview, Part 2 QIF Library, Part 3 MBD, Part 4 Plans, Part 5 Resources, Part 6 Rules, Part 7 Results, and Part 8 Statistics. Version 3.0 of QIF dispenses with the separate Parts format and includes all subject area clauses in a single document.

HTML-based data model viewer

The DMSC will make available an html-file based data dictionary for the entire QIF information model as an aid to understanding QIF. This data dictionary is non-normative material, but describes the normative content of the QIF data model. The html files facilitate viewing the complete data model, including all six application areas and Library content, using pictures and text. A user has the ability, through an internet browser, to follow navigation links forward and backward through the data model description using mouse clicks.

Introduction

This Quality Information Framework (QIF) standard defines an integrated set of information models which enable the effective exchange of metrology data throughout the entire manufacturing quality measurement process – from product design to inspection planning to execution to analysis and reporting. This clause of the QIF standard introduces the purpose and design approach behind QIF, as well as its content. The QIF information models are contained in files written in the XML Schema Definition Language (XSDL). Additional constraints to ensure data quality are contained in files written in the EXtensible Stylesheet Transformation language (XSLT). The models for Version 3.0 consist of six application schema files plus a library of schema files containing information items used by all applications. The Library is described in Clause 6 of this standard. Clause 7 defines the Model Based Definition application model which deals with CAD data plus product manufacturing information (PMI). Clause 8 defines the QIF Plans application model, which deals with plans for quality measurement. Clause 9 defines the QIF Resources application model, which describes hardware and software resources used for inspection. Clause 10 defines the QIF Rules application model which describes practices to be used in an inspection. Clause 11 defines the QIF Results application model which deals with the results of part measurements. Clause 12 defines the QIF Statistics application model which conveys analysis of multiple part inspections.

The QIF models include quality characteristics and measurement features as defined in the ASME Geometric Dimensioning and Tolerancing (GD&T) related and ISO Geometrical Product Specifications (GPS) related specifications, and the Dimensional Measuring Interface Standard (DMIS). The QIF standard covers a wide variety of use cases including dimensional metrology inspection, first article inspection, reverse engineering, and discrete quality measurement.

Type face conventions for this document are:

Attribute and element names: **bold**

The words: “attribute”, “element”, “include”, “key”, and “keyref”: *italicized*
when they are formal terms of XSDL.
In their other meanings, those words are not italicized.

Type names: ***bold and italicized***

1 Scope of the integrated model and fundamental principles

1.1 Contents of this document

This clause describes the general content and structure of the entire QIF information model. It describes the highest level data structures of QIF, that are expanded in Clauses 6 through 12 using data dictionaries and XML schema files. All QIF XML schema files can be found at www.qifstandards.org.

This clause also describes practices for forming QIF instance files, called “documents,” that support quality workflow scenarios. Its focus is to show how the QIF information model, and data formed into XML instance files, support the entire scope of model based definition manufacturing quality workflow. It describes how the information model is partitioned among the XML schema files and contains all terms used in the subject area clauses.

The purpose of this clause is to orient potential users of QIF to the organization of the information model to make their study of the details more rewarding and efficient. It should also help solution providers and users to evaluate QIF for their uses, without needing to go to the lowest technical details of the XML schemas. The information model narrative focuses on the approach to modeling the core data structures of QIF, which model the content of ASME GD&T and ISO GPS, and the plans and results data elements defined in Dimensional Measuring Interface Standards (DMSI) ISO 22093 and ANSI/DMSC DMIS 5.3. The material on XML practices describes consistent design practices to be used by QIF working groups who will be designing new schemas. It should also help data processing experts to write software that writes and reads manufacturing quality data using the XML schemas.

1.2 Scope of the QIF Version 3.0 information model

Figure 1 shows a high level view of the QIF information architecture for version 3.0 standardization. At the core of the QIF architecture is the reusable QIF library which contains definitions and components that are referenced by the application areas, thereby ensuring interoperability and extensibility. Around the QIF library core Figure 1 shows the six QIF application area information models, MBD, Plans, Resources, Rules, Results, and Statistics. The DMIS application, which references the DMSC's Dimensional Measuring Interface Standard (DMIS), is a placeholder for a future QIF execution model that is not a part of QIF. Each QIF application model reuses the QIF Library. QIF data is contained in a QIF Document. The order of generation of QIF data in an enterprise generally proceeds clockwise around the diagram, beginning with QIF MBD and ending with QIF Statistics. Use of the QIF information model does not place any requirements on a user’s workflow architecture.

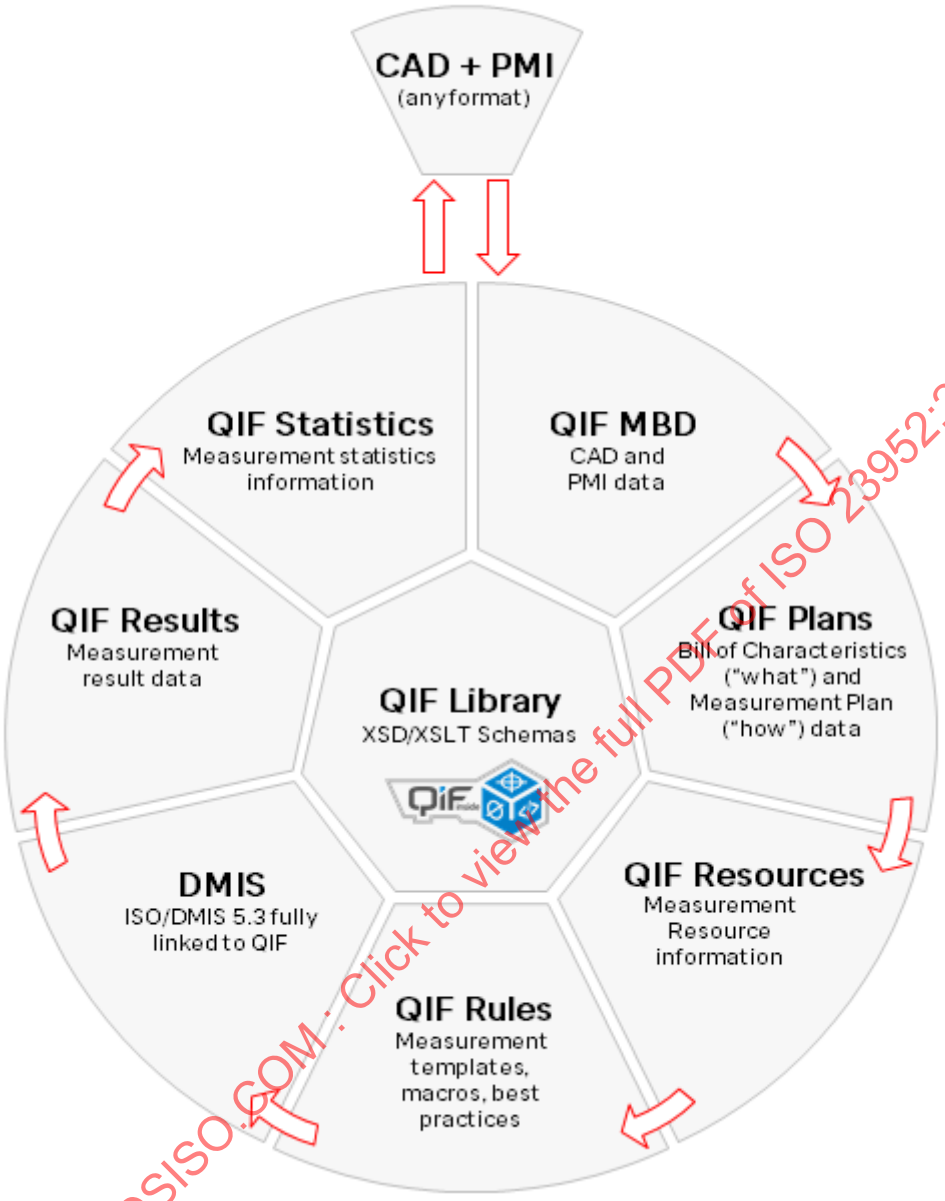


Figure 1 – QIF version 3.0 information architecture

The flow of QIF data typically starts with generation of CAD + PMI data exported as QIF MBD application data. Then Quality planning systems import the MBD and generate measurement requirements as part of the Plans of "what" to measure, then with import of Resources and Rules information, Plans generate an inspection plan on "how" to verify. Then Programming systems import Plans to generate DME-specific programs, or general instructions to guide inspection. Dimensional measurement equipment executes programs and evaluates characteristics of a single manufactured part or assembly and exports the measurements as Results. Analysis systems, typically performing statistical process control, import single parts Results and generate analysis of multiple part batches as QIF Statistics data.

Users of the QIF information model are not required to implement the entire model. Any of the six application models may be used singly for exchange of quality data in its specific application area between a software system that produces QIF data for that area and a software system that

consumes QIF data for the area. Further, other data models and exchange formats can coexist in an enterprise with QIF data.

1.3 Conformance

Software programs that implement this specification to write QIF XML instance files must:

- follow the rules of XML when writing QIFDocument instance files
- generate instance files that validate against the QIFDocument schema
- generate instance files that validate against the Check.xsl XSLT constraints
- employ semantics of the information written that complies with the referenced standards and with the QIF data dictionaries in this specification.

Software programs that implement this specification to read QIF instance files must:

- be able to read any valid QIF XML instance file and extract all numerical and semantic data correctly.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

AIAG MSA Reference Manual 4th Edition, Automotive Industry Action Group *Measurement Systems Analysis*

AIAG SPC – Automotive Industry Action Group *Statistical Process Control* 2nd Edition

ANSI/DMIS 105.3, Part 1-2016, *Dimensional Measuring Interface Standard, DMIS 5.3 Standard, Part 1*

ISO DMIS: ISO 22093:2011 *Industrial automation systems and integration -- Physical device control -- Dimensional Measuring Interface Standard (DMIS)*

ASME B1.7 - 2006, *Screw Threads: Nomenclature, Definitions, and Letter Symbols*

ASME Y14.36 - 1996, *Surface Texture Symbols*

ASME Y14.6 - 2001, *Screw Thread Representation*

ASME Y14.5M-1994 (reaffirmed 2004), *Dimensioning and Tolerancing - Engineering Drawing and Related Documentation Practices*

ASME Y14.5-2009, *Dimensioning and Tolerancing - Engineering Drawing and Related Documentation Practices*

ASME Y14.41-2012, *Digital Product Definition Data Practices*

ASME B89.4.10360.2-2008 *Acceptance Test and Reverification Test for Coordinate Measuring Machines (CMMs) – Part 2: CMMs Used for Measuring Linear Dimensions*

ASME B89.4.22-2004, *Methods for Performance Evalulation of Articulated Arm Coordinate Measuring Machines*

AWS A2.4:2012 Standard Symbols for Welding, Brazing, and Nondestructive Examination

Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation 26 November 2008

ISO/IEC 9834-8:2008. *Information technology -- Open Systems Interconnection -- Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components*

ISO/IEC 11578:1996: *Information technology - Open System Interconnection - Remote Procedure Call (RPC)*

ISO/IEC JCGM 200 – *International vocabulary of metrology – Basic and general concepts and associated terms* (VIM)

ISO/IEC Guide 99:2007 (E/F) – *International vocabulary of metrology – Basic and general concepts and associated terms* (VIM)

SAE Aerospace Standard, AS9102B*: Aerospace First Article Inspection Requirement. 2014.*

ISO 1101:1983 *Technical drawings -- Geometrical tolerancing -- Tolerancing of form, orientation, location and run-out -- Generalities, definitions, symbols, indications on drawings*

ISO 1101:2017 *Geometrical Product Specifications (GPS) -- Geometrical tolerancing -- Tolerances of form, orientation, location and run-out*

ISO 1302:2002 *Geometrical product specifications (GPS) -- Indication of surface texture in technical product documentation*

ISO 5459:2011 *Geometrical Product Specifications (GPS) -- Geometrical tolerancing -- Datums and datum systems*

ISO 14405-1:2016 *Geometrical Product Specifications (GPS) — Dimensional tolerancing — Part 1: Linear sizes*

ISO 14405-2:2016 *Geometrical Product Specifications (GPS) — Dimensional tolerancing — Part 2: Dimensions other than linear sizes*

ISO 14406:2010, *Geometrical product specifications (GPS) – Extraction*

ITU-T X.509 | ISO/IEC 9594-8 (10/2012) *Public-key and attribute certificate frameworks*

RFC 2045 *MIME Part One: Format of Internet Message Bodies*

XML Schema Part 1: Structures Second Edition, W3C Recommendation 28 October 2004

XML Schema Part 2: Datatypes Second Edition, W3C Recommendation 28 October 2004

XSL Transformations (XSLT) Version 2.0, W3C Recommendation 23 January 2007

3 Terms and definitions

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

3.1 Terms defined in ISO 22093:2011 and ANSI/DMIS 105.3-2016, Part 1

3.1.1 **actual**

referring to features or tolerances, the actuals are the real-world, physical instances which can be measured creating feature and tolerance measurements.

Note 1 to entry: Referring to parts and assemblies, the actuals are physical instances called actual components.

3.1.2 **dimensional measuring equipment (DME)**

a class of equipment used to inspect parts and evaluate dimensions and tolerances

Note 1 to entry: DME includes, but is not limited to, coordinate measuring machines, video inspection equipment, optical comparators, robotic measuring devices, theodolites, photogrammetry, laser-based measuring devices, and manual measuring devices such as micrometer and caliper.

3.1.3 **nominal**

a target value of a dimension, feature, or characteristic

Note 1 to entry: A nominal is usually based on as designed engineering criteria.

Note 2 to entry: See also “nominal aspect” at subclause 3.4.11.

3.1.4 **measurement**

referring to a feature, dimension, or tolerance, an approximate representation or value inferred from the actual feature, dimension or tolerance by measurement with DME

Note 1 to entry: See also “measurement aspect” at subclause 3.4.11.

3.1.5 **part coordinate system (PCS)**

a datum reference frame associated with the part to be measured

3.2 Terms defined in XML Schema Part 0: Primer Second Edition, W3C Recommendation 28 October 2004

3.2.1 **attribute**

information represented as an XML attribute in an instance file, usually conforming to an *attribute* declaration in an XML schema

3.2.2 **complexType**

a value type that has elements or attributes

3.2.3 **element**

information represented as an XML element in an instance file, usually conforming to an *element* declaration in an XML schema

3.2.4 **instance file**

a file containing an information set intended to conform to an XML schema

3.3 Terms defined in XML Schema Part 1: Structures Second Edition, W3C Recommendation 28 October 2004

3.3.1 **attribute information item**

information modeled using an *attribute* declaration in an XML schema

Note 1 to entry: often shortened to "attribute." An attribute information item is a field of a data structure. The data type of the field must be a primitive type such as string or number. In an XML instance file, attributes are represented in the form attributeName="attributeValue".

3.3.2 **element information item**

information modeled using an *element* declaration in an XML schema

Note 1 to entry: often shortened to “element.” An *element* information item may be a field of a data structure or an entire data structure. The data type of an *element* may be either a primitive type (such as string or number) or a structured type. In an XML instance file, *element*s are delimited by matching opening and closing tags.

3.3.3 **enumeration**

a term indicating that a single indivisible value follows

3.3.4 **extension**

a term in a complexType definition indicating that the type being defined is derived from a more general type

Note 1 to entry: Extension is the XSDL mechanism used in QIF for building hierarchies of complex types. XSDL also provides for restrictions of complex types, but that is not used in QIF.

3.3.5 **key**

a constraint requiring that selected data fields exist and be unique

Note 1 to entry: In XSDL, a key may require that a combination of data fields be unique, but QIF never uses more than a single data field.

3.3.6 **keyref**

a constraint requiring that there is a match between two sets of values in an instance file

3.3.7 **schema (or XML schema)**

a complete information model in the XML schema definition language

Note 1 to entry: A schema is defined by one or more schema files.

3.3.8 **schema document (or schema file)**

a file containing a well-formed XML schema declaration

Note 1 to entry: Such a file does not necessarily contain a complete information model; it may reference other schema files needed to make the model complete.

3.3.9 **simple type**

a value type that is indivisible or is a list of indivisible items

EXAMPLE number example: 10

EXAMPLE string example: 10 Downing Street

EXAMPLE list example: 10 34 -7

Note 1 to entry: In XSDL, *union* type is also simple, but it is not used in QIF.

3.3.10 **string**

a data type that is a sequence of Unicode characters

3.3.11 **token**

a string in which there is no leading or trailing white space, and the only white space occurring between visible characters is a single space character

3.4 Terms defined in the QIF standard

3.4.1 **accuracy test**

a defined process for determining conformance of a measurement device to specifications

3.4.2 **action**

a plan element that gives information about what to measure or validate

3.4.3 **action group**

a plan element that organizes sets of actions

Note 1 to entry: An action group can be an ordered group, unordered group, one-of group, partially ordered group, or pick-some group.

3.4.4 **action method**

a prescription of how an action is to be performed

Note 1 to entry: Actions with action methods form the core of a measurement plan.

3.4.5 **actual component**

a physical instance of a component

Note 1 to entry: By analogy to a car, the design of a wheel is a *part*; the design of the wheel placed at the front right of the design of a car is a *component*; the front right wheel of a physical car is an *actual component*.

3.4.6 **actual component set**

a set of actual components

3.4.7 **application area**

one of the six QIF workflow interface types, i.e., MBD, Plans, Resources, Rules, Results, and Statistics

3.4.8 **articulating arm CMM**

a coordinate measuring meachine with articulating links

3.4.9 **aspect**

one of four categories used to incrementally accumulate characteristic or feature data based on data source and data shareability

Note 1 to entry: In QIF there are four aspects of characteristics and features: definition, nominal, item, and measured.

3.4.10 **definition aspect**

the aspect of feature or characteristic data categorized by its shareability among different features or characteristics

3.4.11 **nominal aspect**

the aspect of feature or characteristic data particular to an individual feature or characteristic

3.4.12 **item aspect**

the aspect of feature or characteristic incremental data particular to an individual measurement of a feature or characteristic

3.4.13 **measured aspect**

the aspect of feature or characteristic data particular to the results of the measurement of a feature or characteristic

3.4.14 **assembly**

a number of parts or combination thereof that are joined together to perform a specific function and subject to disassembly without degradation of any of the parts

Note 1 to entry: In QIF, assembly means the design of an assembly, and a physical instance of an assembly is called an actual component.

3.4.15 **assembly path**

a sequence with an id of the ids of components

Note 1 to entry: An assembly path shows where in an assembly design a specific instance of a part design or a subassembly design is located

3.4.16 **assignable cause**

a cause of variation in a process which is not random and has some source which can be determined and perhaps eliminated

3.4.17 **attribute characteristic**

a characteristic described using attribute data, or using data that do not have numerical values

EXAMPLE color, malleability

3.4.18 **attribute data**

a result from a characteristic or property that is appraised only as to whether it does or does not conform to a given requirement (for example, go/no-go, accept/reject, pass/fail, etc.)
[1]

3.4.19 **autocollimator**

an optical instrument for non-contact measurement of angles

3.4.20 **bias**

a measure of a gage's tendency toward specific values when compared to a master value

3.4.21 **bill of characteristics (BoC)**

a list of all the characteristics applied to a product.

3.4.22 **boolean condition**

a statement which can be unambiguously evaluated as true or false

EXAMPLE "The feature is a cylinder" would evaluate to "true" if and only if the feature in question is a cylinder feature.

3.4.23 **calibration**

a process that verifies instrument performance or develops correction values to assure claimed accuracy

3.4.24 **caliper**

a device used to measure distance between two points on a workpiece

3.4.25 **capability**

a measure of a process's stability and centralization against a nominal value and tolerance values, also, the ability of a process to produce acceptable parts

3.4.26 **capacitive sensor**

a non-contact instrument that uses change in capacitance to measure displacement

3.4.27 **carriage**

a mechanical element of a measurement device, generally a CMM, that functions as part of the tool-carrying structure

3.4.28 **Cartesian CMM**

a coordinate measuring machine with orthogonal axes

3.4.29 **characteristic**

a control placed on an element of a feature such as its size, location or form, which may be a specification limit, a nominal with tolerance, a feature control frame, or some other numerical or non-numerical control

3.4.30 **characteristic item**

a tolerance or specification applied to a feature or product that needs verification

3.4.31 **charge coupled device camera sensor**

a sensor that senses by means of using a charge coupled device camera

3.4.32 **checked**

refers to a measured feature or characteristic being measured directly or being constructed from previously measured or constructed data

3.4.33 **clipping plane**

a bounding plane surface which abbreviates the intended display of data to that portion which lies on one or the other side of the plane

3.4.34 **complex tactile probe sensor**

a touch probe with more than one stylus

3.4.35 **component**

an instance of a part or an assembly aligned to its parent's space

3.4.36 **composite feature**

a feature composed of two or more sub-features which act as a functional group and to which shared characteristics may be applied

3.4.37 **computed tomography**

an imaging procedure that uses special x-ray equipment to create detailed pictures, or scans, of areas inside the part.

Note 1 to entry: It is also called computerized tomography.

3.4.38 **confocal chromatic sensor**

an optical sensor that takes advantage of chromatic aberration to carry out a chromatically coded distance detection of a target form the focusing lens

3.4.39 **constructed feature**

a feature that is computed from other features

Note 1 to entry: Contrast to a measured feature, which is computed from measured point data.

3.4.40 **control limits**

statistical limits that represent boundary conditions of a process that is in control

Note 1 to entry: Processes that are out of control are said to have unnatural causes or assignable causes of variation.

3.4.41 **control points**

a set of points that determine the shape of a NURBS object

3.4.42 **coordinate measuring machine**

a machine that measures a part by sensing 3D points on the surface of the part

3.4.43 **control polygon**

a polygon formed by the control points of a NURBS object

3.4.44 **corrective action**

a countermeasure that can be applied to an assignable cause of variation in order to reduce the likelihood of recurrence

3.4.45 **corrective action plan**

information related to the establishment of lists of assignable causes of variation and associated corrective actions in the manufacturing process

Note 1 to entry: In production statistical studies, a key aspect of manufacturing process control is the identification of assignable causes of variation that can occur. These special causes of variation are typically associated with corrective action plans that help the manufacturing operator with the ability to adjust processes to eliminate these unnatural causes of variation from occurring again thereby bringing the manufacturing process into a greater state of stability and control.

3.4.46 **data/information quality**

an assessment of the fitness of the data and information to serve its purpose in a given context

3.4.47 **datum definition**

definition of a datum label and optionally, its association with datum targets or feature instances

3.4.48 **datum reference frame**

three mutually perpendicular intersecting datum planes

Note 1 to entry: A datum reference frame establishes a coordinate system.

3.4.49 **dial caliper**

a caliper that has a dial that is read by a human

3.4.50 **digital caliper**

a caliper that is read electronically

3.4.51 **digital micrometer**

a micrometer that is read electronically

3.4.52 **draw wire sensor**

an instrument that measures distance by recording the length of wire drawn out of it using potientiometers or encoders

3.4.53 **DVRT sensor**

A system that detects the position of its core by measuring the differential reluctance in two coils surrounding the core

3.4.54 **eddy current sensor**

a non-contact sensing probe that uses magnetic field interaction and subsequent generation of eddy currents to measure displacement of a target

3.4.55 **entity**

the basic unit of information in a file

Note 1 to entry: The term applies to single items which may be individual elements of geometry, collections of annotations to form dimensions, or collections of entities to form structured entities.

3.4.56 **evaluation**

refers to the process by which the status of a characteristic measurement is determined from nominal, measured and specification limit information

3.4.57 **event**

an occurrence, usually unplanned, which may have an effect on the outcome of a measurement or inspection operation and which should be recorded

3.4.58 **external product definition**

a product shape definition not included in a QIFDocument

EXAMPLE a PDF file, a drawing, or a physical part.

3.4.59 **feature**

an aspect of a part which can be (1) a tangible portion of a physical part, (2) an element of a technical drawing, 3D model, or other abstract representation of the part, or (3) a derived, constructed, intangible portion of a part, such as a feature of size, minimum circumscribed sphere, maximum inscribed cylinder, axis, center plane, theoretically extended edge on a technical drawing, and projected point/line from the part.

3.4.60 **file unit**

a unit defined in the file units section of a QIF instance file

EXAMPLE radian, meter, inch

3.4.61 **fixture**

a device used for workpiece positioning and holding

3.4.62 **gage**

a measurement device that is used to measure one or more characteristics of a workpiece

Note 1 to entry: It may be a general purpose device, e.g. a ring gage, or it may be specially designed hardware dedicated to measuring a specific workpiece or family of workpieces.

3.4.63 **gage repeatability and reproduceability (gage R&R)**

a source of measurement variation based on the ability of a gage to repeat measurements on the same parts and the ability of two or more inspectors using this gage to achieve the same measurement results

3.4.64 **generatrix**

a curve to be swept to generate an extruded surface, or revolved to generate a surface of revolution

3.4.65 **generic feature**

a feature which can be referenced by a user-defined characteristic

Note 1 to entry: Usually, this is a portion of the surface of a part that is not correctly described by any other feature type.

3.4.66 **geometric**

related to shape information

EXAMPLE points, curves, surfaces, and volumes

3.4.67 **geometric characteristic**

a concept characterizing the size, form, orientation or location of a feature or of a component of a feature

EXAMPLE diameter, flatness, parallelism, or position.

3.4.68 **inspection**

a measurement of characteristics on a physical part to determine whether the features are within allowed tolerance, commonly in order to accept or reject the part

3.4.69 **inspection traceability**

information about the circumstances of a quality measurement process

3.4.70 **internal product definition**

a product shape definition contained in a QIFDocument

3.4.71 **item**

an aspect of a characteristic or feature, or an individual instance of an object.

3.4.72 **key characteristic**

a characteristic of a feature, material, process, or part whose variation has a significant influence on product fit/function, safety/compliance, performance, service life, or manufacturability

Note 1 to entry: A key characteristic can be identified by a designator and can have a criticality class.

3.4.73 **knot vector**

an increasing sequence of real numbers which divides the parametric space of a NURBS into intervals (called spans)

Note 1 to entry: The number of knots is equal to the number of control points plus the order. The knot vector is a uniform one if all knots are equally spaced and of multiplicity one.

3.4.74 **laser radar**

a large-scale-metrology instrument that measures the coordinates of a point on a part without using an optical target by directing a frequency modulated laser beam and heterodyne detection of the returned beam

3.4.75 **laser tracker**

an instrument that is typically used to measure large workpieces by determining the positions of optical targets held against the workpiece

3.4.76 **laser triangulation sensor**

a laser source and detector system that uses triangulation principle for non-contact measurement of displacement of a target

3.4.77 **light pen CMM**

a coordinate measuring machine that uses a light pen for measuring

3.4.78 **linearity**

the ability of a gage to accurately measure across a range of values

3.4.79 **Long Term Archiving and Retrieval**

defining and storing digital information models so that the information can be fully retrieved and the item corresponding to the model can be fully realized at any future time, even if the systems used to define and store the information are then obsolete

3.4.80 **LVDT sensor**

A non-contact sensor that uses electromagnetic coupling to measure the linear displacement of a core from its central or null position

3.4.81 **magneto-inductive sensor**

a non-contact positioning sensor that combines magnetic and inductive principles to determine the absolute distance to a permanent magnet fixed to a target

3.4.82 **manufacturing traceability**

information about the circumstances of a manufacturing process.

3.4.83 **measurand**

quantity intended to be measured

Note 1 to entry: An object, quantity, property, or condition to be measured for a specific purpose.

Note 2 to entry: Two examples of a measurand are the measurement of a shape feature to evaluate a specified characteristic (for example tolerance) and the measurement of a shape feature to establish a datum (for example primary datum) within the context of a datum reference frame. One could measure the same feature differently or apply a different substitute feature data fitting algorithm..

3.4.84 **measure feature method**

an action method for measuring a feature

3.4.85 **measurement**

an estimate of a dimension associated with a feature or features on a physical part generated using a physical device

3.4.86 **measurement device**

a measurement resource that provides an indication for a dimensional characteristic of a workpiece

Note 1 to entry: This indication may either be a value (variables) or simply conformance (attributes).

3.4.87 **measurement plan**

a complete plan that contains information on what and how to measure

3.4.88 **measurement resource**

a piece of dimensional metrology equipment that can be used to measure features and/or characteristics on a workpiece

3.4.89 **measurement result**

the measured value of an actual feature or characteristic

Note 1 to entry: A measurement result is obtained by measuring a feature or characteristic with a measurement resource.

3.4.90 **measurement room**

an environmentally controlled room in which measurements are made

3.4.91 **mesh**

a collection of vertices, edges, and facets linked together to form a 3D shape; the facets are typically triangular in shape, but quadrilateral representations are also possible

3.4.92 **micrometer**

a mechanical device for measuring very small distances based on the rotation of a finely threaded screw

3.4.93 **microscope**

an optical instrument used for viewing very small workpieces typically magnified several hundred times

3.4.94 **multiple carriage CMM**

a coordinate measuring machine with two or more carriages

3.4.95 **non-dimensional quality data**

data expressed as either attributes or variables, e.g., number of non-conformities like burrs or dents, color of paint, etc.

3.4.96 **normal**

perpendicular to a surface and pointing away from the surface material

3.4.97 **normal vector**

a unit vector perpendicular to a surface in 3 dimensions or a unit vector perpendicular to a curve in 2 dimensions

3.4.98 **notable event**

a description of a planned or unplanned event that inspection processes should monitor and should record if it occurs during inspection

3.4.99 **note**

an explanatory or descriptive statement in natural language

Note 1 to entry: In QIF, notes provide information in addition to that which is formally modeled.

3.4.100 **noted event**

an event occurring and reported during inspection

3.4.101 **optical comparator**

an optical instrument utilized for measuring

Note 1 to entry: It is also referred to as a profile projector or shadowgraph

3.4.102 **parallel link CMM**

a coordinate measuring with parallel links (e.g., Renishaw equator)

3.4.103 **part**

one item, or two or more items joined together, that is not normally subject to disassembly without destruction or impairment of designed use

Note 1 to entry: In QIF, part means the design of a part, and a physical instance of a part is called an actual component.

3.4.104 **persistent identifier**

a permanent, location and instance independent identifier for a digital object

3.4.105 **plan element**

an action or an action group

Note 1 to entry: A combination of actions and action groups can be structured in a directed hierarchical tree of actions.

3.4.106 **plan note**

descriptive information that applies to an entire measurement plan

3.4.107 **plan root**

the top level plan element (an action or action group) of a measurement plan

3.4.108 **point sampling strategy**

the geometric pattern that is used to distribute the measurement points on a given feature

Note 1 to entry: The range of possible values is taken from ISO-14406:2010. Examples include: an orthogonal grid, a helix, a “birdcage“, etc.

3.4.109 **process variation**

the variation of a process in achieving the same characteristic values across time

3.4.110 **product**

a generic term for a part or an assembly

3.4.111 **product and manufacturing information (PMI)**

non-geometric attributes in 3D Computer Aided Design/Manufacturing/Inspection/Engineering (CAD/CAM/CAI/CAE) systems necessary for manufacturing product components or subsystems

Note 1 to entry: PMI may include geometric dimensions & tolerances (GD&T), 3D annotation (text) and dimensions, surface finish, and material specifications.

3.4.112 **production**

a manufacturing process or operation designed to produce goods

3.4.113 **QIF persistent identifier (QPId)**

a standard universally unique identifier used in QIF

3.4.114 **qualification**

a check on or the refinement of the calibration of a measurement device performed using a known artifact such as a tooling ball or gage block

3.4.115 **rule**

in the context of QIF Rules, a pair usually consisting of a Boolean condition and a specification of the type of measurement activity that should be carried out if the condition evaluates to true

Note 1 to entry: The Boolean condition is actually optional so that unconditional rules (i.e., actions that must or may always be taken) may be written.

Note 2 to entry: For example, if “the feature is a cylinder” evaluates to true, the corresponding action may be requested: “measure 13 points”.

3.4.116 **sampling category**

a number representing a category of rules use

Note 1 to entry: The exact meaning of each category and the number of possible levels is meant to be defined and evaluated by the user. For example, the number 1 might represent first article inspection and the number 2 might represent process control.

3.4.117 **sampling method**

a statistical method of grouping manufactured products for measurement

Note 1 to entry: Key concepts include sample size (how many) and sampling frequency (how often).

3.4.118 **sensor**

a measurement resource that detects a surface or measurement feature of a workpiece

3.4.119 **sensor qualification**

the process of measuring a reference artifact to determine sensor attributes

Note 1 to entry: an example is measuring a reference sphere to determine stylus tip diameters and probe tip offsets for a touch probe on a CMM

3.4.120 **set**

refers to a measured feature or characteristic being set to its nominal without any measurement taking place

3.4.121 **simple tactile probe**

a touch probe with a single stylus

3.4.122 **sine bar**

a hardened, precision ground body with two precision ground cylinders used for measuring angles

3.4.123 **stability**

the ability of a gage to arrive at the same measurements against a master value over time

3.4.124 **standard deviation**

a measure of the dispersion or frequency distribution around a population mean or average

3.4.125 **statistical study plan**

information, defined in the QIF Statistics model, that provides a method for establishing the quality approach and criteria for data studies associated with measurement in manufacturing

Note 1 to entry: Statistical study plans provide a method for establishing the quality approach and criteria for data studies associated with measurement in manufacturing. A supplier may be provided with these plans from a customer in order to ensure the expectation that correct quality control information will be supplied with each manufactured lot.

Note 2 to entry: Statistical sStudy plans can contain the number of expected products to be inspected. They can also contain the expected quality conformance thresholds such as the characteristics to be measured and monitored, sample size, process capability, and gage repeatability and reproduceability requirements.

3.4.126 **statistical study results**

information that contains the measurement summary data that provides an overview of measured product quality

Note 1 to entry: Statistical study results contain the measurement summary data that provides an overview of product quality. From single first article inspection, to capability and production studies, statistical study information can include the measured process capability and performance quality indices that are commonly accepted in manufacturing.

3.4.127 **structured light sensor**

a non-contact optical measurement system that projects a known fringe pattern on a surface to characterize it using the distorted pattern recorded by the detector

3.4.128 **tactile probe sensor**

synonym for touch probe

3.4.129 **theodolite**

an instrument for measuring angles in the horizontal and vertical planes

3.4.130 **thread specification**

a description of the shape and size of a thread

Note 1 to entry: This is thread in the sense of the grooved/ridged portion of a nut or bolt.

3.4.131 **tolerance**

an upper and lower specification of a characteristic dimension

Note 1 to entry: A tolerance usually provides the allowable range for part acceptance based on designed engineering criteria for fit and function.

3.4.132 **tool**

a measurement resource that mounts one or more sensors to a measurement device

3.4.133 **touch probe**

a sensor that is actuated by physical contact with a workpiece

3.4.134 **traceability**

information about the circumstances of a quality measurement process or a manufacturing process

Note 1 to entry: QIF defines five types of traceability. None of these is the classical traceability of measurements from an international standard through intermediate steps to a specific measurement.

3.4.135 **trimming contour**

a 3D contour which is used for trimming a surface

3.4.136 **ultrasonic sensor**

a proximity sensor that uses time of flight of sound waves to determine the absolute distance to a target

3.4.137 **universal length measuring machine**

a one dimensional measuring device used in precision applications

3.4.138 **version**

a unique, non-trivial instance of a document or schema

3.4.139 **weld characteristic**

a characteristic that indicates a desired type of weld

3.4.140 **weld symbol**

a pictorial representation of a weld characteristic

3.4.141 **wire-frame**

a method of geometric modeling in which a two- or three-dimensional object is represented by object edges

3.4.142 **work instruction**

information that provides instructions about actions or action methods to be used in executing a measurement plan

3.4.143 **workpiece**

a physical artifact, real or virtual, intended for subsequent transformation within some manufacturing operation

4 Symbols and abbreviated terms

ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
ASME	American Society of Mechanical Engineers
BREP	Boundary Representation
CAD	Computer-Aided Design
CAIPP	Computer-Aided Inspection Process Planning
CAM	Computer-Aided Machining or Computer-Aided Manufacturing
CAX	Computer-Aided Technologies
CMM	Coordinate Measuring Machine
CoP	Cloud of Points
COTS	Commercial Off-The-Shelf
DME	Dimensional Measuring Equipment
DMIS	Dimensional Measuring Interface Standard
DMSC	Digital Metrology Standards Consortium
DRF	Datum Reference Frame
ERP	Enterprise Resource Planning
GD&T	Geometric Dimensioning and Tolerancing
GPS	Geometrical Product Specifications
GUID	Globally Unique Identifier
ISO	International Organization for Standardization
LOTAR	Long Term Archiving and Retrieval
MBD	Model Based Definition
MES	Manufacturing Execution Systems
MRI	Measurement Resources Information
MRP	Materials Resource Planning
MSA	Measurement Systems Analysis
PDPMI	Product Definition with Product and Manufacturing Information

PMI	Product and Manufacturing Information
QIF	Quality Information Framework
QMS	Quality Measurement Standards (a DMSC committee)
QPId	QIF Persistent Identifier
R&R	Repeatability and Reproducibility
SI	The International Systems of Units
SPC	Statistical Process Control
SQC	Statistical Quality Control
STEP	Standard for the Exchange of Product model data (ISO 10303)
UUID	Universally Unique Identifier
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
XSDL	XML Schema Definition Language
XSLT	eXtensible Stylesheet Language Transformation

5 Overview of the Quality Information Framework (QIF) information model

5.1 Purpose

The goal of the QIF specification is to facilitate interoperability of manufacturing quality data between system software components. Solving the metrology interoperability problem will benefit manufacturers by avoiding wasted resources spent on non-value-added costs of translating data between the different components of manufacturing quality systems. Users should gain flexibility in configuring quality systems and in choosing commercial components, and achieve effortless and accurate flow of data within their factory walls as well as with suppliers and customers. Solution providers should be able to eliminate their efforts previously spent in data translations, and there should be increased opportunities to sell their products and to improve and expand the features of their solutions.

System wide interoperability is achieved by partitioning the information model between a QIF Library of common, reusable components, and six information models for unique application areas. The reusable library components are referenced throughout the comprehensive quality information model thereby ensuring interoperability and extensibility between any data producer and consumer that implements the QIF formats in their software.

DMSC's goal is to write the QIF specifications such that conformity of commercial software products can be assessed by a manufacturer or supplier (first party), a user or purchaser (second party), or an independent body (third party). The ability of developers to test against conformance criteria, and of users to evaluate products for conformance, are key to establishing widespread interoperability of commercial off the shelf software solutions.

5.2 Model based definition manufacturing quality workflow

QIF includes information models for six application areas, as shown in Figure 1. Figure 2 shows a model-based quality workflow activity diagram and the use of QIF formats to convey information between computer-aided quality processes. The scope of QIF is all manufacturing design and quality information required to assess product quality. QIF is also intended to improve manufacturing processes and product design. The work flow model shows five major activities of the quality measurement process:

- Define Product
- Determine Measurement Requirements
- Define Measurement Process
- Execute Measurement Process
- Analyze & Report Quality Data

Activities export and/or import quality information formatted according to the QIF information model and XML encoding rules. The diagram does not show activities that generate manufacturing process information or that implement a manufacturing execution system. The features of QIF facilitate efficient flow of enterprise quality data in a way that does not specify or constrain a user's system architecture.

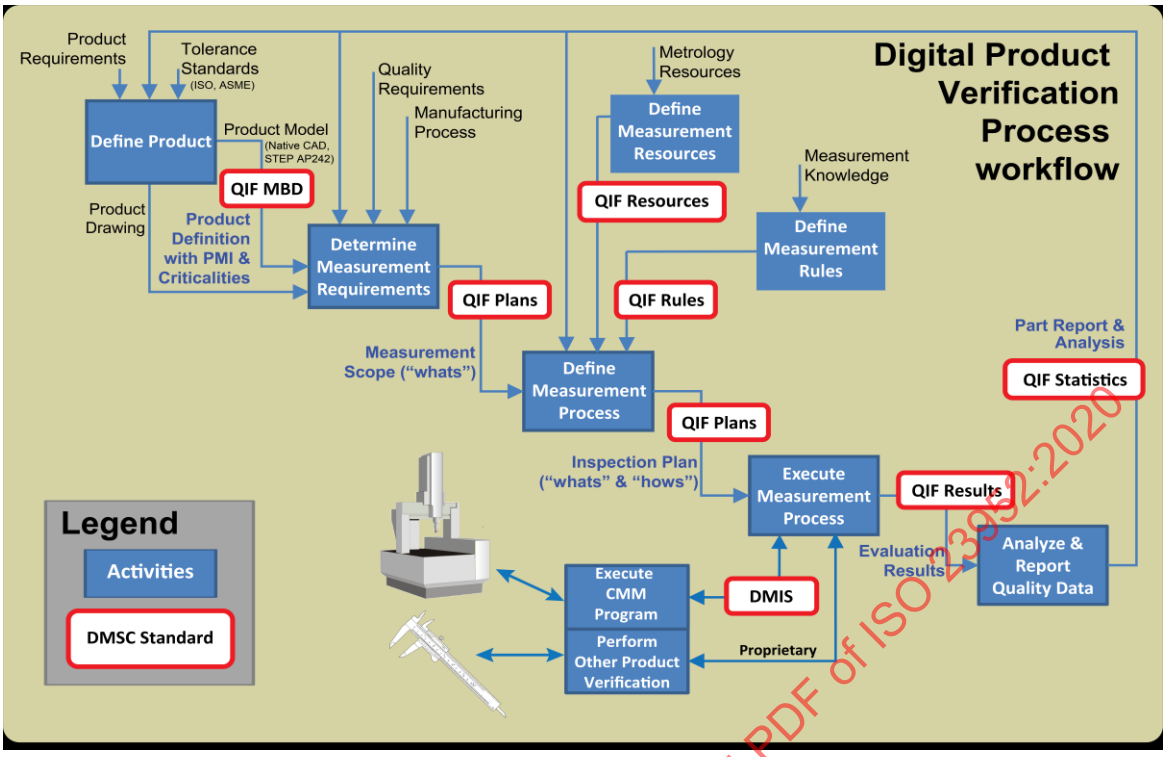


Figure 2 – QIF Model-Based Quality Workflow

The *Define Product* activity generates a model-based definition of a part that can support an enterprise's digital product verification. The product definition contains geometry information plus semantically linked product manufacturing information (PMI). PMI commonly includes geometric and dimensional tolerancing information, product characteristics with criticalities, material, surface texture, roughness, color and hardness. PMI typically includes associations between geometric elements of the product and dimensions, tolerances, and functional datums. The product definition is expressed using the QIF MBD information model.

The *Determine Measurement Requirements* activity imports QIF MBD information or its equivalent expressed in another format or formats. Based upon enterprise quality requirements and/or manufacturing process knowledge, measurement requirements for a part are generated as a set of measurement criteria also known as a bill of characteristic items (BOC). A characteristic item is typically a tolerance or specification applied to a feature or product that needs verification. The plan may also specify the measuring sequence and resources to be used, but is not required to do so. This BOC constitutes a high-level quality plan of "what" needs to be inspected or verified, expressed as a QIF Plans (whats) information packet.

The activity that generates QIF Resources information is *Define Measurement Resources*. This activity defines enterprise hardware and software resources available, either serial-number specific, or generic, that can be harnessed to meet inspection requirements for individual part features. The QIF Resources data format can be used to specify resources required in an inspection plan, or the resources actually used in an inspection. The scope of the QIF Resources application model includes but is not limited to hardware, software, and human operators.

The *Define Measurement Rules* activity generates a QIF instance file that conforms to the QIF Rules information model and specifies inspection practices required by an enterprise to be used in-house or by contractors. Such a file defines, for some or all feature types on a part, information

elements that constrain the measurement on that feature type. Measurement constraints may be applied to things like measurement point density, measurement point pattern, and feature fitting algorithm. Rules may be written for every QIF feature type. The QIF standard defines the generic format to express enterprise Rules, but does not contain specific rules. The instance file generated by the enterprise contains the specific rules. Information areas that are in scope include:

- product measurement point number or density and sampling method for each product feature type
- feature fitting algorithm for each feature type
- selection of measurement resources
- rule ID and corporate ownership.

The *Define Measurement Process* activity inputs resource and metrology knowledge, and the QIF Plans (what), and generates additional instructions on “how” to inspect or verify the bill of characteristic items. The completed inspection plan is output as QIF Plans (whats and hows). The scope of the QIF Plans 3.0 application model includes:

- dimensional product information, e.g., geometric features, measurement features, nominal dimensions, measurement features, and tolerance values
- non-dimensional product information, e.g., product IDs, customer information, key contact, temperature, and roughness
- product characteristics
- traceability values and pointers
- work instructions
- measurement resources to be used
- CAD entity relationships.

The downstream activity *Execute Measurement Process* activity imports the QIF Plan, and if needed, generates a detailed resource specific inspection program. The programs are machine-level measurement programs, formatted according to DMIS or some other measurement programming language, that provide equipment level commands to specific coordinate measuring machine (CMM) control units, to collect point data, fit features to data, and output feature and characteristic data. The workflow shows the export of non-QIF format subsequently translated according to the QIF Results information model. Measurement processes that adopt QIF will likely export results directly without translation. Measurement Execution can also include software solutions that issue instructions to human operators using calipers, go/no-go gauges, and specialized inspection equipment, and generate results data. Actual measurement values may be numerical or non-numerical. Measurement results may include not only raw measurement values, but also summary statistical or derived results (e.g., cylinder radius with standard deviation). Measurement results may also include description of the algorithmic means (e.g., least squares) by which the derived results are calculated. All necessary nominal (as designed) target values may also be included to allow reanalysis. Any other information relevant to the measurements is also in scope. This includes information called inspection traceability, which includes the shift, the equipment operator’s name, a description of the item measured, the date and time of the measurement, etc.

Finally, the measurement results for two or more parts are collected, analyzed, and reported by the activity *Analyze & Report Quality Data*. The output, expressed using the QIF Statistics model, is generally an analysis of a multi-part batch. QIF Statistics is designed to carry information to transport statistical quality control plans, corrective action plans and detailed summary quality

statistics. It builds on the QIF Results framework through supporting multi-part measurement results that can apply to a number of quality study types beyond single or first article inspection. It is designed to haul information in an unambiguous form for pre-production, capability, and production quality studies. In addition it supports the full extent of measurement systems analysis studies including Gage R&R.

Quality information generated in QIF format can be used as input by many other quality and manufacturing management components not shown in Figure 2, including, but not limited to, first article inspection plan and report (FAIR) generation, statistical process control (SPC), materials resource planning (MRP), measurement systems analysis (MSA), manufacturing execution systems (MES), and computer aided manufacturing (CAM).

The digital interface between Execute Measurement Process and the DME (dimensional measurement equipment) has been satisfied by the Dimensional Measuring Interface Standard (DMIS), ANSI/DMIS 105.3 Part 1-2016. DMIS can also be used as a numerical control part program for DMEs such as coordinate measuring machines (CMM).

An example of user-defined flow of QIF information not shown in Figure 2 is the direct use by a human inspector of a QIF Plans file to inspect a part. The operator makes the decisions of Measurement Programming, like selecting the inspection device (e.g., caliper, fixtures, go / no-go gages) and generating the inspection instruction details (e.g., number of points, placement of measurements).

5.3 QIF design requirements

There are four categories of requirements on the content of the QIF information model, and on how the data are encoded for exchange:

- **Business case functional requirements**. The design of the QIF information model is driven by the functional requirements of activities that import, process, and export manufacturing quality data. Requirements are expressed via natural language rules, scenarios, use-case notation, identification of specifications and/or standards, and examples. The DMSC began with a baseline requirement to model ANSI/ASME Y14.5-1994, and the plans and results information in DMIS 5.2. QIF 3.0 extends its coverage of ISO GPS. The requirements list is evolving as the members of the quality community identify more workflow requirements. Functional requirements involve engineering data and workflow details, as well as business case justifications and requirements.

- **Interoperability requirements**. This is primarily the requirement on the QIF standard that developers from different companies should be enabled, without cooperation, to develop software applications that will successfully exchange QIF data packages. This requirement is essential because solutions are developed globally by diverse developers, but also because QIF information must flow between companies, between original equipment manufacturers and contractors, and between primary vendors and their subcontractors. Interoperability requirements are met primarily through complete and accurate semantic annotations embedded within the XML schema files, and by publishing specifications and usage documents. QIF also includes data structures that allow writers of instance files to insert customized data, which should only be used when their data does not match a defined QIF data type.

- **Computational requirements**. These are good practices of computer science that lead to efficient code and efficient use of computation resources. Examples include the use of class inheritance, and schema design practices that minimize the size of instance files.

- **Data/information quality requirements.** The meaning of many information elements in the QIF standard is preserved by constraints and validation tools provided by XML schema language and XSLT stylesheet language transformations, including redundancy checks, data elements to store the results of product data quality checks, and a digital signature with a hash value to detect unauthorized editing or accidental corruption of a QIF document.

5.4 QIF Data Quality

QIF data quality refers to the state wherein all data/information in the QIF standard are defined correctly, completely, and unambiguously and wherein all implementations of the QIF standard employ and interpret the information as explicitly defined in the QIF standard.

The QIF standard is not only a data structure but also a set of formal rules for data interaction with that structure. By virtue of its implementation in XML schema language and standard checks available in the XSLT language standard, QIF transfers much of the burden of data/information quality from the application implementer to the standard itself. If a QIF instance file passes all XML and XSLT software checks, it is valid to the QIF standard in all syntactical and many semantic elements.

5.4.1 XML implementation

The XML implementation of the QIF data model provides a number of benefits to QIF users.

First, the QIF data model is implemented in XML Schema Definition Language which allows for automatic syntax and consistency checking of the standard itself, using widely available free or low-cost software tools.

Second, software implementations of the standard can take advantage of free or low-cost automatic source code generation tools. These tools provide computer language code which automatically consumes and produces QIF document instance files in XML format consistent with the syntax of the standard.

Third, QIF document instance files in XML format can be checked against the standard schemas with free or low-cost automatic tools. These tools will check that the structure of an instance file conforms to the QIF model. They will also check the validity of internal references in a file using *key*s and *keyref*s as described in subclause 5.12.

Fourth, checks of the quality of information in QIF document instance files are available through the use of XSLT (Extensible Stylesheet Language Transformations) language checks. The XSLT checks are a normative part of the QIF specification. The XSLT checks can be performed by a number of widely available free tools (and by low-cost tools). The XSLT checks address both the integrity and semantics of individual instance files and the validity of references between linked instance files. Moreover, when there is tree of linked instance files, applying the XSLT checks to the file at the root of the tree results in the checks being applied to all files in the tree. Applying the XSLT checks results in an error report being generated as an XML file.

The files that contain the XSLT checks are arranged in a small tree of six files with Check.xsl at the root. Check.xsl imports CheckDocuments.xsd, CheckFormat.xsl, CheckQuality.xsl, and

CheckSemantic.xsl. Each of those four imported files imports CheckLibrary.xsl. The Check.xsl file triggers the use of the CheckDocuments.xsd file, which triggers the use of the other three.

The XSLT checks are parameterized. For example, the tolerance on the number of segments in a polyline is given by the MaxNumSegments parameter. Values of the parameters are set in the CheckParameters.xml file, which is accessed by the CheckLibrary.xsl file. A user may reset the parameters by editing CheckParameters.xml. Processing of linked instance files may be activated by the parameter CheckLinkedDocuments in CheckParamaters.xml, where MaxRecursionLevel defines the maximum recursion level. Format, quality and semantic checks may be activated by parameters CheckFormat, CheckQuality and CheckSemantic.

The XSLT checks in QIF are shown in Table 1.

Table 1 – XSLT Checks

XSLT Check	Category	Purpose
Maximum recursion level	General	The recursion level of linked documents must not exceed the specified level.
Number of *element*s	Format	The number of *element*s in sets/arrays must be equal to the value specified in the attribute 'n'.
Maximal **id** of model objects	Format	The object **id**s must be less than or equal to idMax.
Link to an external QIF document	Format	The specified external QIF document must be found and its QPId must match the one given.
Link to an external object	Format	The specified entity must be found in an external document.
Link to a specific type of external object	Format	The specified entity in an external document must be the right type of entity (over 300 cases).
NURBS curve	Format	The number of control points must be equal to the number of knots minus the curve order.
NURBS surface	Format	The number of control points must be equal to ('nKtU' - 'ordU')*('nKtV' - 'ordV').
Feature nominal	Semantic	Feature nominals must be connected to topology entities.
Position characteristic definition	Semantic	If the ToleranceValue in a Position characteristic definition is 0 then the MaterialCondition must be MAXIMUM.
High degree curve	Quality	Excessively high-degree curves (G-CU-HD) must not be used.
Free edge	Quality	Free edges (G-SH-FR) must not be used.
Over-used edge	Quality	Over-used edges (G-SH-NM) must not be used.
Fragmented curve	Quality	Fragmented curves (G-CU-FG) must not be used.
Unit vector length	Quality	The length of a unit vector must be close to 1.

Parameters:

Parameter	Type	Purpose
CheckFormat	boolean	Enable/disable the format checks.
CheckQuality	boolean	Enable/disable the quality checks.
CheckSemantic	boolean	Enable/disable the semantic checks.
CheckLinkedDocuments	boolean	Enable/disable processing of linked documents.
MaxRecursionLevel	unsigned integer	Maximum recursion level of linked documents.
G-CU-HD/MaxDegree	unsigned integer	Maximum NURBS curve/surface degree.
G-CU-FG/MaxNumSegments	unsigned integer	Maximum number of curve fragments.
unitVectorLength	two doubles	Maximum and Minimum unit vector length.

The XSLT checks are performed via four .xsl files, namely, CheckDocument.xsl, CheckFormat.xsl, CheckQuality.xsl, and CheckSemantic.xsl. The checks performed by each of these files are described in the following subclauses.

5.4.1.1 CheckDocument.xsl

The Check Document.xsl file is used to trigger the use of CheckFormat.xsl, CheckQuality.xsl, and CheckSemantic.xsl.

In addition, it contains over 300 checks between linked files that a reference from one aspect of a feature or characteristic to another aspect of the same type of feature or characteristic does indeed do that. For example, if a circle feature measurement in a QIF results file intends to reference a circle feature item in an external QIF plan file, there is a check that the referenced object is a circle feature item and not anything else (such as a flatness definition or a torus feature item). The same set of checks is made within single instance files by key/keyref pairs, but key and keyref apply only within a single file and cannot be used with linked files.

Finally, CheckDocument.xsl checks that the recursion depth of checking does not exceed the MaxRecursionLevel parameter.

5.4.1.2 CheckFormat.xsl

The CheckFormat.xsd file is used to make the following checks.

1. number of *element*s in list - Several dozen types of lists of *element*s are defined in QIF, for example, the **PartSet** in a **Product** is a list of **Part**s. Each of these lists has an **n** *attribute* giving the number of *element*s in the list. It is checked that the actual count of *elements* in the list is equal to **n**. This check is performed on every list of *elements* containing an **n** *attribute*.

2. external references - A QIF document **D** may contain references to *element*s in external QIF documents. To do this, **D** must include the Uniform Resource Identifier (URI) and QPId of each external document. It is checked that, for each external QIF document of **D**:

a. A file **E** exists with the URI given in **D**, and **E** has a **QIFDocument** *element* in it
b. The QPId of the **QIFDocument** in **E** is the one given in **D**
c. Every external entity reference in **D** that is supposed to be in **E** is, in fact, in **E**.

3. NURBS curves - It is checked that in any 2D or 3D NURBS curve, the number of control points equals the number of knots minus the order of the curve.

4. NURBS surfaces - It is checked that for any NURBS surfaces the number of control points equals the product of (**nKtU** - **ordU**) and (**nKtV** - **ordV**).

5.4.1.3 CheckQuality.xsl

The CheckQuality.xsl file is used to make the following checks.

1. It is checked that there are no NURBS surfaces of excessively high degree. The maximum degree is a parameter that may be set in CheckParameters.xml.

2. It is checked that there are no free edges. A free edge is an edge that is used by only one face in a shell.

3. It is checked that there are no overused edges. A overused edge is an edge that is used by more than two faces in a shell.

4. It is checked that 2D and 3D polylines do not have excessively many segments. The maximum number of segments is a parameter that may be set in CheckParameters.xml.

5.4.1.4 CheckSemantic.xsl

The CheckSemantic.xsl file is used to make the following checks.

1. It is checked that a FeatureNominal is connected to an entity (which should be a topology entity). This check is optional and is inactive by default. It can be activated via CheckParamaters.xml. If the connection is to an internal entity, a separate *key*/*keyref* check ensures that the entity is a topology entity. If the connection is to an external entity, its type is not checked.

2. It is checked that in any position characteristic definition, if the tolerance is zero, the material condition must be maximum.

5.4.2 Redundancy Checks

QIF contains redundancy checks at two levels.

At the low level, each type of *element* describing a list, set, or collection includes a required *attribute* "n," which is the number of items contained in the list, set, or collection. If any of the low level "n" counts do not match the actual number of items in a list, set, or collection, or the optional **ValidationCounts** *element* is present and indicates the presence of a list or item which is not present, or indicates a number of items in a list which does not match the actual number of items in that list, or a top level list or item is present but not indicated as such in the **ValidationCounts** *element* if it is present, then there is an internal inconsistency in the QIF document and the document is not valid.

At the high level, the root level of a QIF document has an optional *element* **ValidationCounts**, which can be used to indicate the presense and number of items in key, top level QIF document lists and items.

5.4.3 Product Data Quality

QIF defines an optional **ProductDataQuality** *element* of type ***ProductDataQualityType***, which contains information about product data quality (PDQ) requirements and a list of checks performed on the QIF document. PDQ information includes the declaration of product data quality along with Boolean *element*s indicating if checks have been approved and if checks have been done.

Each PDQ check listed in the optional ProductDataQualityChecks *element* contains information on the type of check, its description, requirements, status, and results statement. Optionally, the source requiring the check, the software application used for the check, and the URL of any XSLT used in the check, can be specified.

5.4.4 Digital Signature

A QIF document can include a digital signature in the optional **Signature** *element* of ***SignatureType*** defined in the **xmldsig-core-schema.xsd** schema, which can be found at **http://www.w3.org/TR/2002/REC-xmldsig-core-20020212**, and which implements data elements to the ITU-T X.509 | ISO/IEC 9594-8 (10/2012) Public-key and attribute certificate frameworks standard.

This digital signature's features include a certificate containing:

- Signature version

- Serial number of the certificate
- Algorithm id
- Certificate issuer name
- Validity period
- Certificate subject name
- Subject public key information
- Optional extensions
- The certificate signature algorithm
- The certificate signature

Included in the certificate signature is a hash value, which can be used to detect unauthorized editing or accidental corruption of the QIF document.

5.4.5 Long Term Archiving and Retrieval

LOng Term Archiving and Retrieval (LOTAR) of product and technical data is an important standard describing a product lifecycle for many products, particularly in aerospace and military applications. In this context, “Long Term” means long enough for information to be affected by changing technologies, including support for new media and data formats, or a changed user community. Similarly, “Archival storage” refers to the process that ensures data remains available for access. The LOTAR goal is to define and store digital information models in such a manner that the information can be fully retrieved, and data files corresponding to the model can be fully realized at any future time, even if the systems used to define and store the information are then obsolete. LOTAR International is the official name of the consortium that generated the LOTAR standard.

The following requirements are required of the QIF standard for it to be suitable for LOTAR:

- It must be an open data standard
- Ingest must be correct
- Retrieve must be accurate
- A unique persistent identifier or identification scheme must exist

Therefore, if a part is modeled in QIF, all relevant information, including measurement-related CAD with associated GD&T part information, part measurement plans, and required resources and rules can all be retrieved and utilized no matter what software and manufacturing systems are available at some far future date.

This claim can be made largely because QIF is an open, non-proprietary standard information model. This gives a strong probability of accurate information retrieval in the far distant future, particularly compared to proprietary, non-standard, quality measurement digital information exchange models.

In particular, the QIF model has been developed in close collaboration with the LOTAR International group to further ensure the archival suitability of QIF. The express objective of LOTAR International is “to develop, test, publish and maintain standards for long-term archiving (LTA) of digital data, such as 3D CAD and PDM data. These standards will define auditable archiving and retrieval processes. Use of the standard series by other branches of industry such as the automotive or shipbuilding industry is possible. The results are harmonized with e.g. the Recommendation 4958 for long-term archiving of the German Association of the Automotive Industry (VDA) and are based on the ISO 14721, Open Archival Information System (OAIS) Reference Model. The documents for

the standard are published as the EN9300 series and, in cooperation with the AIA, also as the National Aerospace Standard (NAS).” (http://www.lotar-international.org/).

5.5 QIF manufacturing functional requirements

Requirements that have been validated by inspection and examples in QIF include ASME GD&T, ISO GPS, ISO 22093, and ANSI/DMIS 105.3. Functional requirements met in QIF include the following:

- Information requirements for QIF encompass the principles of geometric dimensioning and tolerancing as described in ASME GD&T and ISO GPS, as well as workflow practices and quality management functions.
- QIF encompasses all planning and results information defined in ANSI/DMIS 105.3. QIF also includes device, tool, and sensor data.
- QIF instance files support all information defined for first article inspection reports as defined in the standard AS9102B [1].
- The neutral data format specifications are accompanied by fully defined semantics derived where applicable from other standards like ANSI/DMIS 105.3, and AS9102B. The semantics ensure that data cannot be misinterpreted between sender and receiver of QIF instance files.
- Inspection results data formatted as QIF Results can be used both for a reverse engineering process where actual measurement data is stored without the presence of nominal information, and for a conventional measurement process where inspection is planned using nominal part feature data.
- QIF facilitates traceability of quality results to inspection and measurement processes, including identification of measurement devices and operators, software applications, and CAD models.
- QIF facilitates traceability of quality results to manufacturing processes by maintaining links, so that inspection data can be used to monitor, control, and improve manufacturing processes. The QIF scope includes support of manufacturing process and product validation testing.
- QIF Results data can be written to facilitate re-analysis of measured point data.
- QIF data supports quality systems based on model based definition, as well as systems that implement 2D drawing-based processes.

5.6 QIF and STEP

The ISO 10303 family of standards is informally known as the “Standard for the Exchange of Product model data” or “STEP.” QIF has relevance to several of the STEP standards, especially STEP AP203, "Configuration controlled 3D designs of mechanical parts and assemblies", STEP AP242, “Managed model based 3d engineering,” and STEP AP219, “Dimensional inspection information exchange.” The relationship and harmonization between QIF and the relevant STEP standards will be addressed in other documents outside this standard.

5.7 QIF information model design guidelines

A QIF design principle is to follow a decoupled normalized relationship model. As such, many relations between data elements (also known as instanced types) in the QIF schemas are made using identifiers rather than allowing a parent type to directly contain a child type (as you would find in a strictly hierarchical model). Each “object” or “instanced type” that needs to be referenced by another type is given a unique identification designator (id). The referencing object may then establish a relation by calling out the id rather than redefining the entire data element within itself. By decoupling the child from the parent we introduce the ability to reuse components of the relationship without duplicating definitions. This concept of decoupling is one of the pillars of QIF extensibility. It also leads to reducing instance file size compared to purely hierarchical design, and may lead to better data integrity because the risk of duplicating errors is reduced.

The QIF models represent measurement features and characteristic objects with four aspects: item, definition, nominal, and measurement. The relationships between data objects of each aspect type in an instance file are implemented using a relationship scheme described in more detail in subclause 5.9. Because the scope of QIF covers the entire workflow of enterprise quality systems, the aspects allow QIF instance files to describe nominal and measured GD&T quality data, as well as express the relationships and other data generated during the workflow of planning, execution, reporting, and analysis.

We will use the terms “data object” and “object” to mean a grouping of information in a QIF XML instance file, defined by elements of the QIF information model.

5.8 Overview of XML schema file modularity

A complete model built using XML Schema Definition Language (XSDL) is called a *schema*. However, the complete model may consist of information items from several different *schema* files. Typically, the complete model will be defined using a top level (or root) *schema* file which will use subordinate *schema* files. In XSDL, using definitions in other files is indicated by an *include* directive. The top level file *includes* subordinate files, and those subordinates may *include* other subordinates.

For example, **QIFDocument** is a complete application *schema*. The top-level *schema* file is QIFDocument.xsd. It includes the primary *schema* file for each of the six QIF application areas, and each of those *includes* subordinate files collected in the QIF Library. The net effect is that the **QIFDocument** *schema* contains everything defined in all the QIF *schema* files.

An XML data file conforming to an XML *schema* is called an instance file. Every instance file conforming to the QIF model will have **QIFDocument** as the root of a hierarchy of information.

The XML schema file hierarchy of QIF is shown in Figure 3.

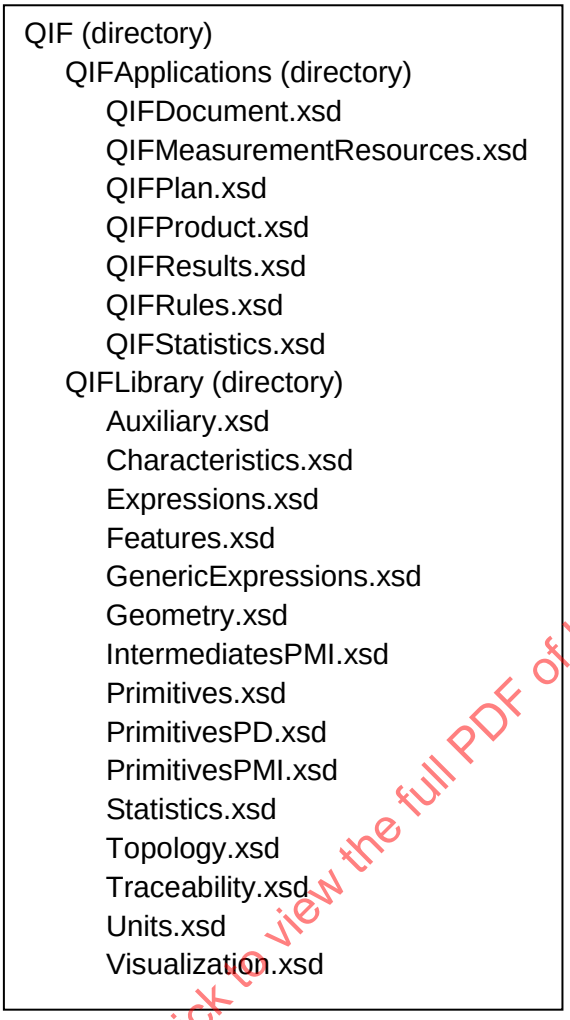


Figure 3 – QIF XML schema directory structure

The QIF Library is modularized by grouping related *type* and *element* definitions together in a *schema* file. Some of the *schema* files in the QIF Library serve multiple applications, and some serve only a single application.

- Expressions.xsd and GenericExpressions.xsd serve only QIFRules and QIFPlan.
- Auxiliary.xsd, Geometry.xsd, Topology.xsd, and Visualization.xsd serve primarily QIFProduct (though QIFMeasurementResources also uses the first three of those).
- Statistics.xsd serves only QIFStatistics.

The QIF application models are each described in a separate subject area clause (QIF Clauses 7 through 12). Clause 6 of the QIF standard describes the contents of each *schema* file in the QIF Library.

5.9 Data structures

5.9.1 The QIFDocument element

QIFDocument is the highest level *element* of all QIF instance files. Its structure is illustrated in Figure 4. QIFDocument can be used to generate many different sorts of instance files. Such a file may include information from one or more application areas. For example, a **QIFDocument** instance file defining QIF Rules will typically have only identifying information and a **Rules** *element*. On the other hand, if a product has been modeled in the **Product** *element* of a **QIFDocument**, an

instance file containing a plan for that product is likely to have the same **Product** *element* and a **Plan** *element* in the **QIFDocument**.

A QIF Statistics instance file is likely to have at least a **Product** *element*, a **Results** *element*, and a **Statistics** *element*.

However, a user has the option of putting related application information in a single instance file or keeping that information in separate instance files. A QIF Statistics instance file might contain only identifying information and a **Statistics** *element*. The product information referenced in the **Statistics** *element* might be in a second instance file that was generated when the product was designed, and the measurement results information referenced in the **Statistics** *element* might be in a third instance file that was generated when an instance of the product was measured. When related information from different applications is in a single file, it is connected using identifiers (**id**s) that are local to the file. When that information is in separate files, it is connected using a combination of local **id**s and QIF Persistent Identifiers (**QPId**s), which are universally unique.

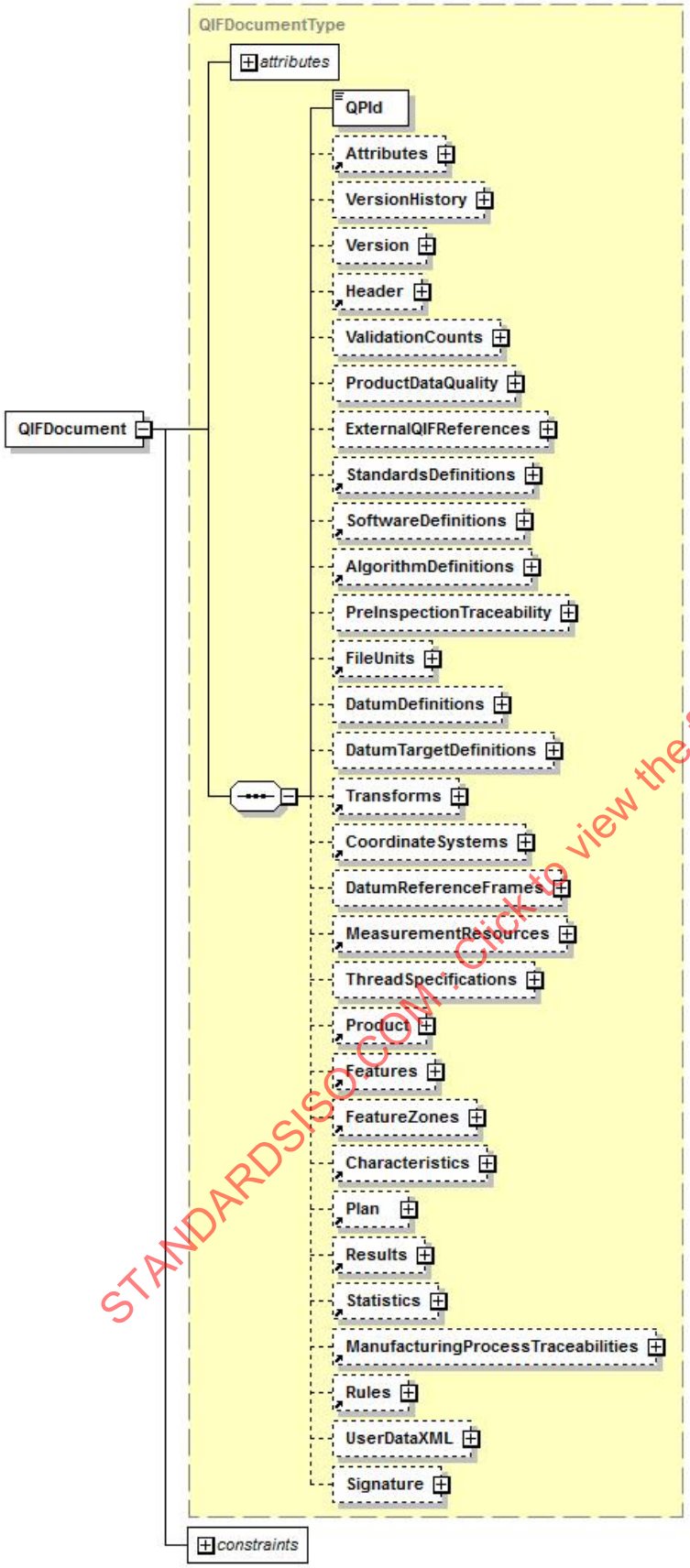


Figure 4 – Structure of the **QIFDocument** *element*

A brief description of all sub-*element*s of the QIF Document *element* follows. All *element*s other than the required **QPId** *element* are optional; however, it is recommended that a **QIFDocument** contains at least a **Header** and one or more of the **MeasurementResources**, **Product**, **Plan**, **Results**,

Rules, and **Statistics** application areas. It is strongly suggested that every QIFDocument also have **Version** and **FileUnits**.

- **QPId** – the QIF persistent identifier for the document. This is a universally unique identifier. QPIds are discussed in subclause 5.13.2. ***QPIdType*** is defined in Primitives.xsd.
- **Attributes** – provides the user with opportunity to attach typed information that is not modeled in QIF. It is described further in subclause 5.25. The ***AttributesType*** and **Attributes** global *element* are defined in Primitives.xsd.
- **VersionHistory** – version history information about the file. The ***VersionHistoryType*** is defined in Intermediates.xsd.
- **Version** – version information about the file, including **TimeCreated** and **Signoffs**.
- **Header** – identification of the source of the file and the scope of the file. The ***QIFDocumentHeaderType*** and **Header** global *element* are defined in QIFDocument.xsd.
- **ValidationCounts** – the size of each of about 50 lists of QIF *element*s. This is provided so that an application can check the integrity of the document by checking the counts against the actual numbers of *element*s in the lists. Each count is optional. If a count is omitted for a list, it does not imply that there are zero *element*s in the list.
- **ProductDataQuality** – a description of quality checks applied to the document. Quality checks are discussed in subclause 5.4.3.
- **ExternalQIFReferences** – a list of external QIF documents referenced in the document, with a local **id** for each external document. External QIF references are discussed in subclause 5.13.3.
- **StandardsDefinitions** – a list of standards referenced in the document.
- **SoftwareDefinitions** – a list of software systems referenced in the document.
- **AlgorithmDefinitions** – a list of algorithm definitions referenced in the document.
- **PreInspectionTraceability** – information about the provenance of the file before inspection has occurred. The ***PreInspectionTraceabilityType*** is defined in Traceability.xsd. Preinspection traceability is discussed briefly in subclause 6.14.2.
- **FileUnits** – information about the units used in the file. Units are discussed in subclause 5.18 and in more detail in subclause 6.15. The ***FileUnitsType*** and **FileUnits** global *element* are defined in Units.xsd.
- **DatumDefinitions** – a list of datum definitions. Datum definitions are discussed in subclause 5.17.6.1. ***DatumDefinitionsType*** is defined in IntermediatesPMI.xsd.
- **DatumTargetDefinitions** – a list of datum targets. Datum target types are listed in subclause 6.8. The ***DatumTargetDefinitionsType*** and the various datum target types are defined in IntermediatesPMI.xsd.
- **Transforms** – a list of transforms. Transforms are discussed in subclause 5.16.3. The ***TransformListType*** and **Transform** global *element* are defined in IntermediatesPMI.xsd.
- **CoordinateSystems** – a list of coordinate systems. Coordinate systems are discussed in subclauses 5.16.4, 5.16.5, and 5.16.6. The ***CoordinateSystemType***, ***CoordinateSystemListType*** and **CoordinateSystems** global *element* are defined in IntermediatesPMI.xsd.
- **DatumReferenceFrames –** a list of datum reference frames. Datum reference frames are discussed in subclause 5.17.6. The ***DatumReferenceFramesType*** and the ***DatumReferenceFrameType*** are defined in IntermediatesPMI.xsd.
- **MeasurementResources –** information about measurement resources. Measurement resources are discussed in Clause 1. The ***MeasurementResourcesType***,

MeasurementResources global *element*, and other measurement resource types are defined in QIFMeasurementResources.xsd.

- **ThreadSpecifications –** a list of thread specifications. Thread specifications are discussed in subclause 5.22. Thread specification types are defined in IntermediatesPMI.xsd. Thread characteristics and features are defined in Characteristics.xsd and Features.xsd, respectively.
- **Product –** information about parts and assemblies, both designs and physical instances of the designs. These are discussed in subclause 5.11 and in more detail in Clause 0 and subclauses 6.2, 6.7, 6.10, 6.13, and 6.16. The ***ProductType*** and **Product** global *element* are defined in QIFProduct.xsd.
- **Features –** information about features. Features are discussed in subclause 5.9.2, 5.9.5, 5.19, 5.20, and 5.23, and in subclause 6.5. The ***FeaturesAspectsListsType*** and **Features** global *element* are defined in Features.xsd.
- **FeatureZones** – information about feature zones. A feature zone defines a portion of a feature and is used to define characteristic tolerance zones or datum target areas or to specify the meaning of a characteristic.
- **Characteristics –** information about characteristics. Characteristics are discussed in subclauses 5.9.3, 5.9.4, 5.9.5, and 5.24 and in subclause 6.3. The ***CharacteristicAspectsListsType*** and **Characteristics** global *element* are defined in Characteristics.xsd.
- **Plan –** information about a measurement plan. Measurement plans are discussed briefly in subclause 5.2 and in more detail in Clause 0. The ***PlanType*** and **Plan** global *element* are defined in QIFPlan.xsd.
- **Results –** information about one or more measurement results. Each measurement results includes all information associated with a single product measurement such as measured feature, characteristic, and coordinate system transform information. Measurement results are discussed briefly in subclause 5.13.4 and in detail in Clause 11. The ***ResultsType*** and **Results** global *element* are defined in QIFResults.xsd.
- **Statistics –** information about quality statistics aimed primarily at process control; both plans and results are included. Statistics is discussed briefly in subclause 5.2 and subclause 6.12. Statistics is discussed in detail in Clause 12. The ***StatisticsType*** and **Statistics** global *element* are defined in QIFStatistics.xsd.
- **ManufacturingProcessTraceabilities** – information about manufacturing processes. Manufacturing process traceability is discussed briefly in subclause 6.14.5. The ***ManufacturingProcessTraceabilitiesType*** and the **ManufacturingProcessTraceabilities** global *element* are defined in Traceability.xsd.
- **Rules** – information about rules to be used in inspection planning and/or programming. Rules are discussed briefly in subclause 5.2 and in detail in Clause 0. Rules types are defined in QIFRules.xsd.
- **UserDataXML –** information in XML format modeled in some non-QIF namespace. The ***UserDataXMLType*** and the **UserDataXML** global *element* are defined in Primitives.xsd. This is intended to be used only for data that cannot be modeled in QIF.
- **Signature** – an electronic signature that identifies the signer and contains a check code against which the signature will not verify if the file was changed after the signature was inserted.

5.9.2 Four aspects of features data

Feature information is defined in the QIF library using four aspects: definition, nominal, item, and measurement. In a QIF instance file each feature data object has a unique identifier, and relationships between objects are expressed by references to the identifiers as shown in Figure 5. These four library data types were designed to express quality information beyond the scope of solely inspection results reporting. The item aspect, in particular, includes information related to part design as well as information generated by planning activities.

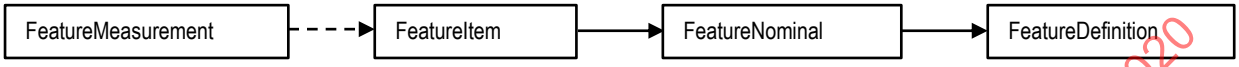


Figure 5 – Reference connections among feature data objects in a QIF XML instance file

Solid lines show required references, dashed lines show optional references.

The four aspects of feature data will be illustrated with the simple example of a plate with four holes as shown in Figure 6.

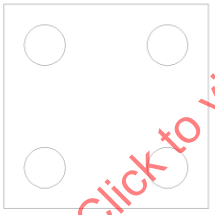


Figure 6 – A plate with four holes

This plate with holes can exist in several contexts: it may be a printed 2D drawing, it may be a 3D solid MBD, it may be a CMM inspection plan or program, it may be an actual physical part, it may be a CMM results report, or it may exist as all of the above. Regardless, in QIF the four holes would be considered as cylinder features. In the MBD or drawing contexts, nominal information for these cylinders will exist. In the physical part context, information about the actual cylinders can exist if they are measured. So, one would assume that QIF would only need to contain objects that define feature nominal and feature measurement information. Such a simple approach can result in the redundant expression of data and may not mirror the design intent.

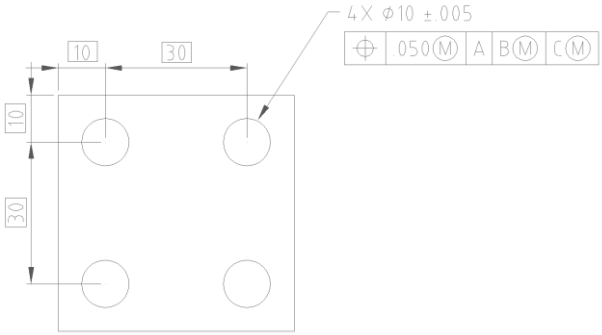


Figure 7 – A plate with four holes and GD&T

The design of the plate shown in Figure 7 illustrates that some nominal information about the holes is shared and some is not. Each cylinder has a unique nominal location defined by basic or theoretically exact dimensions but all have a shared nominal diameter. QIF therefore splits the nominal information between a shareable feature definition and a non-shareable feature nominal.

A **feature definition** data object is intended to be reusable, in that it includes information (e.g., cylinder: diameter) that is common to a feature type but independent of a specific instance of the feature (e.g., a specific hole in our example). A single feature definition can be referenced by many nominal feature objects. Only nominal feature objects may reference feature definition objects.

In the example, the cylinder definition with the shared diameter of the holes might look like this in a QIF XML instance file:

```
<CylinderFeatureDefinition id=”22”>
  <Diameter>10</Diameter>
</CylinderFeatureDefinition>
```

A **feature nominal** data object adds additional feature information to the feature definition by defining information unique to a particular instance of a feature. For example, an instance of the ***CylinderFeatureDefinitionType*** provides the diameter for a cylinder, while an instance of the ***CylinderFeatureNominalType*** references the ***CylinderFeatureDefinitionType***, and gives the location point and the axis vector and optional target points on a specific cylinder on a part to be measured.

Feature nominal data is unique to a particular feature in the context of a simple part. But in the context of multiple instances of a part in an assembly it can be shared. In this case the feature item’s (see below) reference to the nominal will include its assembly path. The feature nominal has optional *elements* for **Name** and **Description** which may be used to identify and describe the feature in the component part context perhaps as it appears in a CAD system feature tree.

If a part is measured, then each cylinder will have unique location and size information. No feature measurement data can be shared among feature instances.

The four holes in the example would have four feature nominals in a QIF XML file:

```
<CylinderFeatureNominal id="23">
  <FeatureDefinitionId>22</FeatureDefinitionId>
  <Axis>
    <AxisPoint>40 40 0</AxisPoint >
    <Direction>0 0 1</Direction>
  </Axis>
</CylinderFeatureNominal>
<CylinderFeatureNominal id="24">
  <FeatureDefinitionId>22</FeatureDefinitionId>
  <Axis>
    <AxisPoint>40 10 0</AxisPoint >
    <Direction>0 0 1</Direction>
  </Axis>
</CylinderFeatureNominal>
<CylinderFeatureNominal id="25">
  <FeatureDefinitionId>22</FeatureDefinitionId>
  <Axis>
    <AxisPoint>10 10 0</AxisPoint >
    <Direction>0 0 1</Direction>
  </Axis>
</CylinderFeatureNominal>
<CylinderFeatureNominal id="26">
  <FeatureDefinitionId>22</FeatureDefinitionId>
  <Axis>
    <AxisPoint>10 40 0</Z> </AxisPoint >
    <Direction>0 0 1</Direction>
  </Axis>
</CylinderFeatureNominal>
```

Each has a unique **id** and specifies a unique location but each references the same feature definition.

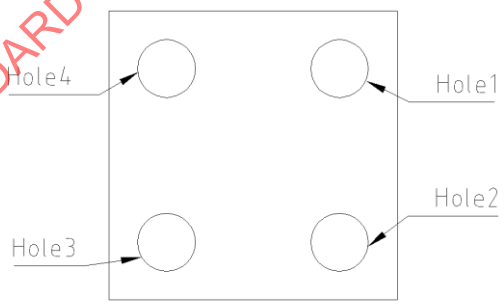


Figure 8 – A plate with four holes with names.

A **feature item** data object represents an instance of a feature at the planning stage of the metrology process. The feature item data object provides: the name assigned to the feature (as in

Figure 8), optional links to upstream CAD data, reference to a part definition object **id**, and a required reference to a nominal feature object.

If the same feature on the same physical part is measured several times, it is expected that a feature item data object will be defined for each measurement. Some other examples of workflow that cause instantiation of a feature item object include: any process at any time that requires a named feature, planning inspection of a part, planning free-form measurement of a part for reverse engineering purposes, bringing a legacy CMM report into QIF, or mining a legacy CMM program for nominal features and characteristics.

A QIF XML instance file might have feature items like:

```
<CylinderFeatureItem id="27">
  <FeatureNominalId>23</FeatureNominalId>
  <FeatureName>Hole_1</FeatureName>
  …
</CylinderFeatureItem>
<CylinderFeatureItem id="28">
  <FeatureNominalId>24</FeatureNominalId>
  <FeatureName>Hole_2</FeatureName>
  …
</CylinderFeatureItem>
<CylinderFeatureItem id="29">
  <FeatureNominalId>25</FeatureNominalId>
  <FeatureName>Hole_3</FeatureName>
  …
</CylinderFeatureItem>
<CylinderFeatureItem id="30">
  <FeatureNominalId>26</FeatureNominalId>
  <FeatureName>Hole_4</FeatureName>
  …
</CylinderFeatureItem>
```

Each item references a single nominal and assigns a feature name (the ellipses … indicate extra required *elements* outside the scope of this simple example).

Feature nominal objects can be defined at the whole product level in the global coordinate system or at the part or subassembly level in a local coordinate system whereas the feature item is always defined at the whole product level. The feature nominal can be referenced by several feature items, one for each part or subassembly instance in an assembly. When a feature item references such a feature nominal, the optional **asmPath** *attribute* on the **FeatureNominalId** *element* is used to unambiguously define the assembly path which takes the feature nominal from the part or subassembly context to the whole part context.

A QIF XML instance file of an assembly containing two instances of the 4-hole plate might have feature items like:

```
<CylinderFeatureItem id="27">
  <FeatureNominalId asmPath=”6”>23</FeatureNominalId>
  <FeatureName>Part1_Hole_1</FeatureName>
  …
</CylinderFeatureItem>
```



```
<CylinderFeatureItem id="28">
  <FeatureNominalId asmPath=”6”>24</FeatureNominalId>
  <FeatureName>Part1_Hole_2</FeatureName>
  …
</CylinderFeatureItem>
<CylinderFeatureItem id="29">
  <FeatureNominalId asmPath=”6”>25</FeatureNominalId>
  <FeatureName>Part1_Hole_3</FeatureName>
  …
</CylinderFeatureItem>
<CylinderFeatureItem id="30">
  <FeatureNominalId asmPath=”6”>26</FeatureNominalId>
  <FeatureName>Part1_Hole_4</FeatureName>
  …
</CylinderFeatureItem>
<CylinderFeatureItem id="31">
  <FeatureNominalId asmPath=”8”>23</FeatureNominalId>
  <FeatureName>Part2_Hole_1</FeatureName>
  …
</CylinderFeatureItem>
<CylinderFeatureItem id="32">
  <FeatureNominalId asmPath=”8”>24</FeatureNominalId>
  <FeatureName>Part2_Hole_2</FeatureName>
  …
</CylinderFeatureItem>
<CylinderFeatureItem id="33">
  <FeatureNominalId asmPath=”8”>25</FeatureNominalId>
  <FeatureName>Part2_Hole_3</FeatureName>
  …
</CylinderFeatureItem>
<CylinderFeatureItem id="34">
  <FeatureNominalId asmPath=”8”>26</FeatureNominalId>
  <FeatureName>Part2_Hole_4</FeatureName>
  …
</CylinderFeatureItem>
```


A **feature measurement** data object provides feature information that has been directly measured or constructed. For example, a ***CylinderFeatureMeasurementType*** will contain the measured location, orientation, and size of a cylinder. For an inspection that has been programmed from CAD data, the feature measurement data object contains a reference to the associated feature item (which must reference the related nominal feature object (which in turn, has a related feature definition object)). Feature measurement data generated during a reverse engineering process might not contain a reference to a feature item.

The four holes when measured might look like this in a QIF XML instance file:

```
<CylinderFeatureMeasurement id="31">
  <FeatureItemId>27</FeatureItemId>
  <Axis>
    <AxisPoint>40.002 39.994 0</AxisPoint>
    <Direction>-0.001 0 1</Direction>
  </Axis>
  <Diameter>10.003</Diameter>
```



```
</CylinderFeatureMeasurement>
<CylinderFeatureMeasurement id="32">
  <FeatureItemId>28</FeatureItemId>
  <Axis>
    <AxisPoint>39.967 10.011 0</AxisPoint>
    <Direction>0.009 -0.009 0.999</Direction>
  </Axis>
  <Diameter>10.005</Diameter>
</CylinderFeatureMeasurement>
<CylinderFeatureMeasurement id="33">
  <FeatureItemId>29</FeatureItemId>
  <Axis>
    <AxisPoint>10.002 10.013 0</AxisPoint>
    <Direction>0.001 0 1</Direction>
  </Axis>
  <Diameter>9.996</Diameter>
</CylinderFeatureMeasurement>
<CylinderFeatureMeasurement id="34">
  <FeatureItemId>30</FeatureItemId>
  <Axis>
    <AxisPoint>9.987 40.013 0</AxisPoint>
    <Direction>0 -0.004 1</Direction>
  </Axis>
  <Diameter>10.007</Diameter>
</CylinderFeatureMeasurement>
```

Each measured hole is represented by a cylinder measurement that references the corresponding cylinder item, and each has a unique measurement result.

5.9.3 Four aspects of characteristics

As with features, characteristics have four aspects defined in the QIF library: definition, nominal, item, and measurement. This is shown in Figure 9. Data objects of each aspect type can be linked in a QIF XML instance file to express the semantics of GD&T and quality workflow using the scheme described in subclause 5.7. As with features, the aspects cover a quality workflow scope wider than solely results reporting.

The **characteristic definition** is the part of a characteristic that can be shared among different characteristics. An example would be a standard diameter tolerance; one manufacturer, for instance, has a standard diameter tolerance for sheet metal parts of (+.25/-.04) mm regardless of the diameter. As another example, one often sees tolerances specified for dimensions based on the number of decimal places: dimensions to one decimal place are ±.2 mm, those to two decimal places are ±.05 mm, etc.

In the example shown in Figure 7, a QIF XML instance file representation of the diameter tolerance as a characteristic definition might look like:

```
<DiameterCharacteristicDefinition id="40">
  <Tolerance>
    <MaxValue>0.005</MaxValue>
    <MinValue>-0.005</MinValue>
    <DefinedAsLimit>false</DefinedAsLimit>
  </Tolerance>
```

```
</DiameterCharacteristicDefinition>
```

The **characteristic nominal** is the part of a characteristic that is not shared among different characteristics, not to be confused with sharing a characteristic among several features. An example would be a diameter tolerance for a set of holes in a pattern all with the same diameter. That shared diameter becomes the target value in the nominal characteristic. Very often, each characteristic definition will only be referenced by a single characteristic nominal, the pair together representing one call-out on a print such as the diameter with tolerance in Figure 7.

In the example, an instance file containing the diameter characteristic nominal might look like:

```
<DiameterCharacteristicNominal id="41">
  <CharacteristicDefinitionId>40</CharacteristicDefinitionId>
  <TargetValue>10</TargetValue>
</DiameterCharacteristicNominal>
```

This single diameter characteristic nominal can be shared among the four holes.

The **characteristic item** is the mechanism used to apply a tolerance to an individual feature. Our plate with holes would have one characteristic item for each hole because each hole will have an individual tolerance condition.

Figure 9 – References among characteristic data objects in a QIF XML instance file.

Each characteristic item references the single shared characteristic nominal, and in turn that characteristic nominal references a single characteristic definition which may or may not be referenced by other characteristic nominals. Characteristic items are required to reference a characteristic nominal.

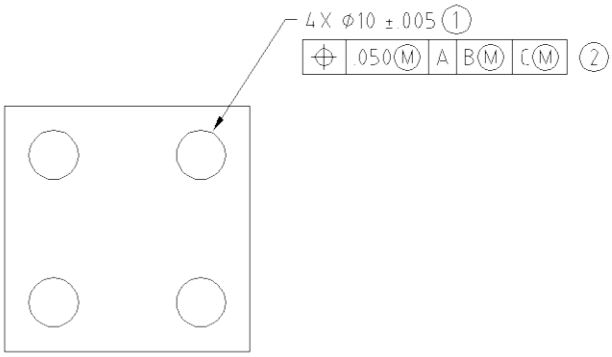


Figure 10 – A plate with ballooned tolerances.

The idea of a characteristic item may at first seem a bit redundant but it allows for a unique identifying name or key characteristic identifier to be assigned on a per-feature basis. Our diameter

tolerance might be “ballooned” on the print (as in Figure 10) as key characteristic number 1, then each characteristic item could be labeled with key characteristics of 1A, 1B, 1C or 1_1, 1_2, 1_3 depending on company standards.

From the example, here is what a QIF XML instance file with key characteristics might look like:

```
<DiameterCharacteristicNominal id="41">
  <CharacteristicDefinitionId>40</CharacteristicDefinitionId>
  <CharacteristicDesignator>
    <Designator>1</Designator>
    <Criticality>
      <LevelEnum>KEY</LevelEnum>
    </Criticality>
  </CharacteristicDesignator>
  <TargetValue>10</TargetValue>
</DiameterCharacteristicNominal>
…
<DiameterCharacteristicItem id="42">
  <Name>Hole_1_diam</Name>
  <CharacteristicDesignator>
    <Designator>1_1</Designator>
    <Criticality>
      <LevelEnum>KEY</LevelEnum>
    </Criticality>
  </CharacteristicDesignator>
  <CharacteristicNominalId>41</CharacteristicNominalId>
</DiameterCharacteristicItem>
<DiameterCharacteristicItem id="43">
  <Name>Hole_2_diam</Name>
  <CharacteristicDesignator>
    <Designator>1_2</Designator>
    <Criticality>
      <LevelEnum>KEY</LevelEnum>
    </Criticality>
  </CharacteristicDesignator>
  <CharacteristicNominalId>41</CharacteristicNominalId>
</DiameterCharacteristicItem>
<DiameterCharacteristicItem id="44">
  <Name>Hole_3_diam</Name>
  <CharacteristicDesignator>
    <Designator>1_3</Designator>
    <Criticality>
      <LevelEnum>KEY</LevelEnum>
    </Criticality>
  </CharacteristicDesignator>
  <CharacteristicNominalId>41</CharacteristicNominalId>
</DiameterCharacteristicItem>
<DiameterCharacteristicItem id="45">
  <Name>Hole_4_diam</Name>
  <CharacteristicDesignator>
    <Designator>1_4</Designator>
    <Criticality>
      <LevelEnum>KEY</LevelEnum>
    </Criticality>
  </CharacteristicDesignator>
```

```
    <CharacteristicNominalId>41</CharacteristicNominalId>
  </DiameterCharacteristicItem>
```

The optional **CharacteristicDesignator** *element* on the characteristic nominal defines the shared characteristic designator (balloon number) from the call-out. The **CharacteristicDesignator** *elements* on each characteristic item define the individual characteristic designators for the diameter of the four individual holes. The **Criticality** *element* of a characteristic item overrides the Criticality of the nominal, if the values differ.

The **characteristic measurement** is the evaluation of the characteristic based on direct measurement or from feature measurement data. Just as with feature measurement information, there is no shareable information among characteristic measurements.

For the example, an instance file with measurements might look like:

```
<DiameterCharacteristicMeasurement id="46">
  <Status>
    <CharacteristicStatusEnum>PASS</CharacteristicStatusEnum>
  </Status>
  <CharacteristicItemId>42</CharacteristicItemId>
  <Value>10.003</Value>
</DiameterCharacteristicMeasurement>
<DiameterCharacteristicMeasurement id="47">
  <Status>
    <CharacteristicStatusEnum>PASS</CharacteristicStatusEnum>
  </Status>
  <CharacteristicItemId>43</CharacteristicItemId>
  <Value>10.005</Value>
</DiameterCharacteristicMeasurement>
<DiameterCharacteristicMeasurement id="48">
  <Status>
    <CharacteristicStatusEnum>PASS</CharacteristicStatusEnum>
  </Status>
  <CharacteristicItemId>44</CharacteristicItemId>
  <Value>9.996</Value>
</DiameterCharacteristicMeasurement>
<DiameterCharacteristicMeasurement id="49">
  <Status>
    <CharacteristicStatusEnum>FAIL</CharacteristicStatusEnum>
  </Status>
  <CharacteristicItemId>45</CharacteristicItemId>
  <Value>10.007</Value>
</DiameterCharacteristicMeasurement>
```


5.9.4 Default tolerances and characteristics

There are four types of dimension: a basic or theoretically exact dimension like 40.0 (a dimension to define the nominal location of a feature to which a geometric tolerance is applied), a reference or auxiliary dimension like (40.0) (a dimension which can be calculated from other dimensions but which is provided for easy reference), a directly toleranced dimension like 40.0±0.1 (either as specification limits or as a bidirectional tolerance about a nominal value), and a dimension which is not directly toleranced like 40.0.

This latter dimension, although not explicitly toleranced, is often subject to an “unless otherwise specified” default tolerance. Such default tolerances are often called “box tolerances” because they can appear in a box at the corner of a drawing and often take the form:

Unless otherwise specified	
X.X	±.2 mm
X.XX	±.05 mm
X.XXX	±.005 mm
XX°	±1°

In QIF characteristics are explicitly typed; a length nominal cannot use a diameter definition to define its tolerance. Because these default tolerances cross type boundaries (the same tolerance may apply to a length, diameter, distance, coordinate etc.) QIF has a mechanism for storing and referencing un-typed default tolerances. Instead of defining a tolerance directly in a characteristic definition such as:

```
<DiameterCharacteristicDefinition id="40">
  <Tolerance>
    <MaxValue>0.005</MaxValue>
    <MinValue>-0.005</MinValue>
    <DefinedAsLimit>false</DefinedAsLimit>
  </Tolerance>
</DiameterCharacteristicDefinition>
```

The tolerance values can be defined by reference to a default tolerance:

```
<DiameterCharacteristicDefinition id="40">
  <Tolerance>
    <DefinitionId>20</DefinitionId>
    <DefinedAsLimit>false</DefinedAsLimit>
  </Tolerance>
</DiameterCharacteristicDefinition>
```

where the **DefinitionId** *element* references a default tolerance found in the **DefaultToleranceDefinitions** *element*. Such a default tolerance has the form:

```
<LinearToleranceDefinition id="20">
  <Tolerance>
    <MaxValue>0.005</MaxValue>
    <MinValue>-0.005</MinValue>
  </Tolerance>
</LinearToleranceDefinition>
```

This tolerance can be referenced by any linear value characteristic. The default tolerance for angular value characteristics is defined in a separate *element*:

```
<AngularToleranceDefinition id="21">
  <Tolerance>
    <MaxValue>1</MaxValue>
    <MinValue>-1</MinValue>
  </Tolerance>
</AngularToleranceDefinition>
```

The relationship between a default tolerance definition and the box tolerance annotation can be made more explicit with a note contained in the optional **Attributes** *element*:

```
<LinearToleranceDefinition id="20">
  <Attributes>
    <AttributeStr name=”Applicability”, value=”X.XXX”\>
  <\Attributes>
  <Tolerance>
    <MaxValue>0.005</MaxValue>
    <MinValue>-0.005</MinValue>
  </Tolerance>
</LinearToleranceDefinition>
```

A similar concept is that of a default characteristic applied to all non-directly toleranced features of a part, for example, “Unless otherwise specified all surfaces ⌓ | 0.5 | A | B | C”. Such contextual tolerances are stored in a special **DefaultCharacteristicDefinitions** *element* which is a list of characteristic definitions.

At the PMI stage of design, default tolerances and characteristics can be indicated simply by their presence in the **DefaultToleranceDefinitions** *element* and the **DefaultCharacteristicDefinitions** *element*. The connections between these tolerances and the features to which they apply can be made later at the metrology planning stage.

5.9.5 Relationships between the aspects

Three of the four aspects of characteristics are instantiated in the global **Characteristics** *element* of type ***CharacteristicAspectsListsType***. This type has an *element* for listing instances of each of the following aspects: **CharacteristicDefinitions**, **CharacteristicNominals**, and **CharacteristicItems**. The fourth, **CharacteristicMeasurements** is found on a per-measured-part basis in the **MeasurementResults** *element* of **Results**. This allows for multiple part measurements to be reported in a single QIF instance file. If more than one part has been measured then there will be more than one characteristic measurement referencing the same characteristic item.

To traverse all characteristics in a QIF XML instance file, the starting point would be the **CharacteristicItems** sub-*element* of the **Characteristics** *element*.

The four aspects of features are similarly instantiated in the global **Features** *element* of type ***FeatureAspectsListsType*** and in the **FeatureMeasurements** *element* on a per-measured-part basis in the **MeasurementResults** *element* of **Results**. ***FeatureAspectsListsType*** has an *element* for listing instances of **FeatureDefinitions**, **FeatureNominals**, and **FeatureItems**, all of which are optional. The feature item will come into existence either at the planning stage with reference to

nominal (and definition) data or optionally come into existence during a reverse-engineering use case.

To traverse all features in a QIF XML instance file, the starting point would be the **FeatureItems** sub-*element* of the **Features** *element*.

5.9.5.1 Connecting characteristics and features

The connections among the four aspects for both characteristic aspects as a group and feature aspects as a group have been discussed earlier. In general, the measurement references an item, the item references a nominal, and the nominal references the definition. The connection between characteristics and features is accomplished with the **FeatureItemIds** *element* on the ***CharacteristicItemBaseType*** from which all characteristic item types are derived.

The **FeatureItemIds** *element* allows a characteristic item to reference the feature item or feature items of the feature or features to which it applies. When more than one feature is referenced by the **FeatureItemIds** *element* it is because more than one feature is required to evaluate the tolerance, such as a distance or angle between two features. To determine the characteristic-feature relationships in a QIF instance file, the starting point would be the **CharacteristicItems** sub-*element* of the **Characteristics** *element*.

There is a secondary method of connecting characteristics and features for use at the PMI design stage when neither feature items nor characteristic items yet exist. Product manufacturing information (characteristics) are attached to the model based definition (features) by the **FeatureNominalIds** *element* on the ***CharacteristicNominalBaseType***.

Unlike the **FeatureItemsIds** *element* which references the minimum number of features necessary for the measurement of a single characteristic item, the **FeatureNominalIds** *element* can reference the many feature nominals to which the characteristic nominal applies. This one-to-many reference for a single characteristic nominal will give rise to as many characteristic items as there are features (or feature pairs, etc.).

From the example, in a QIF XML instance file containing both characteristics and features, the connections between characteristic items and feature items and characteristic nominals and feature nominals might look like this:

```
<DiameterCharacteristicNominal id="41">
  <CharacteristicDefinitionId>40</CharacteristicDefinitionId>
  <FeatureNominalIds n="4">
    <Id>23</Id>
    <Id>24</Id>
    <Id>25</Id>
    <Id>26</Id>
  </FeatureNominalIds>
  <CharacteristicDesignator>
    <Designator>1</Designator>
    <Criticality>
      <LevelEnum>KEY</LevelEnum>
    </Criticality>
  </CharacteristicDesignator>
 <TargetValue>10</TargetValue>
</DiameterCharacteristicNominal>
…
```

```
<DiameterCharacteristicItem id="42">
  <Name>Hole_1_diam</Name>
  <FeatureItemIds n="1">
    <Id>27</Id>
  </FeatureItemIds>
  <CharacteristicNominalId>41</CharacteristicNominalId>
  </DiameterCharacteristicItem>
<DiameterCharacteristicItem id="43">
  <Name>Hole_2_diam</Name>
 <FeatureItemIds n="1">
   <Id>28</Id>
  </FeatureItemIds>
  <CharacteristicNominalId>41</CharacteristicNominalId>
</DiameterCharacteristicItem>
<DiameterCharacteristicItem id="44">
  <Name>Hole_3_diam</Name>
  <FeatureItemIds n="1">
    <Id>29</Id>
  </FeatureItemIds>
  <CharacteristicNominalId>41</CharacteristicNominalId>
</DiameterCharacteristicItem>
<DiameterCharacteristicItem id="45">
  <Name>Hole_4_diam</Name>
  <FeatureItemIds n="1">
    <Id>30</Id>
  </FeatureItemIds>
  <CharacteristicNominalId>41</CharacteristicNominalId>
</DiameterCharacteristicItem>
```

(Characteristic designators have been removed from the characteristic items in the above example.)

In the context of a single part measurement this connection between the characteristic item and feature item would be sufficient to imply the connection between the single characteristic measurement and single feature measurement. But, if the results of more than one part measurement are included in a single QIF document this connection is no longer sufficient. Therefore, to explicitly make the connection between a characteristic measurement and the measured feature or features to which it applies the **FeatureMeasurementIds** *element* is used.

From the example, in a QIF XML instance file containing both characteristics and features, the connections between characteristic measurements and feature measurements might look like this:

```
<DiameterCharacteristicMeasurement id="46">
  <Status>
    <CharacteristicStatusEnum>PASS</CharacteristicStatusEnum>
  </Status>
  <CharacteristicItemId>42</CharacteristicItemId>
  <FeatureMeasurementIds n="1">
    <Id>31</Id>
  </FeatureMeasurementIds>
  <Value>10.003</Value>
</DiameterCharacteristicMeasurement>
<DiameterCharacteristicMeasurement id="47">
  <Status>
    <CharacteristicStatusEnum>PASS</CharacteristicStatusEnum>
```



```
    </Status>
    <CharacteristicItemId>43</CharacteristicItemId>
    <FeatureMeasurementIds n="1">
      <Id>32</Id>
    </FeatureMeasurementIds>
    <Value>10.005</Value>
  </DiameterCharacteristicMeasurement>
  <DiameterCharacteristicMeasurement id="48">
    <Status>
      <CharacteristicStatusEnum>PASS</CharacteristicStatusEnum>
    </Status>
    <CharacteristicItemId>44</CharacteristicItemId>
    <FeatureMeasurementIds n="1">
      <Id>33</Id>
    </FeatureMeasurementIds>
    <Value>9.996</Value>
  </DiameterCharacteristicMeasurement>
  <DiameterCharacteristicMeasurement id="49">
    <Status>
      <CharacteristicStatusEnum>FAIL</CharacteristicStatusEnum>
    </Status>
    <CharacteristicItemId>45</CharacteristicItemId>
    <FeatureMeasurementIds n="1">
      <Id>34</Id>
    </FeatureMeasurementIds>
    <Value>10.007</Value>
  </DiameterCharacteristicMeasurement>
```

5.9.5.2 Connections at various stages of the metrology process

The diagrams in this clause illustrate the various possible connections among feature aspects, among characteristic aspects, and between characteristic aspects and feature aspects at various stages of the metrology process when feature data is present.

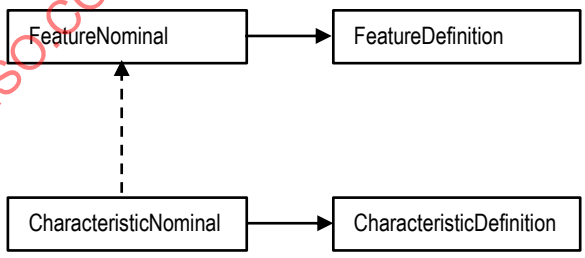


Figure 11 – Connections at the PMI Stage

Figure 11 shows the possible connections at the product manufacturing information stage of the metrology process. In this stage all information necessary to associate PMI (dimensions and tolerances, etc.) with features on a part is known. The dotted arrow between CharacteristicNominal and FeatureNominal indicates that the associated QIF *element* (**FeatureNominalIds**) is an optional *element*. It is nevertheless required for this use case in order to associate a characteristic with the feature or features to which it applies.

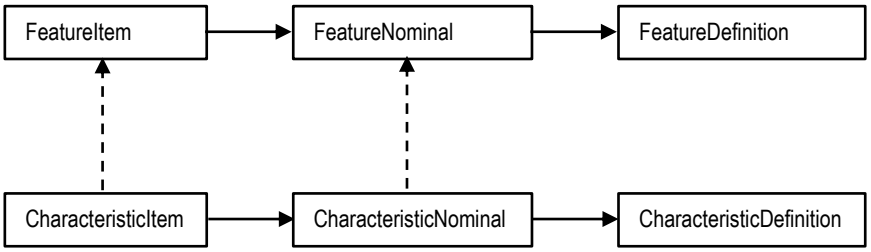


Figure 12 – Connections at the Planning Stage

Figure 12 shows the additional connections that come into being at the planning stage of metrology. Existing connections from the PMI stage persist and remain unchanged. Feature items and characteristic items which represent individual feature measurements and characteristic evaluations reference the data from the PMI stage. Again dotted arrows represent optional elements which are nonetheless required at the planning stage.

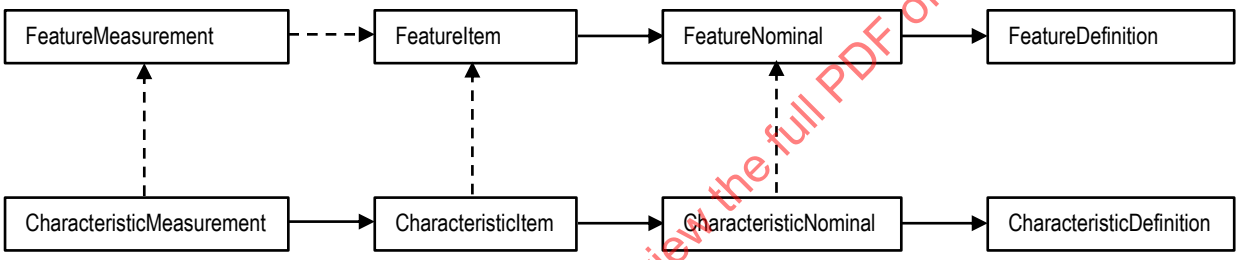


Figure 13 – Connections at the post-measurement stage

Figure 13 shows the new connections that arise at the post measurement stage. The feature and characteristic information and connections from the planning and PMI stages persist and remain unchanged and additional connections between measured characteristics and the measured features to which they apply and between measurements and their associated items are made. Again dotted arrows represent optional elements which are nonetheless required at the post-measurement stage.

Figure 13 shows the post-measurement connections when both feature and characteristic data are present. If only characteristic information is present then Figure 9 shows the connections between the four characteristic aspects at the post-measurement stage. Similarly, if only feature information, both nominal and measurement, is present then Figure 5 shows the connections between the four feature aspects at the post-measurement stage.

From the solid arrows in Figure 9 we see that characteristic nominal information is necessary when characteristic measurement information is present. The dotted arrow in Figure 5 and on the top row of Figure 13 shows that this is not the case with feature information. In the reverse engineering use case scenario, features may be measured without nominal information, often with the goal of creating a drawing or electronic model for a part where one does not exist.

5.9.5.3 Parent Features

The **FeatureItem** *element* (defined in the ***FeatureItemBaseType***) contains an optional sub-*element* called **ParentFeatureItemId**. The purpose of this *element* is to connect a feature item created after the initial planning stage with an existing feature in a measurement plan.

Occasionally, a feature item in a measurement plan cannot be measured when the plan is to be executed and a new derivative feature is measured in its place. Some examples include:

- A cylinder in a thin walled material, perhaps with break-away from a punching operation, is measured as a circle.
- A part rests on a datum feature making it inaccessible and the surface plate is measured to simulate the datum surface.
- The edge of a sheet-metal part is inaccessible with available probe configurations so a gage block is held against the edge and measured.
- A countersink is not measured directly but instead a spherical tooling ball is placed in the hole and measured.

5.10 Hierarchy of required information

5.10.1 QIF use of optional elements

The QIF information model contains many optional data elements. Sometimes these elements represent data for special cases which do not arise in most applications. But mostly, data elements have been made optional to allow for a variety of use cases while avoiding the possibility of the necessity to create fake or dummy data for a required data element that is outside the scope of a particular use case. For example, a caliper can only measure the size of an actual cylinder feature so required measured XYZ center location and IJK vector orientation elements for the actual cylinder would be outside the scope of the caliper-based use case.

There are situations where different use cases require that all data elements of a QIF type be optional. A caliper can measure the size of a hole but not the location or orientation, and a manual CMM with a conical hard probe can measure the location and orientation but not the size of a hole. Therefore, the location, orientation and size are all optional elements of the cylinder measurement in QIF. The presence of such optional data in the QIF information model in no way indicates that the optional data is unimportant.

Wherever possible, data elements in QIF that are co-requisite are placed together in a data type so that all the data that is required is mandatory. An *element* of that data type may be optional. The result is that either all or none of the data is present. The QIF feature and characteristic aspects which isolate sharable information like size from non-sharable information like location sometimes result in co-requisite information appearing in two different data types. In such cases the annotation describing the data type will make reference to any co-requisite data elements.

The hierarchies of required information for various aspects of the QIF information model are discussed in the individual subclauses pertaining to those aspects.

5.10.2 Example: diameter characteristic

Consider the measurement of the diameter of a hole. The diameter might have a nominal representation and an associated tolerance which needs to be evaluated: or the diameter might be

measured for the purposes of reverse engineering. Several examples are provided to illustrate how various levels of detail can be accommodated in QIF.

Common to the examples where a diameter is specified with a tolerance are the characteristic nominal and definition. Consider the diameter specification from Figure 10: ⌀10±.005. The tolerance is contained in the characteristic definition:

```
<DiameterCharacteristicDefinition id="40">
  <Tolerance>
    <MaxValue>0.005</MaxValue>
    <MinValue>-0.005</MinValue>
    <DefinedAsLimit>false</DefinedAsLimit>
  </Tolerance>
</DiameterCharacteristicDefinition>
```

and the nominal is contained in the characteristic nominal as described in subclause 5.9.3.

```
<DiameterCharacteristicNominal id="41">
  <CharacteristicDefinitionId>40</CharacteristicDefinitionId>
  <TargetValue>10</TargetValue>
</DiameterCharacteristicNominal>
```

5.10.2.1 Measurement with a hard gage

Suppose that a hard gage is used to determine the tolerance condition. The gage consists of two pins: a smaller pin which must fit in the hole and a larger pin that must not. The result of the inspection is a simple pass/fail evaluation.

The actual results of the measurement might appear as follows in a QIF XML instance file:

```
<DiameterCharacteristicMeasurement id="46">
  <Status>
    <CharacteristicStatusEnum>PASS</CharacteristicStatusEnum>
  </Status>
  <CharacteristicItemId>42</CharacteristicItemId>
</DiameterCharacteristicMeasurement>
```

The actual diameter is not known and the optional **Value** *element* does not appear in the instance file.

5.10.2.2 Measurement with a caliper

If the same hole is measured with a caliper not only can the tolerance condition be determined but the actual diameter is available.

The actual results of the measurement might appear as follows in a QIF XML instance file:

```
<DiameterCharacteristicMeasurement id="46">
  <Status>
    <CharacteristicStatusEnum>PASS</CharacteristicStatusEnum>
  </Status>
  <CharacteristicItemId>42</CharacteristicItemId>
  <Value>10.003</Value>
</DiameterCharacteristicMeasurement>
```


5.10.2.3 Measurement with a coordinate measuring machine

If the same hole is measured with a coordinate measuring machine then even more data about the size can become available. In addition to the actual best-fit diameter, the minimum and maximum diameters might be reported.

The actual results of the measurement might appear as follows in a QIF XML instance file:

```
<DiameterCharacteristicMeasurement id="46">
  <Status>
    <CharacteristicStatusEnum>PASS</CharacteristicStatusEnum>
  </Status>
  <CharacteristicItemId>42</CharacteristicItemId>
  <Value>10.003</Value>
  <MaxValue>10.004</MaxValue>
  <MinValue>10.001</MinValue>
</DiameterCharacteristicMeasurement>
```

5.10.2.4 Reverse engineering

If the same hole is measured with a caliper but no nominal information or tolerance information is available then the measurement results must be captured as a feature rather than as a characteristic.

The actual results of the measurement might appear as follows in a QIF XML instance file:

```
<CylinderFeatureMeasurement id="31">
  <FeatureItemId>27</FeatureItemId>
  <Diameter>10.003</Diameter>
</CylinderFeatureMeasurement>
```

In this example the optional **FeatureItemId** *element* is present. The feature item will contain the feature name and material disposition (inner or outer).

5.10.2.5 Limit dimensions

A characteristic may be described by specification limits like `10.005/9.995` in which case the example diameter characteristic definition might appear as follows in a QIF XML instance file:

```
<DiameterCharacteristicDefinition id="40">
  <Tolerance>
    <MaxValue>10.005</MaxValue>
    <MinValue>9.995</MinValue>
    <DefinedAsLimit>true</DefinedAsLimit>
  </Tolerance>
</DiameterCharacteristicDefinition>
```

Because the size is defined by the limits, the target nominal diameter need not be specified, in which case the example diameter characteristic nominal might appear as follows in a QIF XML instance file:

```
<DiameterCharacteristicNominal id="41">
  <CharacteristicDefinitionId>40</CharacteristicDefinitionId>
</DiameterCharacteristicNominal>
```


Note that the optional **TargetValue** *element* is not present.

5.10.2.6 Basic or theoretically exact dimensions

A characteristic may be described with a basic or theoretically exact dimension like 10, a nominal without tolerance limits. If a basic or theoretically exact diameter is to be measured then the diameter characteristic definition might appear as follows in a QIF XML instance file:

```
<DiameterCharacteristicDefinition id="40">
  <DimensionType>BASIC_OR_TED</DimensionType>
  <NonTolerance>MEASURED</NonTolerance>
</DiameterCharacteristicDefinition>
```

The diameter characteristic nominal target value would specify the basic or theoretically exact dimension:

```
<DiameterCharacteristicNominal id="41">
  <CharacteristicDefinitionId>40</CharacteristicDefinitionId>
  <TargetValue>10</TargetValue>
</DiameterCharacteristicNominal>
```

The diameter might be measured with a caliper but with no specification limits, no tolerance evaluation can be performed. The results of a measurement of a basic or theoretically exact diameter might appear as follows in a QIF XML instance file:

```
<DiameterCharacteristicMeasurement id="42">
  <Status>
     <CharacteristicStatusEnum>BASIC_OR_TED</CharacteristicStatusEnum>
  </Status>
  <CharacteristicNominalId>40</CharacteristicNominalId>
  <Value>10.003</Value>
</DiameterCharacteristicMeasurement>
```

5.10.3 ScaleCoefficient

If the **QIFDocument/QIFProduct/Header** portion of an instance file includes a **ScaleCoefficient** *element*, all elements representing distances in the **QIFDocument/QIFProduct** and **QIFDocument/Features** portions of an instance file are scaled. The default value of the **ScaleCoefficient** is 1. The type of all values to be scaled is either ***LinearValueType*** or ***xs:double***. To convert such values in an instance file to actual values, divide by the **ScaleCoefficient** value. To record actual values in an instance file, multiply the actual value by the **ScaleCoefficient** value.

5.11 Actual parts and assemblies

The QIF data model is able to represent both nominal models of products and actual instances of products. A detailed description of the QIF model for nominal product definition including parts and assemblies can be found in subclause 7.4 Product Structure. What follows is an overview of the QIF model structure for the purpose of connecting actual data to nominal product definitions. The nominal product model does not include "nominal" in the names of its terms. The terms used in the actual product model do include "actual". The nominal product model is in the **Product** *element* of the QIF document, while the actual model is in the **Results** *element*. The actual model includes references to the nominal model.

The QIF data model represents a (nominal) product as one or more (nominal) components. Conceptually, a component is a (nominal) instance of a part or assembly placed into the context of the final product or a higher order assembly. In the data structure, the component just references the part or assembly so that data is not needlessly duplicated. The root component represents the whole, final product and is the root of a tree of components representing the assembly structure. An assembly path defines the path from the root component to an individual component in a product design. This tree structure allows for each part or sub-assembly to be defined once and then be placed into the final product or sub–assembly with a transformation matrix. A product consisting of a single part will have one component.

Each assembly path has a real world counterpart in the actual product. For example the concept of a “right rear wheel” exists both in a (nominal) automobile design and on an actual automobile. Therefore when measured data is associated with the product definition it is by reference to this assembly path. An assembly path may be shared by many objects in a QIF document. Therefore, assembly paths are stored in one place: the **AsmPaths** *element*. Each assembly path has a QIF **id** by which it is referenced.

The simplest example showing the connection between an actual product and the product design is that of a single part. The product definition might be:

```
<Product>
  <PartSet n="1">
    <Part id="1">
      <Name>WING_MIR_REENF</Name>
      …
    </Part>
  </PartSet>
  <ComponentSet n="1">
    <Component id="2">
      <Part>
        <Id>1</Id>
      </Part>
    </Component>
  </ComponentSet>
  <AsmPaths n="1">
    <AsmPath id="3">
      <ComponentIds n="1">
        <Id>2</Id>
      </ComponentIds>
    </AsmPath>
  </AsmPaths>
</Product>
```

A single part is defined which is used in a single component. The assembly path to that component contains one **id**.

It is the assembly path of the product design component that is associated with the actual component. Actual components are located in the **ActualComponentSet** *element* of the **Results** *element*:

```
<Results>
    …
    <ActualComponentSet n="1">
      <ActualComponent id="4">
        <SerialNumber>X124578-17</SerialNumber>
```

```
        <Status>
          <InspectionStatusEnum>PASS</InspectionStatusEnum>
        </Status>
        <AsmPathId>3</AsmPathId>
      </ActualComponent>
    </ActualComponentSet>
    …
  </Results>
```

Each actual component has a QIF **id** which allows it to be referenced by one or more measurement results (the same product sample may be measured more than once) and/or by individual feature measurements:

```
<Results>
   …
   <MeasurementResults id="86">
      …
      <MeasuredFeatures>
        <FeatureMeasurements>
          <PointFeatureMeasurement id="20">
            <FeatureItemId>19</FeatureItemId>
            <Location>2466.9 774.31 944.84</Location>
            <ActualComponentId>4</ActualComponentId>
          </PointFeatureMeasurement>
          …
      </MeasuredFeatures>
      <ActualComponentIds n="1">
        <Id>4</Id>
      </ActualComponentIds>
   </MeasurementResults>
   …
</Results>
```

5.12 Checking connections between data objects

In QIF instance files, many connections between objects are made using an identifier and a reference to the identifier. For example the connection from a feature measurement to a feature item is made by putting an identifier in the feature item and a reference to the identifier in the feature measurement. The derivation hierarchy of ids and references to ids is shown in Figure 14.

Figure 14 – QIF id and reference types (below)

xs:unsignedInt
QIFIdAndReferenceBaseType
QIFIdType
QIFReferenceBaseType
QIFReferenceSimpleType
QIFReferenceType
QIFReferenceActiveType
QIFReferenceFullType

All local (i.e. not significant outside the instance file) identifiers in the QIF schemas are of type ***QIFIdType***. All references to local identifiers are derived from type ***QIFReferenceBaseType***. Both ***QIFIdType*** and ***QIFReferenceBaseType*** are unsigned positive integers without leading zeros. For

example, 1 and 3079 are valid, but 001 and +3079 are not. For use between instance files, the ***QIFReferenceType*** and its derived types may have, in addition, an **xId** *attribute*. The ***QIFReferenceActiveType*** has, in addition, a boolean **active** *attribute* indicating whether the reference is active. The ***QIFReferenceFullType*** has an **asmPathId** *attribute*. References that are significant between instance files are discussed in the subclause 5.13.

Arrays of ids are defined in the ***StatsArrayIdType***, ***ArrayReferenceType***, ***ArrayBinaryQIFReferenceType***, ***ArrayReferenceActiveType***, ***ArrayReferenceFullType***, and ***ArrayBinaryQIFReferenceFullType***.

In a QIF XML instance file, the reference contains the same symbol as the identifier of the object being referenced. For example, if the value of the **id** attribute of an instance of ***CylinderFeatureItemType*** is 26 then the value of the **FeatureItemId** *element* of an instance of ***CylinderFeatureMeasurementType*** that uses that feature item is also 26.

QIF seeks to ensure that connections made using identifiers and references join the correct types of objects. For example, a reference from a cylinder feature measurement to its item must identify a cylinder feature item, not any other type of object such as a transformation or a cone feature definition.

To ensure that identifier/reference pairs make matches between objects of the correct types, the QIF schemas contain several hundred *key*/*keyref* pairs, one for each variety of identifier/reference pair. *Key* and *keyref* are standard parts of the XML schema definition language (XSDL). Readily available XML instance file checkers will check whether or not *key*/*keyref* constraints are satisfied in an instance file governed by an XML schema. A detailed description of how *key* and *keyref* work may be found in books about the XML schema language such as [2]. In simplistic terms, a *key*/*keyref* pair locates two places in an (upside down) tree of objects that must contain identical information items. The two places are identified by describing the two paths that go downward to them from a common starting point. The *key*/*keyref* pair is located in a schema file at the common starting point.

For example, the CylinderFeatureMeasurementToItemKeyref:

- is located in the **QIFDocument** *element* (the common starting point)
- references the CylinderFeatureItemKey
- has the xpath "t:**Results**/t:**MeasurementResultsSet**/t:**MeasurementResults**/ t:**MeasuredFeatures**/t:**FeatureMeasurements**/t:**CylinderFeatureMeasurement**"
- has the field **FeatureItemId**.

The "t:" used with each *element* name in the xpath is a prefix standing for the QIF namespace and is required by the rules of XSDL in xpaths in any schema that uses a namespace. The header of each QIF schema file declares that the "t:" prefix will be used for the http://qifstandards.org/xsd/qif3 namespace. Almost any other prefix would have worked as well; "t:" is nice and short.

The CylinderFeatureItemKey:

- is also located in the **QIFDocument** *element* (the common starting point)
- has the xpath "t:**Features**/t:**FeatureItems**/t:**CylinderFeatureItem**", and
- has the field **@id**

The @ sign means the **id** field is an *attribute* rather than an *element*.

This sounds complex, but if we look at an instance file, it's fairly straightforward. Here is a snippet from an abbreviated QIFDocument instance file.

```
<QIFDocument …>
  …
  <Results …>
    <MeasurementResultsSet n=”1”>
      <MeasurementResults …>
        …
        <MeasuredFeatures>
          <FeatureMeasurements n=”30”>
            <CylinderFeatureMeasurement id="53">
              <FeatureItemId>26</FeatureItemId>
                <Axis> … </Axis>
              <Diameter>2.5</Diameter>
            </CylinderFeatureMeasurement>
            <CylinderFeatureMeasurement id="54">
              <FeatureItemId>27</FeatureItemId>
              <Axis> … </Axis>
              <Diameter>2.0</Diameter>
            </CylinderFeatureMeasurement>
            …
          </FeatureMeasurements>
        </MeasuredFeatures>
      </MeasurementResults>
    <MeasurementResultsSet n=”1”>
  </Results>
  …
  <Features>
    <FeatureItems n=”30”>
      <CylinderFeatureItem id="26">
        <FeatureNominalId>33</FeatureNominalId>
        <FeatureName>top inner cylinder</FeatureName>
        …
      </CylinderFeatureItem>
      <CylinderFeatureItem id="27">
        <FeatureNominalId>34</FeatureNominalId>
        <FeatureName>'dial' outer cylinder</FeatureName>
        …
      </CylinderFeatureItem>
    </FeatureItems>
  </Features>
  …
</QIFDocument>
```


When a *key/keyref* checking system reads this instance file and comes to the `<QIFDocument>` part, it knows that it needs to check the **CylinderFeatureItemKey** and the **CylinderFeatureMeasurementToItemKeyref** (since they are located in the **QIFDocument** *element*). To check the *key*, the checker follows every **Features/FeatureItems/CylinderFeatureItem** path down to the **id** *attribute*. In the snippet above, there are two such paths, one ending in “26”, the other ending in “27”. The checker puts the two **id**s into a collection of **id**s of **CylinderFeatureItem**s, checking to make sure that each of these is different from those already in the collection. To check the *keyref*, the checker follows every

Results/ MeasurementResultsSet/MeasurementResults/MeasuredFeatures/ FeatureMeasurements/CylinderFeatureMeasurement path down to the **FeatureItemId** *element*. In the snippet above, there are two such paths with *elements*, one with the value 26 and the other with value 27. The checker checks that each of the values is in its collection of **id**s of **CylinderFeatureItem**s. The snippet above passes these checks.

The *key/keyref* pair mechanism catches errors in matching identifiers and references, but it is not foolproof. In all cases, when there are many objects of the same type, no automated check (such as *key/keyref*) will know which one is intended. For example, if an instance file has several instances of ***CylinderFeatureItemType*** and several instances of ***CylinderFeature*****MeasurementType** (as in the snippet above), only the builder of the instance file will know which item (26 or 27 in the snippet) is supposed to go with which measurement. In other cases, there is no way to define a *key* that discriminates between similar types of object because they are mixed together at the same location (in a set of transformations containing both actual transformations and nominal transformations, for example).

The *key/keyref* mechanism is useful primarily within a single file. Most of the keyrefs in QIF allow a reference to the internal **id** of an external file as an alternative to the **id** of an internal object. This is described in the following subclause. In the case of a reference to an object in an external file, the *key/keyref* mechanism does not provide an effective check on the type of the object in the external file. The XSLT checks (as described in subclause 5.4.1) include a check that a referenced object with a given **id** exists in an external file, but that particular check does not include checking that the external object is of the correct type. In addition, all the connections between aspects of features in external files and between aspects of characteristics in external files are checked by the XSLT checks. It is planned to augment the XSLT files so that all *key/keyref* checks within a file have XSLT counterparts that check between files.

5.13 Tracking information through the product lifecycle

QIF is constructed to enable a seamless flow of information from upstream applications to downstream applications and to enable tracking information through a product's lifecycle.

5.13.1 Persistent Identifiers

The primary mechanism for identifying the same information independent of location and instance is by the use of permanent persistent identifiers (IDs). A persistent ID is an unchanging ID assigned to an object which can be used to identify the object unambiguously.

Any item in QIF that can have user defined attributes can also be assigned a persistent ID.

5.13.2 UUIDs and QPIds

Persistent IDs provide a mechanism for QIF to preserve connections to data objects in external QIF files and data objects in the world outside of QIF. The main QIF data object types can have persistent IDs added as user defined attribute data. The data types of user attributes applicable to persistent ID definition could include an integer value, a string value, or a QPId value. QPID (pronounced "cupid") is a short form of QIF Persistent Identifier. A QPId is the QIF implementation of a universally unique identifier (UUID), as standardized in ISO/IEC 9834-8. UUIDs have that name because the chance of randomly generating two identical UUIDs is exceedingly low (one chance in about 10^{38}). Computer libraries for generating UUIDs conforming to the standard are widely available in many computer languages.

Using UUIDs, non-communicating systems can identify information uniquely. That information can be combined later into a single application or database without needing to resolve identifier conflicts.

As a number, a UUID is a 128 bit unsigned integer. As a text string in an instance file, a UUID is represented by 32 hexadecimal digits displayed in five groups separated by hyphens in the form 8-4-4-4-12 for a total of 36 characters (32 alphanumeric characters and four hyphens). An example of a UUID string is 550e8400-e29b-0518-a716-445664449c0b. The letters a through f are hexadecimal digits representing the numbers 10 through 15. Either lower case letters or upper case letters may be used in QIF for those digits.

The null UUID, which is equivalent in practice to no UUID since it is not unique, is one that has all 128 bits set to zero. In text form, that is written 00000000-0000-0000-0000-000000000000.

The ***QPIdType*** in QIF is required to have UUID format, and an instance of ***QPIdType*** must be created by a well known UUID generator. An *element* whose type is ***QPIdType*** is often named **QPId** or **UUID**. Where a QPId is used to identify a data object or a file, its data type is ***QPIdType***. Where a QPId is used to reference a data object or a file, its data type is ***QPIdReferenceType***, and it must match an existing QPId.

5.13.3 External File References

Typically, a persistent ID is assigned to an item in a QIF document as a link to an object in the world outside of QIF. A more lightweight method of referencing items inside a particular external QIF document is made available by external file references.

The external QIF file reference mechanism is enabled by the fact that each QIF document must have a QPId identifying the document and by the fact that all QIF **id**s within a document must be unique. Thus the document-QPId/item-QIF-**id** pair can be used to uniquely identify a QIF item.

In order 1) to avoid using the external document QPId in each reference, 2) to make the external QIF file reference mechanism light weight, and 3) to enable automated *key*/*keyref* checking and XSLT checking, an alias, which can be referenced by a simple QIF **id**, is defined in the **ExternalQIFReferences** *element* of the QIF document.

Each **ExternalQIFDocument** *element* of type ***ExternalQIFDocumentReferenceType*** captures the QPId of the external QIF document in the **QPId** *element* and has a QIF **id**. Optionally, the URL of the external file and/or its description can be specified.

External QIF file referencing is best illustrated by example. Consider a diameter characteristic measurement which must reference a diameter characteristic item. In a single QIF document this might look like:

```
<QIFDocument …>
  <QPId>DF837194-1E55-441b-A5D9-9279F56BCBE9</QPId>
  …
  <DiameterCharacteristicItem id=”15”>
  …
  </DiameterCharacteristicItem>
  …
  <DiameterCharacteristicMeasurement id=”47”>
  <CharacteristicItemId>15</CharacteristicItemId>
```



```
…
</DiameterCharacteristicMeasurement>
…
```

In the snippet above, the measurement aspect references the item with **id** 15 in the same document.

Now, consider the case in the snippets below where the characteristic items are defined in an external QIF file. The external QIF file might include the following snippet.

```
<QIFDocument …>
  <QPId>D7258334-F9BF-4201-94FB-3D39EA74937B</QPId>
  …
  <DiameterCharacteristicItem id=”15”>
  …
  </DiameterCharacteristicItem>
  …
```

To reference the item **id** in the external QIF document, first, an external document reference must be established and then the external item is referenced using the **xId** *attribute*:

```
<QIFDocument …>
  <QPId>AFD08F25-CAE6-4647-9667-674D5D66033F</QPId>
  …
  <ExternalQIFReferences n=”1”>
    <ExternalQIFDocument id=”101”>
      <QPId>D7258334-F9BF-4201-94FB-3D39EA74937B</QPId>
    </ExternalQIFDocument>
  </ExternalQIFReferences>
  …
  <DiameterCharacteristicMeasurement id=”47”>
  <CharacteristicItemId xId=”15”>101</CharacteristicItemId>
  …
  </DiameterCharacteristicMeasurement>
  …
```

The `<ExternalQIFDocument id=”101”>` object identifies the external QIF file via its QPId and provides **id** 101 to enable referencing that file. The `xId=”15”` in the diameter characteristic measurement then points to the item **id** in the external QIF document whose internal **id** is 101.

5.13.4 QIF data flow

Version 3.0 of QIF has six application areas. A **QIFDocument** instance file can contain any one of the applications or any combination of them. The most natural combinations in a single instance file are those that contain a sequence along the workflow shown in Figure 2, for example, a **Product** *element* from the Define Product activity, a **QIFPlan** *element* from the Define Measurement Process activity, and a **Results** *element* from the Execute Measurement Process activity. Those *element*s would reference information common to all three of them contained elsewhere in the **QIFDocument,** such as:

- file units
- datum definitions and datum reference frames
- measurement resources
- feature definitions, nominals, and items

- characteristics, definitions, nominals, and items

At the time a downstream process is started, (when measurements are starting to be taken, for example), a **QIFDocument** instance file containing the output of an upstream planning process is likely to exist; call it the QIFPlan file The plan file may have a **QIFDocument** with **Product** and **Plan** *element*s as shown in the following instance file snippet.

```
<QIFDocument …>
  <QPId>AFD08F25-CAE6-4647-9667-674D5D66033F</QPId>
  …
  <Product>
    product contents
  </Product>
  <Plan>
    plan contents
  </Plan>
</QIFDocument>
```

When measurements are taken and it is desired to put the data in a **QIFDocument** instance file, the data must not simply be added to the QIFPlan file. A new **QIFDocument** instance file (call it the QIFResults file) should be created containing a **Results** *element*. Most *element*s of the **QIFDocument** in the QIFPlan file may be copied without change into the **QIFDocument** in the QIFResults file, but any **Version**, **VersionHistory**, or **Header** *element* must be new. The QIFResults file might appear as shown in the following snippet. Note the new **QPId**.

```
<QIFDocument …>
  <QPId>AFD08F25-CAE6-4647-9667-674D5D66071A</QPId>
  …
  <Product>
    product contents (same as in previous snippet)
  </Product>
  <Plan>
    plan contents (same as in previous snippet)
  </Plan>
  <Results>
    results data here
  </Results>
</QIFDocument>
```

The way in which local identifiers (**id**s) are handled when a new downstream file is created from an old upstream file is up to the application building the new file. The local identifiers from the upstream process might be preserved or they might be changed. A simple method of preserving **id**s from the QIFPlan file is to start numbering new **id**s in the QIFResults file at a value larger than the maximum **id** value in the QIFPlan file. In any event, no system processing QIF instance files should rely on local identifiers remaining the same between files.

5.13.5 Using QPIds in QIF

To provide a method of uniquely identifying QIFDocument instance files and objects in them, QPIds may optionally be assigned to the **QIFDocument** *element* and *elements* representing the six QIF applications as shown in Figure 15. To avoid confusing QPIds with **id**s, QPIds are always given as *element*s, and **id**s are always given as *attribute*s. The figure shows the *element* name. The value of

each of the *element*s listed is of ***QPIdType***. Where the *element* is in a base type, all of the derived types will also have the *element*.

QPIds used for interconnecting QIF information (in contrast to QPId user attribute identifiers) identify data on a coarse level: the whole document, or file sections describing a measurement plan, a measurements result, etc. The QIF **id** mechanism is used for identifying data at a fine level: an individual feature, characteristic, measurement point, etc.

QIFDocument
Required **QPId** *element* of the ***QIFDocumentType***
UUID *element* of the ***CharacteristicDesignatorType*** at several places in the **Characteristics** *element*
UUID *element* of feature nominals and feature items in the **Features** *element*

MeasurementResources
ThisInstanceQPId in the **Version** *element* of the ***MeasurementResourcesType***

Plan
ThisInstanceQPId in the **Version** *element* of the ***PlanType***

Product
ThisInstanceQPId in the **Version** *element* of the **File** *element* of the **Header** *element* of the ***ProductType***
ThisInstanceQPId in the **Version** *element* of the **File** *element* of the **Header** *element* of the ***PartType***
ThisInstanceQPId in the **Version** *element* of the **File** *element* of the **Header** *element* of the ***AssemblyType***
UUID *element* of the ***PartAssemblyBaseType*** and the ***ComponentType***

Results
ThisInstanceQPId in the **Version** *element* of the ***ResultsType***
ThisResultsInstanceQPId *element* of the ***MeasurementResultsType***

Rules
ThisInstanceQPId in the **Version** *element* of the ***QIFRulesType***
UUID *element* of the ***QIFRuleBaseType***

Statistics
ThisInstanceQPId in the **Version** *element* of the ***StatisticalStudyPlanBaseType***
ThisInstanceQPId in the **Version** *element* of the ***CorrectiveActionPlanType***
ThisStatisticalStudyResultsInstanceQPId *element* of the ***StatisticalStudyResultsBaseType***

Figure 15 – ***QPIdType*** elements

All uses of QPIds are optional with the exception of the required **QPId** *element* of the ***QIFDocumentType***. The document level QPId is required because uniquely identifying QIF documents is the only reliable method of connecting files across the workflow shown in Figure 2. Optional version QPIds may be used to strengthen connections. A *key* in QIFDocument.xsd (when

applied by an instance file processor) checks that all instances of ***QPIdType*** in an instance file have the correct format and are unique within the file.

When a downstream QIF instance file is created from an upstream QIF instance file, although the **QPId** of the document itself must be new, the QPIds identifying coarse elements in the QIF document may remain unchanged if the measurement plan, resources, etc. they identify are unchanged. If *element*s such as **characteristics**, **features**, and **product** are copied into the downstream file, any QPIds they contain (in QPId user attribute data, for example) should be copied unchanged. If any part of a file section identified by a QPId (other than local **id**s) is changed after copying, the QPId should be changed, too. A changed local **id** should result in a changed QPId only if the local **id** change results in a change in the structure obtained when all local **id** references have been resolved.

QPIds uniquely identify QIF instance files and objects in the files. In addition, references to QPIds may be used to connect sections of QIF instance files (in different files or in the same file) as shown in Figure 16. Where QPIds are used in QIFPlan.xsd they indicate the rules that were used in developing the plan and/or the rules to be used in refining the plan. Where QPIds are used in QIFStatistics.xsd, they serve to identify results from external QIF instance files. Where QPIds are used in Traceability.xsd they serve to identify the plan that was used for **MeasurementResults** and for **Statistics**. A reference to a QPId is always made using an instance of ***QPIdReferenceType***.

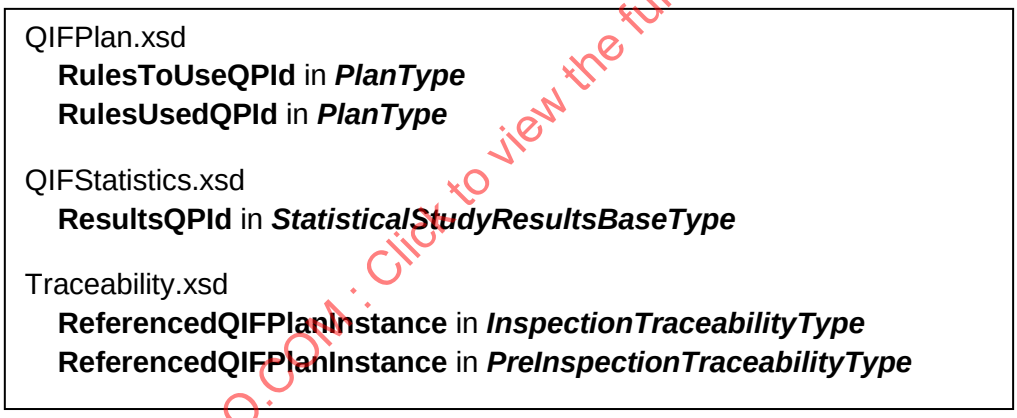


Figure 16 – ***QPIdFullReferenceType*** elements

The references shown in Figure 16 are made using the ***QPIdFullReferenceType*** which has two *element*s, each of which is of ***QPIdReferenceType***. The required **ItemQPId** *element* is the QPId of the referenced item. Each optional **DocumentQPId** *element* is the QPId (found in the **Version**) of a **QIFDocument** that contains the item. For example, the **RulesToUseQPId** in a measurement plan might be as follows.

```
<RulesToUseQPId>
    <ItemQPId>ed43400a-29bf-4ec6-b96c-e2f846eb6f00</ItemQPId>
    <DocumentQPId>fd43400a-29bf-4ec6-b96c-e2f846eb6ff6</DocumentQPId>
</RulesToUseQPId>
```

The **DocumentQPId** *element* is included so that the item will be easier to find. Adding an entry to the **ExternalQIFReferences** *element* associating the document QPId with its location defined by its URI further facilitates finding the item.

Another use of references to QPIds occurs in the **VersionHistory** of the ***QIFDocumentType***, which may be used to describe earlier versions of a QIF document. The information given there for each earlier version has an optional **QPIdReference** *element* of type ***QPIdReferenceType***.

5.14 Linking PMI information to product shape models

The QIF design provides for connecting detailed shape information in internal or external product models to features, characteristics, parts, and assemblies in a QIFDocument instance file. The ***PartType*** and the ***AssemblyType*** both have an optional **DefinitionInternal** *element* and an optional **DefinitionExternal** *element*.

Each part and assembly in a QIF instance file may correspond to an internal model. The internal model is as described in Clause 0.

Each part and assembly in a QIF instance file may correspond to multiple external models, for example, a CAD file and two drawings.

In the case of an external CAD model, the identifiers in the model must be at least minimally persistent in the sense that each time a given file is loaded into a system that can handle it, the same identifier should be attached to each information item as is attached any other time the file is loaded. It may be assumed that identifiers in a digital drawing that is not a CAD model are persistent.

QIF supports the following types of external models for parts and assemblies in the *element*s of the **DefinitionExternal** *element*.

Parts
- ***PrintedDrawingType***
- ***DigitalDrawingType***
- ***DigitalModelType***
- ***PhysicalModelType***

Assemblies
- ***PrintedDrawingType***
- ***DigitalDrawingType***
- ***DigitalModelType***
- ***PhysicalModelType***

To connect digital drawing or digital model entities to QIF features and characteristics, the digital entities must be identified, and the association from a feature or characteristic to one or more digital entities must be made. Printed drawings and physical models do not have entity identifiers that could be connected to features or characteristics.

Identifying external digital entities is done in the **DefinitionExternal** *element*, which is of ***DefinitionExternalType***. The **DigitalDrawing** and **DigitalModel** *element*s of the ***DefinitionExternalType*** each have a subordinate **Entities** *element* that is a list of **Entity** *element*s of ***EntityExternalType***. The ***ExternalEntityType*** has **EntityId**, **Name**, and **Description** *element*s and an **id** *attribute* of ***QIFIdType***. The value of the **EntityId** is a string that is a persistent identifier in the external model. That string is created by the creator of the external model, not the author of the QIF file. The **id** *attribute* is used to identify the entity in other parts of the QIF instance file. For each

information item in the external model that has a persistent identifier and is to be referenced elsewhere in the file, an **Entity** is defined.

The connection of an external digital entity to a feature or characteristic is made in the nominal feature or nominal characteristic, both of which contain an **EntityExternalIds** *element* that is a list of the QIF **id**s for the associated external entities.

Internal entities are connected to features and characteristics in exactly the same way, except that the **id**s of the internal entities are in an **EntityInternalIds** *element*.

An abbreviated **Part** section from a QIF instance file is shown below. It has one model (with **id** 64). The snippet shows two **Entity**s (with **id**s 68 and 69). One of those refers to the **EntityId** "TopDistanceRightPlane", and the other refers to the **EntityId** "TopDistanceLeftPlane".

```
<Part>
  …
  <DefinitionExternal>
    …
    <DigitalModel id="64">
      …
      <Entities n=”2”>
        <Entity id="68">
          <EntityId>TopDistanceRightPlane</EntityId>
        </Entity>
        <Entity id="69">
          <EntityId>TopDistanceLeftPlane</EntityId>
        </Entity>
      </Entities>
    </DigitalModel>
  </DefinitionExternal>
</Part>
```

The association from a feature to one or more CAD entities is made in the **Features** section of a QIF instance file (and similarly for characteristics). This is done by listing the **EntityExternalId**s that correspond to each feature instance. Here is a snippet from a QIF instance file in which a single opposite parallel planes feature references the planes in the preceding snippet.

```
<Features>
 …
  <FeatureNominals n=”30”>
   …
    <OppositeParallelPlanesFeatureNominal id="34">
     …
       <EntityExternalIds n=”2”>
        <Id>68</Id>
        <Id>69</Id>
      </EntityExternalIds>
     …
    </OppositeParallelPlanesFeatureNominal>
   …
```



```
  </FeatureNominals>
</Features>
```

5.15 Welding Characteristics and Symbols

The QIF library contains a set of characteristic types that provide complete welding information on a model as shown in Figure 17.

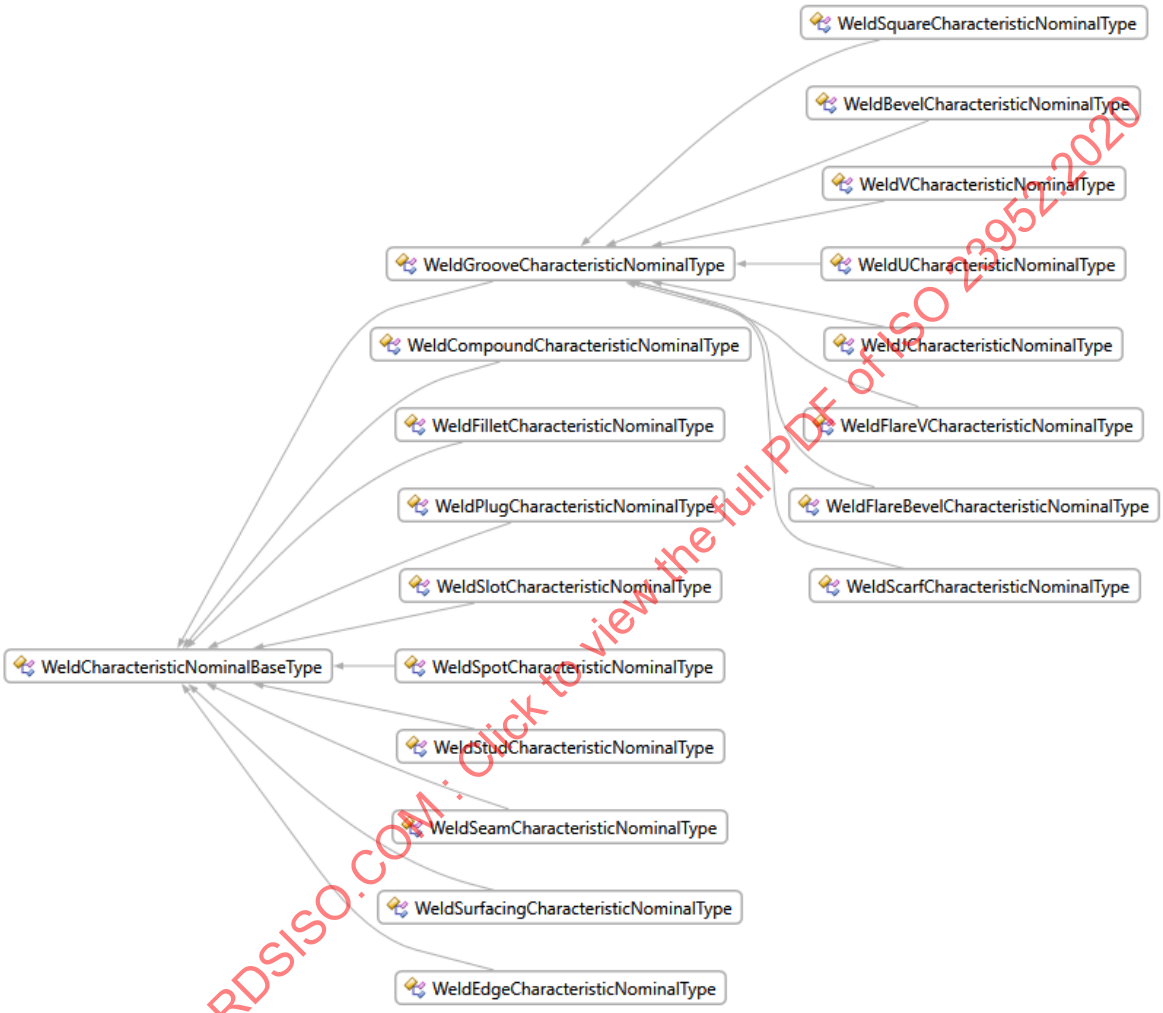


Figure 17 – Weld Characteristics Hierarchy

The weld characteristic types are implemented in correspondence with the American Welding Society (AWS) standard A2.4:2012 “Standard Symbols for Welding, Brazing, and Nondestructive Evaluation”. That standard describes both the symbols and their meaning. QIF models the meaning of the weld symbols in the AWS standard as characteristics. The appearance of the weld symbols in a graphical presentation of weld characteristics must be as prescribed in the AWS standard. The meanings of the weld characteristics are described in this subclause.

The appearance of the weld symbols is described here and also in subclause 7.5.7.5.16, which deals with the details of graphical presentation. A weld symbol consists of at least a horizontal

reference line with an arrow angled 45 degrees up or down at one end, as shown in Figure 18, Figure 20, Figure 21, Figure 22, and Figure 23. The symbol may also have several other graphical devices and meaningfully placed text.

5.15.1 Base parameters

In QIF, all weld characteristic types have the following base parameters:

Field Name	Data Type	Description
AllAround	xs:boolean	This *element* instructs to weld on all sides of the joint.
Field	xs:boolean	This *element* instructs to weld on a job site, not in the welding shop.
Specification	xs:token	This *element* contains a specification that describes a specific welding process. This note should use the standard welding abbreviations such as AAW (Air Acetylene Welding), AB (Arc Blazing), AHW (Atomic Hydrogen Welding), ..., TC (Thermal Cutting).
WeldingProcess	WeldingProcessType	This optional *element* defines a welding process.
NonDestructiveTesting	ArrayNonDestructiveTestingType	This optional *element* contains an array of non-destructive tests.

5.15.2 Location Significance parameter

Location Significance defines the location of a weld. A weld characteristic can be one-sided (placed on the arrow side or on the other side of the reference line) or both-sided, as shown by the weld symbols in Figure 18. Each type of weld characteristic has a specific enumeration of the location significance values.

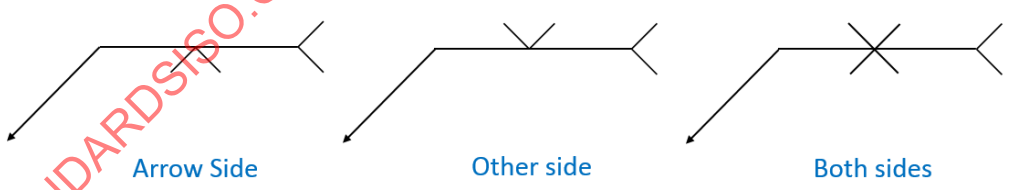


Figure 18 – Location Significance

5.15.3 Weld Characteristic parameters

The Weld characteristic parameters define welding dimension parameters (such as size, depth, pitch, length etc.) and finishing information (contour symbol and finishing designator). In QIF, the weld characteristic parameters are organized in the hierarchy shown in Figure 19.

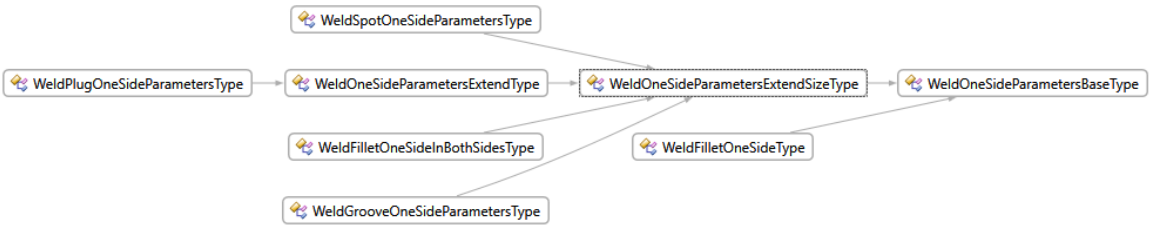


Figure 19 – Weld Characteristic Parameters Hierarchy

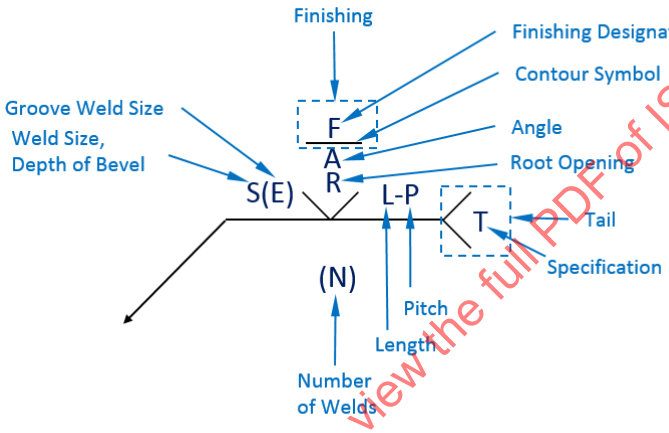


Figure 20 – Weld Characteristic Parameters

The weld characteristic parameters (from all types of weld characteristic parameters sets) and their meanings follow. Figure 20 shows where the parameter values are placed in a graphical presentation.

Parameter Name	Data Type	Description
Finishing	WeldFinishingType	This *element* defines finishing parameters of the weld.
Finishing/ContourSymbol	WeldContourSymbolEnumType	This *element* defines a supplementary contour symbol that is used with the weld symbols to indicate how the face of the weld should be finished.
Finishing/FinishingDesignator	WeldFinishingDesignatorEnumType	This *element* defines a finishing designator that is used to indicate the method for forming the contour of the weld.
Size	FractionType	This *element* defines the size/strength of a one-sided weld. In case of the bevel weld this parameter should be interpreted as its depth.
Groove	FractionType	This *element* defines a groove weld size.
Angle	AngularValueType	This *element* defines a groove angle or the angle of countersink for a plug weld.
RootOpening	FractionType	This *element* defines a root opening or the fillet depth for a plug weld.
Length	LinearValueType	This *element* defines the length of the weld.
Pitch	LinearValueType	This *element* defines the pitch of the weld.
Depth	FractionType	This *element* defines the depth of the filling.

Table 2 – Weld Parameters

Example:

```
<WeldFilletCharacteristicNominal id="10302" label="WELD_FILLET_10302">
  <FeatureNominalIds n="1">
    <Id>9630</Id>
  </FeatureNominalIds>
  <LocationSignificance>BOTH_SIDE</LocationSignificance>
  <BothSides>
    <ArrowSide>
      <Size>
        <Numerator>5</Numerator>
        <Denominator>16</Denominator>
      </Size>
      <LengthOfEachWeld>1</LengthOfEachWeld>
      <Pitch>3</Pitch>
    </ArrowSide>
    <OtherSide>
      <Size>
        <Numerator>5</Numerator>
        <Denominator>16</Denominator>
      </Size>
      <LengthOfEachWeld>1</LengthOfEachWeld>
      <Pitch>3</Pitch>
```



```
    </OtherSide>
    <Staggered>true</Staggered>
  </BothSides>
</ WeldFilletCharacteristicNominal>
```

5.15.4 Supplementary parameters

The following supplementary parameters may be added to the basic weld characteristics: spacer, back weld, melt through, consumable insert, removable backing. Figure 21 shows the graphical presentation of melt through. The figure is followed by a QIF instance file snippet giving an example of melt through.

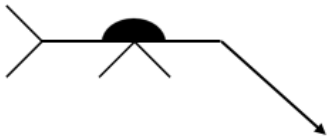


Figure 21 – Melt Through

Example:

```
<WeldVCharacteristicNominal id="10301" label="WELD_V_10301">
  <FeatureNominalIds n="1">
    <Id>9631</Id>
  </FeatureNominalIds>
  <OneSide>
    <LocationSignificance>ARROW_SIDE</LocationSignificance>
    <SupplementarySymbol>MELT_THROUGH</SupplementarySymbol>
  </OneSide>
</WeldVCharacteristicNominal>
```

5.15.5 Non-Destructive Testing types

Non-Destructive Testing is defined by a set of non-destructive testing types. In a graphical presentation, it is placed on the reference line or on the second reference line when multiple reference lines are used, as shown in Figure 22.

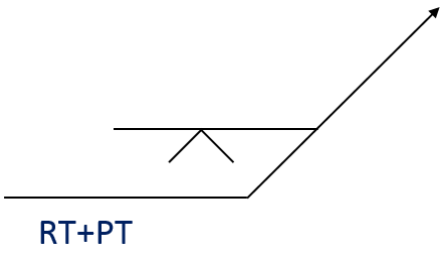


Figure 22 – Non-Destructive Testing with Multiple Reference Lines

5.15.6 Compound Welds

The Compound Weld characteristic defines a number of welds (up to 4) to be filled in the same welding operation sequence. An example of the graphic presentation is shown in Figure 23.

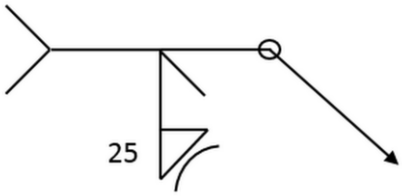


Figure 23 – Compound Weld

5.16 QIF handling of transforms, transformations, and coordinate systems

5.16.1 Coordinate Spaces

In QIF, the locations and orientations of all features, both nominal and measurements, are defined in three-dimensional, right-handed, Cartesian coordinate spaces.

The location and orientation of nominal features can be defined in the coordinate space of the product component to which they belong allowing nominal feature information to be defined once for a part and shared across multiple instances of the same part in an assembly. Components (parts and assemblies) are combined to form the product and in the process the nominal feature information undergoes coordinate transformations until it is in the coordinate space of the root component, i.e. that of the product itself. In many situations the product will consist of a single component in which case all nominal feature information is defined in a single coordinate system.

The coordinate space of the root component is analogous to the global or world coordinate system in a CAD system: it is the de facto Cartesian coordinate space to which all other Cartesian spaces can be related.

The location and orientation of measured features are defined in this root component coordinate space. Measured feature information can be transformed from this root component coordinate space into other coordinate spaces using transformation information stored in QIF.

5.16.2 Transformation matrix

A transformation matrix is used to map (X, Y, Z) coordinates and (I, J, K) unit vectors between Cartesian coordinate spaces. Component transformations map the location and orientation of a component to its location and orientation in an assembly.

In QIF a transformation matrix takes the form of a 4 by 3 matrix consisting of a 3 by 3 rotation matrix defining real number triplets representing the orientation of one coordinate space in another, and the origin of one coordinate space in another defined as a real number triplet. Both the rotation matrix and the origin are optional allowing for definions of rotation-only transforms, translation-only transforms, and the identity transform (no rotation or translation) as well as combined rotation and translation transforms.

XYZ points, IJK direction vectors, and transformation matrices of local coordinate spaces can all be defined in a common coordinate system. The origin of the transform is the XYZ location of the local coordinate space in the common coordinate space. Each row of the rotation matrix is the direction of an axis of the local coordinate space in the common coordinate space.

Such a transformation matrix is defined by the ***CoordinateSystemCoreType*** which has two *elements* defining the rotation (as a 3x3 matrix) and the origin (as a triplet):

Rotation: a 3x3 matrix representing the X, Y and Z axis directions of the local coordinate space in the common coordinate space having 3 sub-*elements*:

- **XDirection**: a unit vector defining the IJK direction of the X-axis direction of the local coordinate space in the common coordinate space.
- **YDirection**: a unit vector defining the IJK direction of the Y-axis direction of the local coordinate space in the common coordinate space.
- **ZDirection**: a unit vector defining the IJK direction of the Z-axis direction of the local coordinate space in the common coordinate space.

Origin: a point defining the origin of the local coordinate space in the common coordinate space.

The three vectors defined by the **XDirection**, **YDirection** and **ZDirection** *elements* are orthonormal: each is of unit length and each is perpendicular to the other two. Furthermore they are right-handed in the sense that the right-hand vector cross product of the vector defined by the **XDirection** *element* with the vector defined by the **YDirection** *element* is equal to the vector defined by **ZDirection** *element*, the right-hand vector cross product of the vector defined by the **YDirection** *element* with the vector defined by the **ZDirection** *element* is equal to the vector defined by **XDirection** *element*, and the right-hand vector cross product of the vector defined by the **ZDirection** *element* with the vector defined by the **XDirection** *element* is equal to the vector defined by **YDirection** *element*. Or in short:

$||\mathbf{Xdirection}|| = 1$

$||\mathbf{Ydirection}|| = 1$

$||\mathbf{ZDirection}|| = 1$

$\mathbf{XDirection} \cdot \mathbf{YDirection} = 0$

$\mathbf{YDirection} \cdot \mathbf{ZDirection} = 0$

$\mathbf{ZDirection} \cdot \mathbf{XDirection} = 0$

$\mathbf{XDirection} \times \mathbf{YDirection} = \mathbf{ZDirection}$

$\mathbf{YDirection} \times \mathbf{ZDirection} = \mathbf{XDirection}$

$\mathbf{ZDirection} \times \mathbf{XDirection} = \mathbf{YDirection}$

This results in a transformation which can translate and rotate but neither skew nor scale axis systems.

To map (or transform) a point in a local space with coordinates (x, y, z) to a point in the common space with coordinates (X, Y, Z) where the local space is defined by the transformation matrix with both rotation and translation:

$$\begin{matrix} X_i & X_j & X_k \\ Y_i & Y_j & Y_k \\ Z_i & Z_j & Z_k \\ \lfloor O_x & O_y & O_z \rfloor \end{matrix}$$

the following calculations are used:

$$X = (X_i)x + (Y_i)y + (Z_i)z + O_x,$$

$$Y = (X_j)x + (Y_j)y + (Z_j)z + O_y, \quad (1)$$

$$Z = (X_k)x + (Y_k)y + (Z_k)z + O_z.$$

The mapping of vectors between common and local spaces is similar. To map the vector with components (i, j, k) in local space to the vector with components (I, J, K) in common space the following calculations are used:

$$I = (X_i)i + (Y_i)j + (Z_i)k,$$

$$J = (X_j)i + (Y_j)j + (Z_j)k, \quad (2)$$

$$K = (X_k)i + (Y_k)j + (Z_k)k.$$

Conversely, to map points from common space into local space the inverse transformation is used. To map the point (X, Y, Z) in common space to the point (x, y, z) in local space the following calculations are used:

$$x = (X_i)\,(X-O_x) + (X_j)\,(Y-O_y) + (X_k)\,(Z-O_z),$$

$$y = (Y_i)\,(X-O_x) + (Y_j)\,(Y-O_y) + (Y_k)\,(Z-O_z), \quad (3)$$

$$z = (Z_i)\,(X-O_x) + (Z_j)\,(Y-O_y) + (Z_k)\,(Z-O_z).$$

And to map the vector (I, J, K) in common space to the vector (i, j, k) in local space the following calculations are used:

$$i = (X_i)I + (X_j)J + (X_k)K,$$

$$j = (Y_i)I + (Y_j)J + (Y_k)K, \quad (4)$$

$$k = (Z_i)I + (Z_j)J + (Z_k)K.$$

A transformation matrix with both rotation and translation might look like this in a QIF instance file:

```
<Transform id="2">
  <Rotation>
    <XDirection>0.8660254037844 0 -0.5</XDirection>
    <YDirection>0 1 0</YDirection>
    <ZDirection>0.5 0 0.8660254037844</ZDirection>
  </Rotation>
  <Origin>5.5179491924311 0.5 3</Origin>
</Transform>
```


If the transformation matrix is a rotation only matrix then the optional **Origin** *element* is missing. In this case, the origin offset is $O_x = O_y = O_z = 0.0$ and formula (1) simplifies to:

$$X = (X_i)x + (Y_i)y + (Z_i)z,$$

$$Y = (X_j)x + (Y_j)y + (Z_j)z, \quad (1a)$$

$$Z = (X_k)x + (Y_k)y + (Z_k)z.$$

Formula (2) for the mapping of vectors between common and local spaces is unchanged.

Formula (3) for the mapping points from common space into local space simplifies to:

$$x = (X_i)\ X + (X_j)\ Y + (X_k)\ Z,$$

$$y = (Y_i)\ X + (Y_j)\ Y + (Y_k)\ Z, \quad (3a)$$

$$z = (Z_i)\ X + (Z_j)\ Y + (Z_k)\ Z.$$

And formula (4) for mapping the vector (I, J, K) in common space to the vector (i, j, k) in local space remains unchanged.

A rotation-only transformation matrix might look like this in a QIF instance file:

```
<Transform id="2">
  <Rotation>
    <XDirection>0.8660254037844 0 -0.5</XDirection>
    <YDirection>0 1 0</YDirection>
    <ZDirection>0.5 0 0.8660254037844</ZDirection>
  </Rotation>
</Transform>
```

If the transformation matrix is a translation only matrix then the optional **Rotation** *element* is missing. In this case, the rotation matrix is the 3 by 3 identity matrix and formula (1) simplifies to:

$$X = x + O_x,$$

$$Y = y + O_y, \quad (1b)$$

$$Z = z + O_z.$$

If there is no rotation then vectors remain unchanged and formula (2) simplifies to:

$$I = i,$$

$$J = j, \quad (2b)$$

$$K = k.$$

Formula (3) for mapping points from common space into local space simplifies to:

$$x = X - O_x,$$

$$y = Y - O_y, \quad (3b)$$

$$z = Z - O_z.$$

Because vectors remain unchanged and formula (4) simplifies to:

i = I,

j = J, (4b)

k = K.

A translation-only transformation matrix might look like this in a QIF instance file:

```
<Transform id="2">
  <Origin>5.5179491924311 0.5 3</Origin>
</Transform>
```

If the transformation matrix has neither translation nor rotation then both the optional **Origin** and **Rotation** *elements* are missing and points and vectors remain unchanged in the transformation:

Formula (1) simplifies to:

X = x,

Y = y, (1c)

Z = z.

Formula (3) simplifies to:

x = X,

y = Y, (3c)

z = Z.

And the vector transformations are described by formulas (2b) and (4b).

A transformation matrix with neither rotation or translation might look like this in a QIF instance file:

```
<Transform id="2"/>
```

5.16.2.1 Transformation matrix example

Figure 24 shows a part with two Cartesian axis systems for the common space and a local space. The Cartesian coordinates and axis vector direction of a hole center are shown in both the common and local spaces.

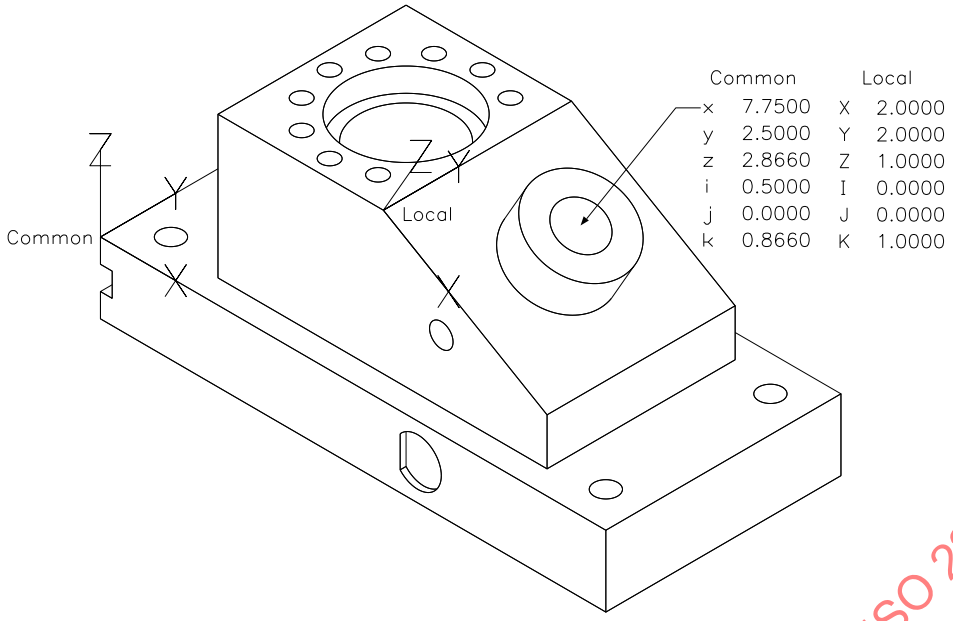


Figure 24 – Transformation matrix example

The origin of the local space in common space coordinates is (O_x = 5.5179, O_y = 0.5000, O_z = 3.0000). The X-axis of the local space in terms of common space is (X_i = 0.8660, X_j = 0.0000, X_k = -0.5000), the Y-axis of the local space in terms of common space is (Y_i = 0.0000, Y_j = 1.0000, Y_k = 0.0000), and the Z-axis of the local space in terms of common space is (Z_i = 0.5000, Z_j = 0.0000, Z_k = 0.8660) giving the 4x3 transformation matrix:

$$\begin{bmatrix} 0.8660 & 0.0000 & -0.5000 \\ 0.0000 & 1.0000 & 0.0000 \\ 0.5000 & 0.0000 & 0.8660 \\ 5.5179 & 0.5000 & 3.0000 \end{bmatrix}$$

To transform the hole center in local coordinates (2.000, 2.000, 1.000) to coordinates in the common space the formula set (1) is used:

X = (0.8660)(2.0000) + (0.0000)(2.0000) + (0.5000)(1.000) + 5.5179 = 7.7500

Y = (0.0000)(2.0000) + (1.0000)(2.0000) + (0.0000)(1.000) + 0.5000 = 2.5000

Z = (-0.5000)(2.0000) + (0.0000)(2.0000) + (0.8660)(1.000) + 3.0000 = 2.8660

To transform the hole axis vector in local coordinates (0.000, 0.000, 1.000) to coordinates in the common space the formula set (2) is used:

I = (0.8660)(0.0000) + (0.0000)(0.0000) + (0.5000)(1.000) = 0.5000

J = (0.0000)(0.0000) + (1.0000)(0.0000) + (0.0000)(1.000) = 0.0000

K = (-0.5000)(0.0000) + (0.0000)(0.0000) + (0.8660)(1.000) = 0.8660

The transform of the hole center from common space coordinates (7.7500, 2.500, 2.8660) to local coordinates uses formula set (3):

x = (0.8660)(7.7500-5.5179) + (0.0000)(2.5000-0.5000) + (-0.5000)(2.8660-3.0000) = 2.0000

y = (0.0000)(7.7500-5.5179) + (1.0000)(2.5000-0.5000) + (0.0000)(2.8660-3.0000) = 2.0000

z = (0.5000)(7.7500-5.5179) + (0.0000)(2.5000-0.5000) + (0.8660)(2.8660-3.0000) = 1.0000

And the transform of the hole axis vector from common space coordinates (0.5000, 0.0000, 0.8660) to local coordinates uses formula set (4):

i = (0.8660)(0.5000) + (0.0000)(0.0000) + (-0.5000)(0.8660) = 0.0000

j = (0.0000)(0.5000) + (1.0000)(0.0000) + (0.0000)(0.8660) = 0.0000

k = (0.5000)(0.5000) + (0.0000)(0.0000) + (0.8660)(0.8660) = 1.0000

5.16.3 Transforms

The ***CoordinateSystemCoreType*** provides the minimum mathematical description for a coordinate transformation. The ***TransformMatrixType*** is derived from the ***CoordinateSystemCoreType*** adding the ability to define the units, accuracy, etc. of the origin point.

The ***TransformInstanceType*** is derived from the ***TransformMatrixType*** and so contains the 4x3 transformation matrix information but in addition has an **id** *attribute*, and **Name** and **Attributes** *elements*.

The **id** *attribute* allows the transform to be referenced and therefore allows a single transformation matrix to be shared by several objects.

The optional **Name** and **Attributes** *elements* allow for the transform to be identified, have arbitrary information and data added.

5.16.4 Coordinate systems

The ***CoordinateSystemType*** defines a coordinate axis system on a measurement device. A coordinate system on a measurement device typically involves aligning measured features on a real part to their nominal counterparts either using a holding fixture that physically interfaces with the measured alignment features, or by measuring the alignment features and performing the alignment mathematics in software.

In addition to aligning measured features to their nominal counterparts a coordinate system on a measurement device establishes a coordinate axis system and coordinate origin associated with the alignment. This results in two transformation matrices: one which defines the axis system on the nominal part with respect to the common space, and one which defines the axis system on the actual part with respect to the common space. The nominal and actual transformation matrices are stored in separate types to allow for multiple actual part measurements to share the same nominal transformation information.

A single coordinate system can coincide with the coordinate space of the root component. This coordinate system can be identified with the optional **CommonCoordinateSystemId** *element* of the ***CoordinateSystemListType***.

5.16.4.1 Nominal and actual transforms

The ***CoordinateSystemType*** has an optional *element*, **NominalTransform** and the ***CoordinateSystemActualTransformType*** has an *element* **ActualTransform** both of ***TransformMatrixType***, which hold the nominal and actual versions of the transformation matrix. The actual transform is stored separately on a per measurement results basis and is associated with the corresponding nominal coordinate system by **id**.

The meaning of and differences between these two transformation matrices is perhaps best illustrated by example.

The example in Figure 24 shows two Cartesian axis systems. In the context of QIF, if these two axis systems are established by aligning to measured features, they can become coordinate systems. The coordinate system labelled “common” is established by a set of actual part features and coincides with the root component space. Both the nominal and actual transformation matrices for this coordinate system will be the identity rotation matrix with a (0, 0, 0) origin offset:

```
  <CoordinateSystem id=”87”>
    <NominalTransform>
      <Rotation>
        <XDirection>1 0 0</XDirection>
        <YDirection>0 1 0</YDirection>
        <ZDirection>0 0 1</ZDirection>
      </Rotation>
      <Origin>0 0 0</Origin>
    </NominalTransform>
  </CoordinateSystem>
…
  <Transform>
    <ActualTransform>
      <Rotation>
        <XDirection>1 0 0</XDirection>
        <YDirection>0 1 0</YDirection>
        <ZDirection>0 0 1</ZDirection>
      </Rotation>
      <Origin>0 0 0</Origin>
    </ActualTransform>
   <CoordinateSystemId>87</CoordinateSystemId>
  </Transform>
```

or equivalently (using the ability to omit the optional **Rotation** *element* if there is no rotation, and omit the optional **Origin** *element* if there is no origin offset):

```
 <CoordinateSystem id=”87”>
   <NominalTransform/>
   …
 </CoordinateSystem>
  …
```

```
 <Transform>
  <ActualTransform/>
  <CoordinateSystemId>87</CoordinateSystemId>
</Transform>
```

If the coordinate system labelled “local” is established by a different set of measured features than those used to establish the coordinate system labelled “common” then (unless the actual part is a perfect representation of the nominal part) the actual transformation will not exactly match the nominal transformation:

```
<CoordinateSystem id=”88”>
  <NominalTransform>
    <Rotation>
      <XDirection>0.866025 0 -0.5</XDirection>
      <YDirection>0 1 0</YDirection>
      <ZDirection>0.5 0 0.866025</ZDirection>
    </Rotation>
    <Origin>5.517949 0.5 3</Origin>
  </NominalTransform>
  …
</CoordinateSystem>
 …
 <Transform>
   <ActualTransform>
    <Rotation>
      <XDirection>0.867865 0.009721 -0.496705</XDirection>
      <YDirection>-0.009058 0.999952 0.003743</YDirection>
      <ZDirection>0.496717 0.001251 0.867912</ZDirection>
    </Rotation>
    <Origin>5.517937 0.499941 2.999991</Origin>
  </ActualTransform>
  <CoordinateSystemId>88</CoordinateSystemId>
</Transform>
```

5.16.4.2 Alignment operations

A coordinate system on a measurement device is established using nominal and measured features by performing a set of alignment operations. This set is represented in the ***CoordinateSystemType*** by the **AlignmentOperations** *element,* which is of ***AlignmentOperationsType***. The ***AlignmentOperationsType*** is a list of **AlignmentOperation** *element*s, each of which is nominally of ***AlignmentOperationBaseType***. However, the **AlignmentOperation** *element* is the head of a substitution group of *element*s (described in the following subclause) that are of more specific types of alignment operation. *Element*s that are members of the substitution group must be used instead of **AlignmentOperation** *element*s. All of the more specific alignment operation *types* derive from the ***AlignmentOperationBaseType*** which has the required *element* **SequenceNumber** used to order alignment operations.

Any number of alignment operations can exist in a coordinate system. Only the transformation matrices (nominal and actual) of the accumulated effect of all alignment operations is stored on the coordinate system. If the transformation matrix information is required for the individual steps in a real alignment process, then a QIF coordinate system instance must be generated for each step.

5.16.4.2.1 *Alignment operation types*

The **PrimaryAlignment** *element* (which is of ***PrimaryAlignmentOperationType)*** is used to describe an alignment operation where a coordinate axis is made to align exactly with an alignment feature normal or axis.

The **SecondaryAlignment** *element* (which is of ***SecondaryAlignmentOperationType*)** is used to describe an alignment operation where a coordinate axis is made to align with an alignment feature normal or axis as exactly as possible with the constraint that a previously defined primary axis remains unchanged. In best practice the feature normal or axis of the secondary alignment feature is often nominally orthogonal to the primary axis but need not be so as long as it is not parallel to the primary axis.

The **ActualOffset** *element* (which is of ***ActualOffsetAlignmentOperationType*)** is used to describe the establishment of coordinate origins based on the location of alignment features.

The **NominalOffset** *element* (which is of ***NominalOffsetAlignmentOperationType*)** is used to describe the offset of a coordinate origin by a nominal numerical value.

The **NominalRotation** *element* (which is of ***NominalRotationAlignmentOperationType*)** is used to describe the rotation about a specified axis by a nominal numerical angle.

The **DatumPrecedence** *element* (which is of ***DatumPrecedenceAlignmentOperationType*)** is used to describe an alignment based on datum precedence and degrees-of-freedom rule.

The **BestFitAlignment** *element* (which is of ***BestFitAlignmentOperationType*)** is used to describe a best-fit alignment from a set of alignment features.

The **Machine** *element* (which is of ***MachineCoordinateSystemOperationType*)** is used to describe a switch to the integral coordinate system of a measurement device.

5.16.5 CAD coordinate systems

The ***CADCoordinateSystemType*** which has a **CoordinateSystemCore** *element* of type ***CoordinateSystemCoreType*** is used to define a coordinate system graphical element in a model based definition. These coordinate systems can have graphical representations and may be used for user interface purposes transforming between local and CAD global coordinates and to define sketch planes.

A CAD coordinate system may be associated with a PMI coordinate system by populating the **InternalCADCoordinateSystemId** *element* of the ***CoordinateSystemType*** with the id of an instance of ***CADCoordinateSystemType***. A similar link to a coordinate system defined in a CAD file using a non-QIF format can be established by using the **ExternalCADCoordinateSystemId** *element*.

5.16.6 Coordinate system lists

Coordinate systems representing alignments for metrology purposes are collected as a list of **CoordinateSystem** *elements* (of type ***CoordinateSystemType***) in the ***CoordinateSystemListType***. It is this type that also contains the **CommonCoordinateSystemId** *element* which identities the single coordinate system which matches the root component coordinate system equivalent to the CAD global coordinate system.

The actual transforms associated with these coordinate systems are collected as a list of **Transform** *elements* (of type ***CoordinateSystemActualTransformType***) in the

CoordinateSystemsActualTransformsType. Each **MeasurementResults** *element* has a sub-*element* **CoordinateSystemsActualTransforms** (of type ***CoordinateSystemsActualTransformsType***) which holds the actual transform list; the results of a single product measurement.

CAD coordinate systems representing CAD user interface coordinate system graphical elements are collected as a list of **CoordinateSystem** *elements* (of type ***CADCoordinateSystemType***) in the ***CoordinateSystemSetType***.

5.17 Feature control frames

A feature control frame like ⌖ | Ø.05Ⓜ | A | BⓂ | CⓂ typically consists of several components: the characteristic type defined by the GD&T symbol ⌖; the tolerance zone size with zone shape, material condition and other modifiers Ø.05Ⓜ; and a datum reference frame consisting of datum labels with material condition or material boundary modifiers A | BⓂ | CⓂ. Feature control frames for form tolerances do not include a datum reference frame. The meaning of the feature control frame can be further refined using other symbols, notes or leader line modifiers like ⌀.

5.17.1 Geometric tolerance characteristic types

The GD&T symbol contained in the leftmost box of the feature control frame of a geometric tolerance defines which QIF types are to be used to describe the characteristic. The defined types follow; ***XXX*** in a type name represents ***Definition***, ***Nominal***, ***Measurement***, or ***Item***,

Form characteristics:

- ○ CircularityCharacteristicXXXType
- ⌭ CylindricityCharacteristicXXXType
- ▱ FlatnessCharacteristicXXXType
- — StraightnessCharacteristicXXXType

Location characteristics:

- ◎ CoaxialityCharacteristicXXXType (ISO specific)
- ◎ ConcentricityCharacteristicXXXType
- ⌖ PositionCharacteristicXXXType
- ⌯ SymmetryCharacteristicXXXType

Orientation characteristics:

- ∠ AngularityCharacteristicXXXType
- // ParallelismCharacteristicXXXType
- ⊥ PerpendicularityCharacteristicXXXType

Profile charactertistics:

- ⌓ LineProfileCharacteristicXXXType

 SurfaceProfileCharacteristicXXXType

Runout charactertistics:

 CircularRunoutCharacteristicXXXType

 TotalRunoutCharacteristicXXXType

In addition there is a ***PointProfileCharacteristicXXXType*** to implement a vector tolerance at a single measured point. Several point profile characteristics can be placed in a characteristic grouping to be evaluated simultaneously.

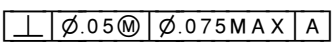
5.17.2 Tolerance zone size

The size of the tolerance zone is defined in the definition base type for all geometric tolerance characteristics: ***GeometricCharacteristicDefinitionBaseType***.

Depending on the characteristic type, the size of the tolerance zone may be subject to a bonus tolerance. This is indicated in the feature control frame by the use of a material condition modifier symbol which indicates the material condition at which the tolerance zone applies. When a particular characteristic type allows for the use of a material condition modifier, the required **MaterialCondition** *element* must be used. The **MaterialCondition** *element* can have one of six values:

REGARDLESS	(Y14,5 1983 specific) when the Ⓢ symbol appears after the tolerance (RFS)
LEAST	when the Ⓛ symbol appears after the tolerance (LMC)
MAXIMUM	when the Ⓜ symbol appears after the tolerance (MMC)
LEAST_RPR	(ISO specific) when the Ⓛ symbol appears after the tolerance followed by the reciprocity requirement symbol Ⓡ
MAXIMUM_RPR	(ISO specific) when the Ⓜ symbol appears after the tolerance followed by the reciprocity requirement symbol Ⓡ
NONE	when no material condition modifier symbol appears after the tolerance

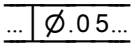
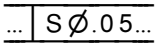
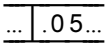
Table 3 – Material Condition Values

In the case of orientation characteristics, the amount of bonus applied to the tolerance zone may be limited by defining the maximum tolerance size in the feature control frame. For example the size of the perpendicularity tolerance zone in  is limited to .075 regardless of the available bonus. In QIF the optional **MaximumToleranceValue** is used to indicate the maximum size of an orientation characteristic tolerance zone.

A tolerance zone which varies in size must use the optional **ZoneLimit** *element* to define the limiting points for the variable tolerance zone. The tolerance zone starts with a value of **ToleranceValue** at the **FromPoint** and changes linearly in size to the value of **ToPointToleranceValue** at the **ToPoint**.

5.17.3 Zone shape

The shape of the tolerance zone is defined by symbols preceding the tolerance zone size in the feature control frame. These fall into three broad categories:

	Diametrical (cylindrical)
	Spherical
	Non-diametrical

The orientation of a tolerance zone implied by the orientation of the feature to which a feature control frame is applied, or by the placement of a feature control frame on a drawing, or by a combination of the two, can be explicitly defined by the optional **ZoneOrientationVector** *element*.

The application of a position characteristic diametrical zone to an elongated feature like the round ends of a slot is indicated with the optional **ElongatedZone** *element*.

The application of a position characteristic non-diametrical zone to the boundary of a feature by use of a BOUNDARY note on the feature control frame is indicated with the optional **BoundaryZone** *element*.

A position characteristic diametrical or non-diametrical zone may apply either to a three-dimensional feature or to a two-dimensional feature (a planar section of a real three-dimensional feature). The dimensionality of the zone is defined by the optional **Dimensionality** *element*. A spherical zone is always three dimensional; so when used on a spherical zone the value of the **Dimensionality** *element* is fixed.

5.17.4 Zone extents

The extents of the tolerance zone in a feature control frame are naturally defined by the extents of the feature to which the feature control frame applies. This default behavior can be modified in the following ways: using a projected tolerance zone, defining the upper disposition of the tolerance zone, using chain lines on a drawing to define the extents of the tolerance zone, using a note like A ↔ B and point identifiers on the drawing, using other standard notes, or using a leader line modifier like  .

The use of a projected tolerance zone indicated by the Ⓟ symbol in a feature control frame, is defined by the optional **ProjectedToleranceZone** *element*. Regardless of whether the length of the projected tolerance zone is defined numerically in the feature control frame or as a dimension on the drawing, the **ProjectedToleranceZone** *element* defines the length of the projected tolerance zone.

A profile characteristic tolerance zone is by default centered on the nominal feature to which it is applied. This behavior may be modified by chain lines on the drawing or by using the upper disposition symbol Ⓤ in the feature control frame. Both these methods are defined by the optional

OuterDisposition *element* which defines the size of the tolerance zone outside the material which can vary from zero to the whole tolerance zone.

Zone limits defined by chain lines or by identified points used in notes like A ⟷ B are both implemented in QIF with the optional **ZoneLimit** *element*. This *element* contains two sub-*elements*: **FromPoint** and **ToPoint** which together define the extents of the tolerance zone. The plane in which the zone limits are defined is given by the **NormalDirection** *element* which is the vector normal to the drawing view. Any ambiguity about the path to follow between the from-point and to-point around the part is removed by the **StartDirection** *element*. When the from-point and to-point are identified by labels on a drawing for use in a A ⟷ B style note the optional **Name** sub-*element* on the **FromPoint** and **ToPoint** *elements* is used.

Standard feature control frame notes or leader line modifiers which control the zone extents are handled by various optional *elements* in QIF:

EACH ELEMENT	**EachElement** *element*
EACH RADIAL ELEMENT	**EachRadialElement** *element*
ALL AROUND or ALLAROUND	**ExtentType** *element* with **ExtentEnum** sub-*element* set to ALLAROUND
ALL OVER ALLOVER	**ExtentType** *element* with **ExtentEnum** sub-*element* set to ALLOVER

5.17.5 Other zone modifiers

Other tolerance zone modifiers defined by GD&T symbols are shown below with their corresponding optional QIF *element*:

⟨ST⟩ statistical tolerance	**StatisticalCharacteristic**
Ⓕ free state	**FreeState**

5.17.6 Datum reference frames

The datum reference frame is defined by the required **DatumReferenceFrameId** *element* on all geometric tolerance characteristic types except for form characteristics. Because the same datum reference frame may be found in several feature control frames, a datum reference frame is instantiated from the ***DatumReferenceFrameType*** with a QIF **id**, placed in the **DatumReferenceFrames** *element*, and referenced by the feature control frame via its QIF **id**.

The datum reference frame which the **DatumReferenceFrameId** *element* references contains a hierarchy of information to define datum labels, material condition or boundary modifiers and datum precedence. In cases where a non-form geometric tolerance characteristic has no datum reference frame, the **DatumReferenceFrameId** *element* must still be present but the datum reference frame it references will be empty.

5.17.6.1 Datum definitions

A datum definition defines a datum label and optionally associates it with datum targets or feature items. Because datum definitions can be shared among several datum reference frames or used in coordinate systems they are instantiated from the ***DatumDefinitionType*** with a QIF **id**, placed in the **DatumDefinitions** *element*, and referenced by their **id**s.

The datum label is defined by the **DatumLabel** *element*. This is typically meant to be a single datum identifier like ▶A used to identify a datum feature on a part. Compound datums like A-B that use two or more datum identifiers are only found in feature control frames and are handled in QIF with the ***CompoundDatumType***. In practice, a compound datum may be handled with the ***GroupFeatureXXXType*** or as a constructed feature, in which case a datum label like A-B may be assigned to the compound feature.

The optional **FeatureNominalIds** *element* is used to reference the feature or features which comprise the datum. The optional **DatumTargetIds** *element* is used to reference the datum target or targets associated with the datum.

5.17.6.2 Datum with precedence

The ***DatumWithPrecedenceType*** is the mechanism by which a datum reference frame is composed of datums. Each framed box in a datum reference frame corresponds to a **Datum** *element* of type ***DatumWithPrecedenceType***.

The ***DatumWithPrecedenceType*** has the required **Precedence** *element* which is used to order datums, simple or compound, or datum features into a datum reference frame. The first datum in a datum reference frame will use the **PrecedenceEnum** *element* with a value set to PRIMARY, the second datum will used SECONDARY, the third TERTIARY, etc.

If a simple datum like A is used, then the **SimpleDatum** *element* of ***DatumType*** is chosen which references the QIF **id** of a datum definition with the **DatumDefinitionId** *element*. The **MaterialModifier** *element* is used to apply a material condition or material boundary modifier to the datum. And the **ReferencedComponent** defines whether it is the measured or nominal component of the datum feature associated with the datum that is used.

For a compound datum like A-B the **CompoundDatum** *element* of ***CompoundDatumType*** is chosen. The ***CompoundDatumType*** has two or more **Datum** *elements* of ***SequencedDatumType*** which are used to order simple datums of type ***DatumType***.

The ***DatumWithPrecedenceType*** also supports a datum feature without a datum definition and therefore no datum label. Such a construct will never be seen in a feature control frame. It is included in QIF to handle DMIS and other languages which allow a reference directly to a datum feature. For example, in DMIS:

```
T(PERP1)=TOL/PERP,0.05,FA(PLANE1)
```

As opposed to:

```
DATDEF/DAT(A),FA(PLANE1)

T(PERP1)=TOL/PERP,0.05,DAT(A)
```


When using a feature measurement as a datum feature, the **MeasuredDatumFeature** *element* is chosen. When using a feature nominal as a datum feature the **NominalDatumFeature** *element* is chosen. Both reference the QIF **id** of a feature item but only the **MeasuredDatumFeature** allows for a material condition modifier.

5.17.6.3 Datum reference frame type

The ***DatumReferenceFrameType*** can have zero to five **Datum** *elements* of type ***DatumWithPrecedenceType***.

The case where zero **Datum** *elements* are present corresponds to a feature control frame with no datum reference frame, like ⌓ | 0.5.

The ***DatumReferenceFrameType*** can optionally reference a coordinate system with the **CoordinateSystemId** element. The coordinate system referenced by its QIF **id** can optionally contain nominal and actual transformations. These are not to be confused with the **DRFTransformActualId** which is a reference to a transform which represents the change in the actual coordinate system associated with a mobile datum reference frame.

5.18 QIF handling of units

5.18.1 Introduction

The QIF design seeks to handle units simply and unambiguously in instance files, especially to allow quantities to appear without explicit units specified for each value. QIF uses a scheme where primary and alternate units are specified once in an instance file. Quantities using the primary units can occur in instance files without an explicit attribute giving the name of the unit. Alternate units can be assigned to individual quantities by including an attribute giving the name of the unit. All unit names must be unique for a given unit type.

Primary and alternate units are specified in a QIF instance file by using the **FileUnits** *element* of the QIFDocumentType. The **FileUnits** *element* is defined in Units.xsd. The **FileUnits** *element* specifies an optional primary unit for each of the unit types used in the instance file, and optional alternate units. If any quantity of a given unit type appears with a unit name in an instance file, the corresponding unit type must appear in the **PrimaryUnits** or the **OtherUnits** of the **FileUnits**. Common XML file checkers will signal an error if this rule is violated.

For example, an instance file might give the diameter of a circle feature as follows:

```
<Diameter>7.5</Diameter>
```

If the **LinearUnit** in the **PrimaryUnits** is millimeter, the line above would mean that the diameter is 7.5 millimeters. This association occurs because the **Diameter** *element* is of ***LinearValueType*** in the schema.

The specific unit types and values defined in QIF are:

- linear (i.e. distances)
- angular
- area
- temperature
- time
- speed
- mass

- force
- pressure

In addition, there is ***UserDefinedUnitsType*** and corresponding ***UserDefinedUnitValueType***.

The default unit for all unit types is the SI unit (meter, radian, kelvin, etc.). If it is desired to have a primary unit type not be a SI unit, a **UnitConversion** *element* should be included in the declaration of the primary unit. The **UnitConversion** *element* gives an **Offset** and a multiplication **Factor** that may be used to convert values of the primary unit type to values in terms of SI units.

For example, the meter is the SI unit for length. If a user wants to use the millimeter as the primary length unit in an instance file, the user puts the following lines into the **FileUnits** portion of the instance file:

```
<PrimaryUnits>
  <LinearUnit>
    <SIUnitName>meter</SIUnitName>
    <UnitName>millimeter</UnitName>
    <UnitConversion>
      <Factor>0.001</Factor>
      <Offset>0</Offset>
    </UnitConversion>
  </LinearUnit>
 …
</PrimaryUnits>
```

In the **FileUnits** portion of the instance file, wherever a unit is declared, the name of the SI unit may be given regardless of whether it is the primary unit or not. If the **UnitConversion** is not included in the instance file, the **UnitName** just serves as an alias for the SI unit. For example if the unit type is **LinearUnit**, the **SIUnitName** must be meter if it is used, but the **UnitName** might be meter or m, or anything else the user likes. If the **UnitConversion** is included in the instance file, naming the SI unit makes it clear what units result from applying the conversion. The conversion is always accomplished using the equation:

SI = ((X plus Offset) times Factor)

where SI is the value in SI units, and X is the value in declared units.

5.18.2 PMI units

The **PrimaryUnits** may also designate a **PMILinearUnit**, **PMIAngularUnit**, and **PMIAreaUnit**. For example:

```
<PrimaryUnits>
  <LinearUnit>
    <SIUnitName>meter</SIUnitName>
    <UnitName>millimeter</UnitName>
    <UnitConversion>
      <Factor>0.001</Factor>
      <Offset>0</Offset>
    </UnitConversion>
  </LinearUnit>
  <PMILinearUnit>
```



```
      <SIUnitName>meter</SIUnitName>
      <UnitName>inch</UnitName>
      <UnitConversion>
        <Factor>0.0254</Factor>
        <Offset>0</Offset>
      </UnitConversion>
    </PMILinearUnit>

    …
  </PrimaryUnits>
```

If a **PMILinearUnit** is included in the **PrimaryUnits**, it applies in (only) the following portions of an instance file, regardless of whether a **LinearUnit** is included in the **PrimaryUnits** (and similarly for angular unit and area unit).

1. all of the **QIFDocument/Characteristics** portion of an instance file.
2. all **CharacteristicMeasurements** *element*s in the **QIFDocument/Results** portion of an instance file.
3. all *element*s of ***AbsoluteLimitsByUnitType*** or ***CriteriaByUnitType*** in the **QIFDocument/Statistics/StatisticalStudiesPlans** portion of an instance file.

5.18.3 Default units

If a **PMILinearUnit** is not included in the **PrimaryUnits**, but a **LinearUnit** is included, the **LinearUnit** applies by default everywhere in an instance file (and similarly for angular unit and area unit).

If a **LinearUnit** is not included in the **PrimaryUnits**, the SI unit (meter in the case of linear unit) applies by default everywhere in an instance file that is not covered by a **PMILinearUnit** that is included in the **PrimaryUnits** (and similarly for all other unit types).

5.18.4 Other units

The **FileUnits** *element* also includes an **OtherUnits** sub-*element* for specifying alternate units. For example, a **LinearUnit** named inch could be defined in the **OtherUnits** *element* as follows:

```
  <OtherUnits>
    <LinearUnit>
      <SIUnitName>meter</SIUnitName>
      <UnitName>inch</UnitName>
      <UnitConversion>
        <Factor>0.0254</Factor>
        <Offset>0</Offset>
      </UnitConversion>
    </LinearUnit>
    …
  </OtherUnits>
```

If a quantity in an instance file is represented using an alternate unit, the name of the unit type must be given. If the definition for inch just given is used in an instance file, a diameter of 5 inches in an instance file would be expressed as follows:

```
<Diameter linearUnit="inch">5</Diameter>
```

5.19 Modeling slots in QIF

5.19.1 Introduction

The QIF library contains four feature types, opposite parallel lines feature defined by the ***OppositeParallelLinesFeatureXXXType***, opposite angled lines feature defined by the ***OppositeAngledLinesFeatureXXXType***, opposite parallel planes feature defined by the ***OppositeParallelPlanesFeatureXXXType***, and angled opposite planes feature defined by the ***OppositeAngledPlanesFeatureXXXType***, which are designed to accommodate a variety of real features commonly referred to as slots, grooves, ribs, webs or blocks. These features have the following characteristics in common: those with parallel sides are features of size, all have a center-line or center-plane about which two straight or flat sides are symmetrically opposed (the use of complex draft on an opposite planes feature may affect strict symmetry). Collectively these four feature types will be referred to as opposite sides features.

If the sides of a real feature are not flat, then the extruded cross section feature defined by the ***ExtrudedCrossSectionFeatureXXXType*** would be a more appropriate QIF feature for representing the real feature. The opposite lines feature might still be a suitable representation of a planar section of such a feature provided the criterion of symmetrically opposed straight sides is met.

The relationship between the opposite lines features and the opposite planes features is much like the relationship between a circle and a cylinder: the opposite lines features are a two-dimensional planar section of a real feature and the opposite planes features are a three-dimensional representation of a real feature.

The relationship between the angled planes feature and parallel planes feature is much like the relationship between a cone and a cylinder. The former is not a feature of size and the latter is not. A cone may be sectioned by a plane producing a circle which is a feature of size. Similarly an angled planes feature, which is not a feature of size, can be sectioned by a suitably oriented plane to produce a parallel lines feature which is a feature of size.

5.19.2 Internal and external

An opposite sides feature can be either internal or external as indicated by the value of the required **InternalExternal** *element*. An internal feature is one defined by the removal of material from the bulk of a part. The center-line/plane of the feature will typically be in open space with the surface normal vectors of the opposite sides pointing generally in the direction of the center-line/plane. These real features are commonly referred to as slots or grooves. Conversely, an external feature is one defined by the bulk of a part. The center-line/plane of the feature will typically be inside the material of the part with the surface normal vectors of the opposite sides pointing generally in the direction away from the center-line/plane. These real features are commonly referred to as ribs, webs or blocks.

5.19.3 Location and size

The location and orientation of an opposite lines feature is defined by the center-line represented by a center-point and the axis unit vector defined by the **StartPoint** and **Vector** *elements* respectively of the required **CenterLine** *element*, and a unit vector representing the normal of the plane in which the feature lies defined by the required **Normal** *element*. The size of the opposite lines feature is given by the required **Width** *element* which applies in a direction perpendicular to the center-line axis in the plane of the feature. In the case of an angled lines feature the width is specified at the

location of the start point. The feature may also have a length defined by the optional **Length** *element* which applies in the direction of the center-line axis.

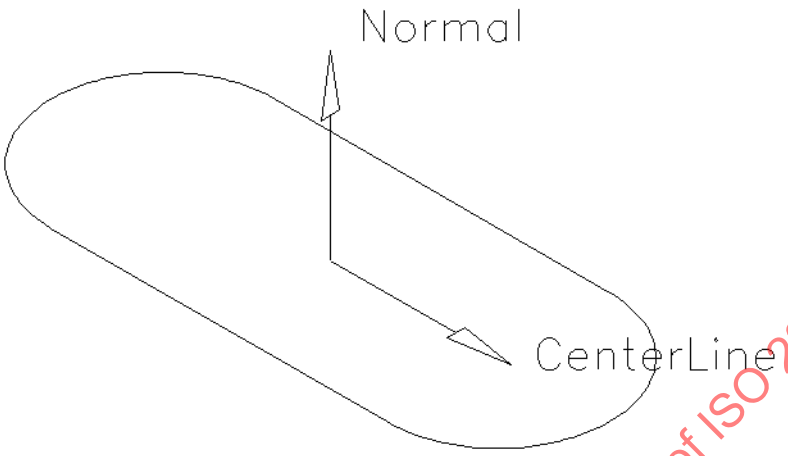


Figure 25 – An opposite parallel lines feature with round closed ends

Figure 25 shows the relationship between the **CenterLine** *element* which defines both the location and the orientation of the axis of the round-ended slot, and the **Normal** *element* which defines the plane in which the feature lies.

The location and orientation of an opposite planes feature is defined by the center-plane defined by a center-point and the center-plane's normal unit vector given by the **Point** and **Normal** *elements* of the required **CenterPlane** *element*. The size of the opposite planes feature is defined by the required **Width** *element* which applies in the direction along the center-plane normal vector. In the case of an angled planes feature the width is specified at the location of the point defined in the center plane. The feature may also have a length defined by the optional **Length** *element* which applies in the direction of the co-requisite **LengthVector** *element*. Furthermore, the feature may have a depth defined by the optional **Depth** *element* which applies in the direction of the co-requisite **DepthVector** *element*.

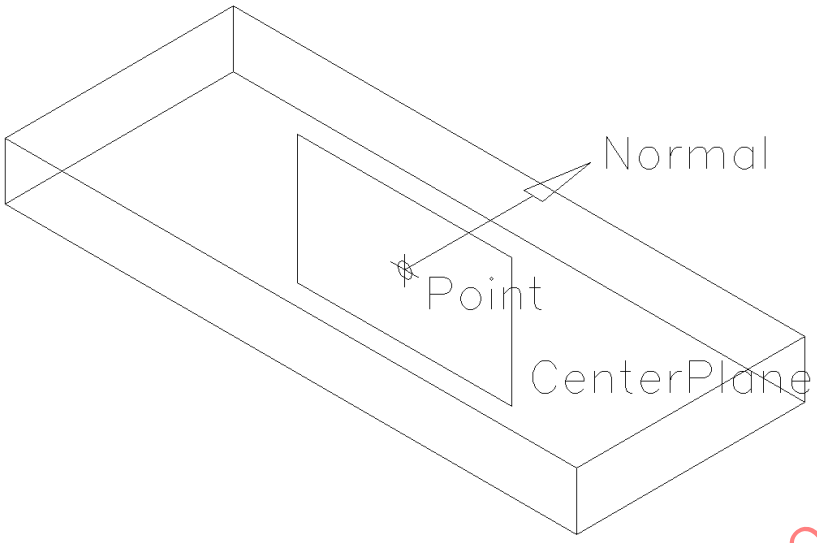


Figure 26 – An opposite parallel planes feature with flat closed ends

Figure 26 shows the relationship of the CenterPlane element with sub-*element*s **Point** and **Normal** which define the orientation of the main sides of a flat-ended slot. In order to define the orientation of the ends of the slot, the optional **LengthVector** *element* must be specified (not shown in Figure 26). The **LengthVector** *element* would have the same orientation as the **CenterLine** in Figure 25.

For both opposite lines and opposite planes features with closed ends the length is from the material boundary at one end to the material boundary at the other end measured along the center-line vector or along the length vector in the center-plane. The length is not between radius centers when a feature has rounded ends. When a feature has one or two open ends then the length can be the distance to a virtual material boundary equivalent to the real material boundary created by placing a flat block over the open end, or it can be an indication of the measureable region of the feature. In all cases the length is symmetrically disposed about the center-point of the feature.

5.19.4 End types

The opposite sides feature types also have an end-type as defined by the required **EndType** *element*. **EndType** provides a choice between (1) one of the enumerated values of the **SlotEndEnum** *element* (ROUND, FLAT, OPEN or UNDEFINED), and (2) a user-defined string in the **OtherSlotEnd** *element*. If the end-type of the opposite sides feature is unknown then the UNDEFINED enumerated end-type is used. If the end-type is known but not covered by any of the enumerated end-types, then a string is used to describe the end-type.

Figure 25 and Figure 26 show examples of the ROUND and FLAT end types respectively. Figure 29 below shows an example of the OPEN end type (upper right).

The shape of the ends of a ROUND and FLAT opposite sides feature can be further modified by using the optional **EndRadius1** and **EndRadius2** *elements* which apply in the directions against and along the center-line axis/length vector respectively. The default condition for round-ended features is to have a circular end tangent to both sides (the actual ends may be described by circular arcs for opposite lines or by cylindrical or conical segments for opposite planes but in a cross-section at any depth the circular ends will be tangent to both sides). By using the end radii

elements, circular cross-section ends that are not tangent to the sides can be specified. The size of the end is given by the **EndRadius** *element* which must be larger than the radius of a tangent end. Whether the end expands beyond the width like a dumbbell shape or not is given by the value of the optional **Expanded** *element*.

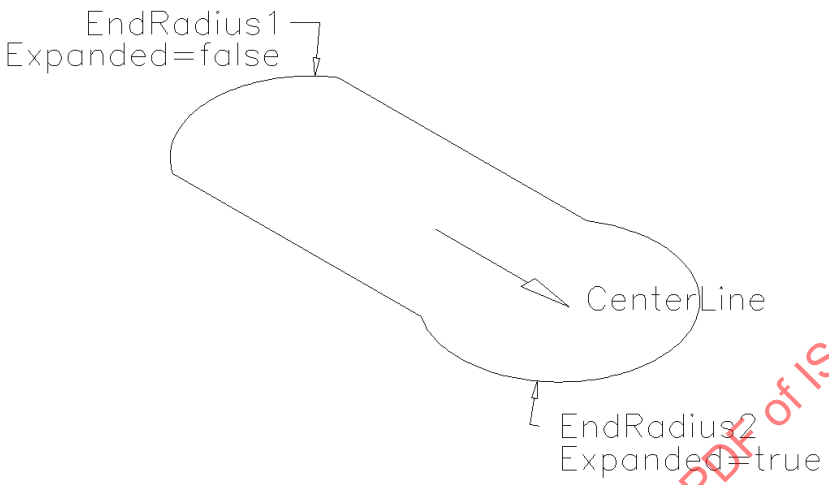


Figure 27 – A slot with non-tangent round ends

Figure 27 shows a round-ended slot with non-tangent ends. The relationship between the ends modified by the **EndRadius1** and **EndRadius2** *elements* and the **LengthVector** or **CenterLine** *elements* is shown. In the example the values of the two end radii are equal, but one is expanded and the other is not. The center of the slot is midway between the extremes of the slots in the axis direction, and not at the midpoint of the centers of the circular ends.

For flat ended opposite sides features the end radii can be used to apply a fillet radius to an otherwise flat end. The end radius must be small enough to leave a portion of the flat end.

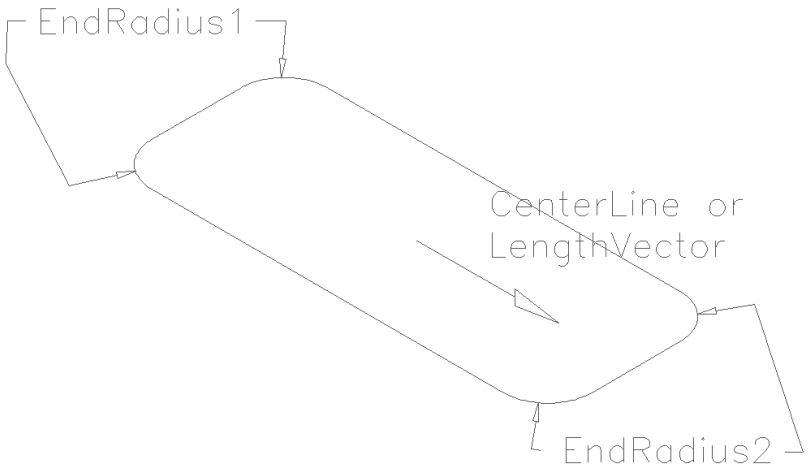


Figure 28 – A flat-ended slot with rounded corners

Figure 28 shows a flat-ended slot with rounded corners. In the example the two end radii values are equal (and both must be smaller than half the width of the slot).

An opposite sides feature may have one closed end and one open end. The presence of a single open end is indicated by the optional **SingleOpenEnd** *element*. In the case of an opposite lines feature the center-line axis must point towards the open end, and for the opposite planes feature the co-requisite **LengthVector** *element* points toward the open end.

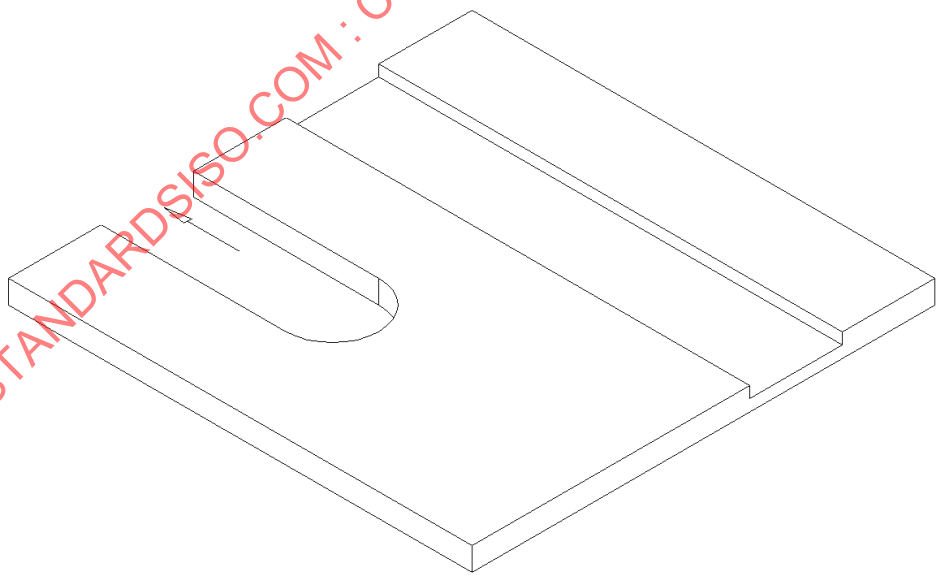


Figure 29 – Opposite planes features with open ends

Figure 29 shows two slots with open ends. The lower left slot would have end type ROUND and use the optional **SingleOpenEnd** *element*. The arrow shows the direction of the **LengthVector** *element*

towards the open end. The upper right slot would have end type OPEN because both ends of the slot are open.

5.19.5 **Bottom types**

The opposite planes features can also define a bottom type with the optional **Bottom** *element*. That element provides a choice between (1) one of the enumerated values of the **BottomEnum** *element* and (2) a user-defined string in the **OtherBottom** *element*. The bottom enum values are BLIND, THROUGH, and UNDEFINED. If the bottom-type of the opposite planes feature is unknown then the **BottomEnum** *element* is used with the value UNDEFINED. If the bottom-type is known but not covered by any of the enumerated bottom-types, then the **OtherBottom** *element* is used with a string value that describes the bottom-type.

The slot in the lower left of Figure 29 is an example of the THROUGH bottom type; the slot in the upper right is an example of the BLIND bottom type.

5.19.6 **Taper**

If an opposite sides feature is tapered then it is represented by one of the angled sides features. The **TaperAngle** *element* is populated with a non-zero value and the width of the opposite sides feature changes along the center-line vector/length vector. The sign of the taper angle defines whether the feature gets larger in the direction of the vector (positive) or gets smaller in the direction of the vector (negative).

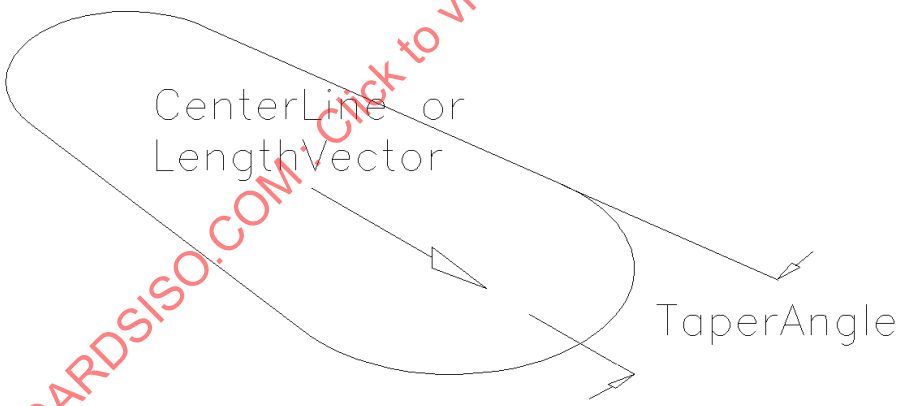


Figure 30 – A tapered slot (opposite angled lines)

Figure 30 shows the relationship between the taper angle and the center line or length vector of a tapered slot. The example shows a slot with a ROUND end type. The center of the slot is midway between the extremes of the slots in the axis direction, and not at the midpoint of the centers of the circular ends.

5.19.7 **Draft**

The opposite angled planes feature instead of a taper angle may have a draft angle defined by the optional **DraftAngle** *element*. A positive draft angle means the feature will open up (get larger) in

the direction defined by the co-requisite **DepthVector** *element* or the co-requisite **DraftVector** *element*, a negative draft angle means the feature will close up (get smaller). When an opposite planes feature is drafted the width and length apply at the center-point. The **DraftVector** *element* overrides the **DepthVector** *element* when the draft vector is not perpendicular to the length vector, the axis vector, or both.

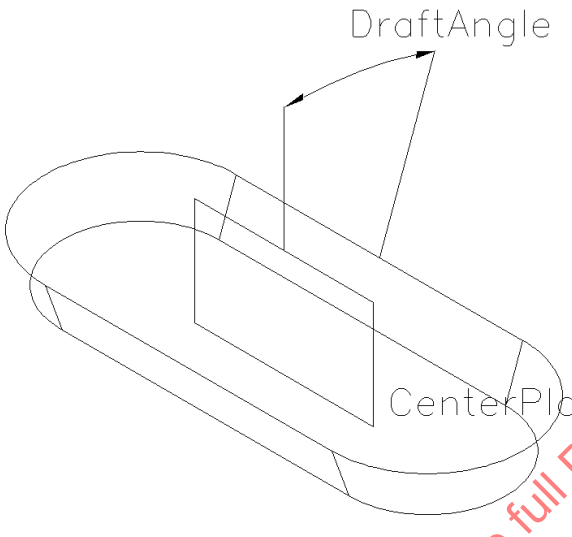


Figure 31 – A slot with draft (opposite angled plane feature)

Figure 31 shows the relationship between the draft angle and the center plane of a slot with draft.

5.19.8 Feature measurement

The above descriptions of required, optional and co-requisite data were particular to nominal opposite sides features. All data elements for measured features are optional to handle different use cases.

In addition to measured elements which correspond directly with nominal elements like measured width and length corresponding with nominal width and length respectively, there are measured data elements for minimum and maximum of size values.

5.20 Modeling cones and conical segments in QIF

5.20.1 Introduction

The QIF library contains two feature types, cone feature defined by the ***ConeFeatureXXXType*** and conical segment feature defined by the ***ConicalSegmentFeatureXXXType***, for defining features with a single, conical surface. The cone feature is meant to describe a feature of size; a feature to which a location characteristic may apply which in turn may acquire bonus from a material condition modifier. The conical segment feature is meant to describe a feature which is not a feature of size; a feature to which a profile characteristic or a radius characteristic at a given cross section may be applied.

The *elements* necessary to describe these two feature types are the same with the exception that the sweep extents for a cone feature are optional while for a conical segment they are required. In

the case of a cone feature the optional **Sweep** *element* might be used to exclude a portion of the conical surface to avoid a keyway. For a conical segment the required **Sweep** *element* defines the extent of the feature which may be the corner of a rounded pocket with draft.

5.20.2 Location, orientation and angle

The minimum information necessary to define the conical surface is its location, orientation and angle. Rather than forcing the location for the cone to be its vertex, the location of cone can be defined anywhere along its axis at a specified diameter. The reason for this is twofold. If the angle of a cone is very small so that the cone is almost cylindrical, the vertex is unstable with respect to small changes in angle. This instability can be removed by defining the location at a diameter rather than at the vertex. The feature to which a cone substitute feature fitting algorithm is to be applied may in fact nominally be a cylinder in which case the location of the vertex is indeterminate.

The locating point for a cone or conical segment is defined by the **AxisPoint** sub-*element* of the **Axis** *element*. The orientation of the cone or conical segment is defined by the **Direction** sub-*element* of the **Axis** *element*. The cone's axis direction defined by the **Direction** sub-*element* always points towards the widening end of the cone The **Diameter** *element* is the diameter of the cone in a plane perpendicular to the direction vector at the axis point. The cone or conical segment can be defined by either its full included angle (the angle between opposite sides) or its half angle (the angle between a side and its central axis) by choosing between the **FullAngle** or **HalfAngle** *element* respectively.

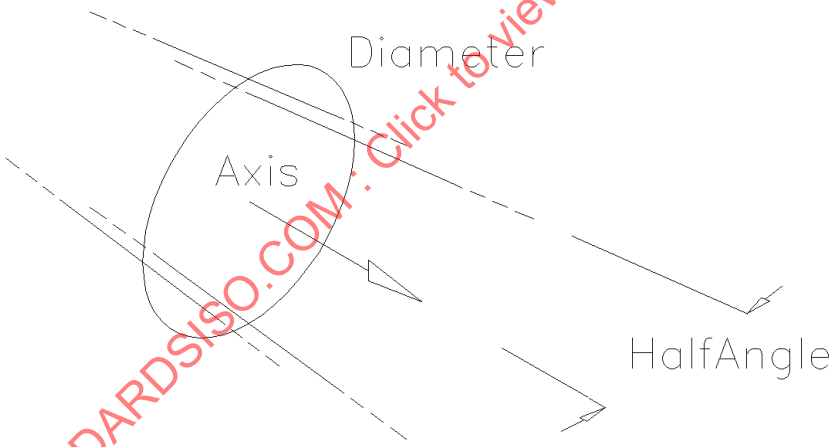


Figure 32 – An unbounded cone located at a reference diameter and defined by its half angle

Figure 32 shows a cone with its location defined at a diameter and with its half angle. This cone definition is stable at small angles and handles the degenerate case of a cylinder.

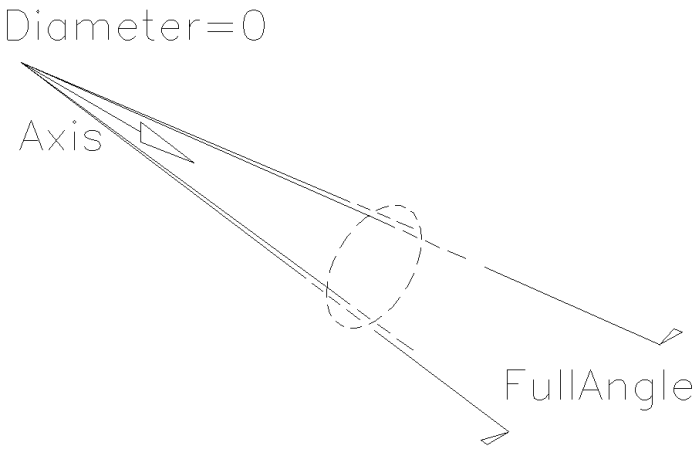


Figure 33 – An unbounded cone located at its vertex and defined by its full angle

Figure 33 shows a cone with its location defined at its vertex and with its half angle. This cone definition is equivalent to that of many CMM systems including DMIS.

5.20.3 **Linear extents**

The simple definitions illustrated in Figure 32 and Figure 33 show unbounded cones with undefined extents in the direction away from the vertex. The linear extents of a cone or conical segment along its axis can be defined with the optional **LargeEndDistance** and **SmallEndDistance** *elements*. These distances can be positive or negative depending on their relationship with the locating point. If the end is along the axis vector from the locating point then the distance will be positive. If the end is in a direction against the axis vector from the locating point then the distance will be negative.

If both the optional **LargeEndDistance** and **SmallEndDistance** *elements* are given then the cone is a conical frustum (or a segment of a frustum): a truncated cone without a pointed end. If the optional **LargeEndDistance** *element* is present but the **SmallEndDistance** *element* is missing then the cone or conical segment has a pointed end.

The effect of the locating point on the sign and value of the **LargeEndDistance** and **SmallEndDistance** *elements* is shown in

Figure 34, Figure 35, Figure 36, and Figure 37. Not shown is the case where the diameter at the locating point is larger than either the small or large ends in which case both distances would be negative. (In all diagrams the required full or half angle *element* is not shown.)

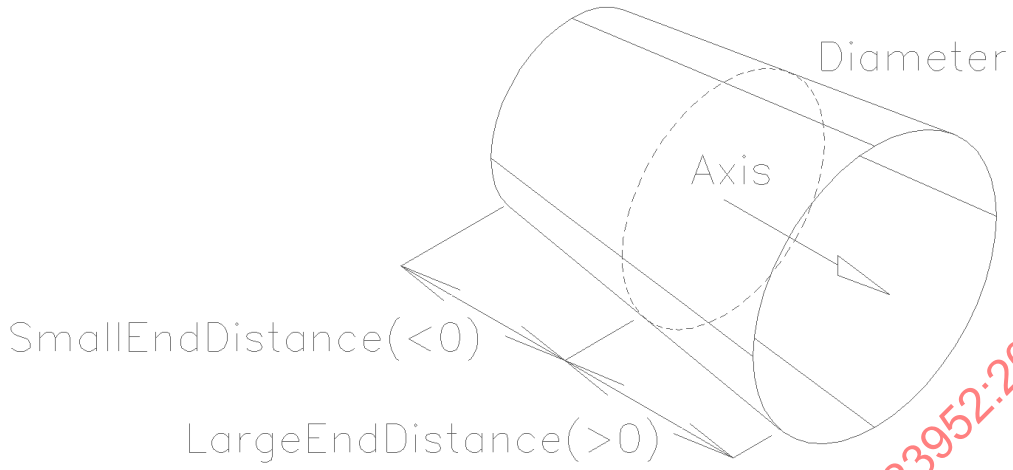


Figure 34 – A bounded, truncated cone located at a reference diameter

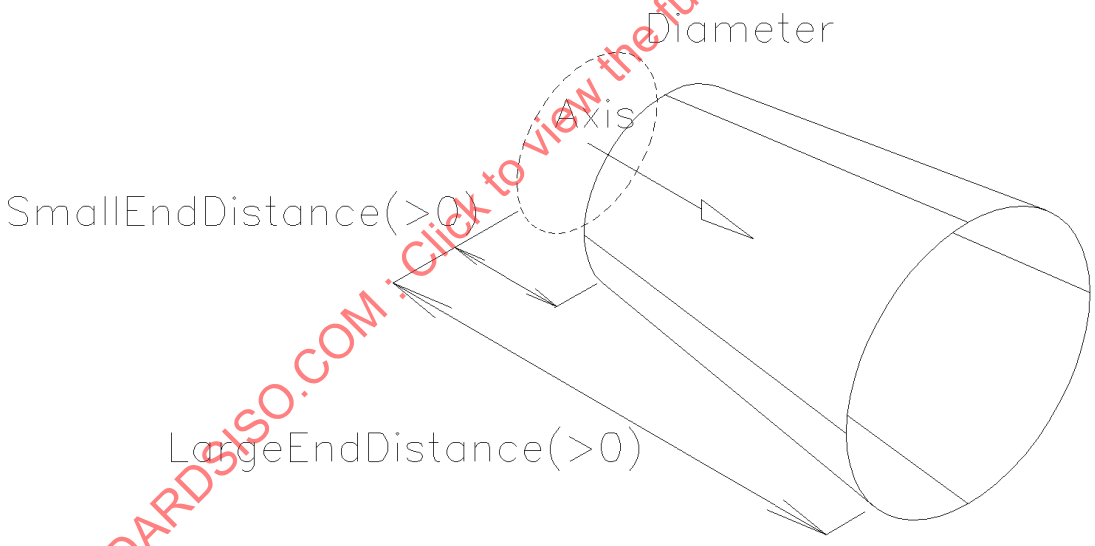


Figure 35 – A bounded, truncated cone located at a virtual reference diameter

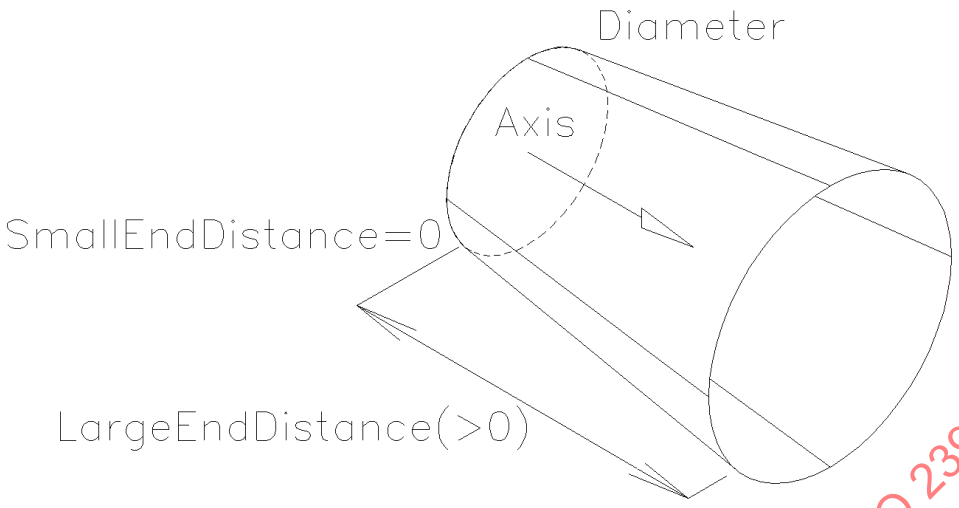


Figure 36 – A bounded, truncated cone located at its small end

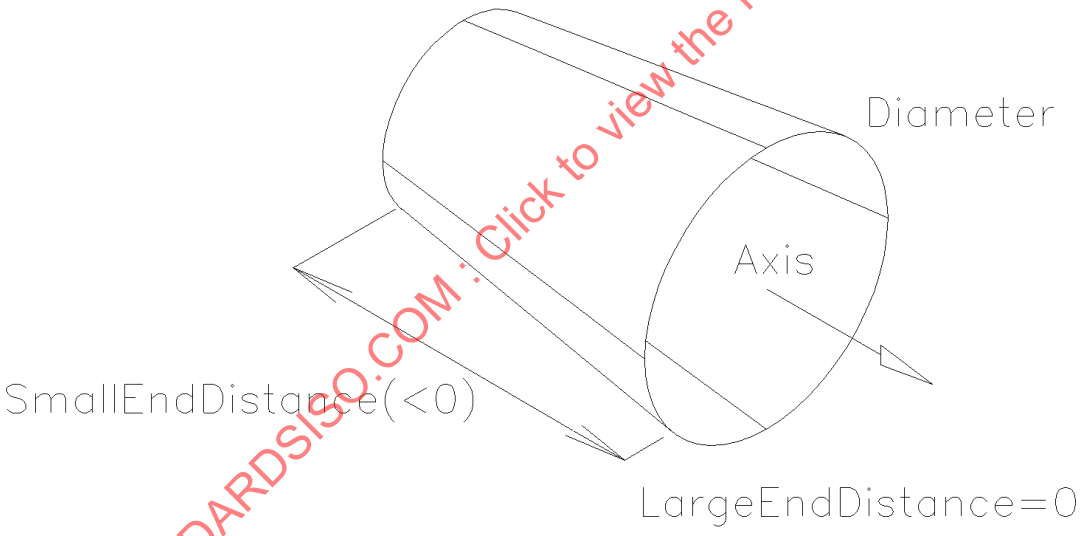


Figure 37 – A bounded, truncated cone located at its large end

Figure 38 and Figure 39 show the bounding of a cone defined with its locating point at its vertex.

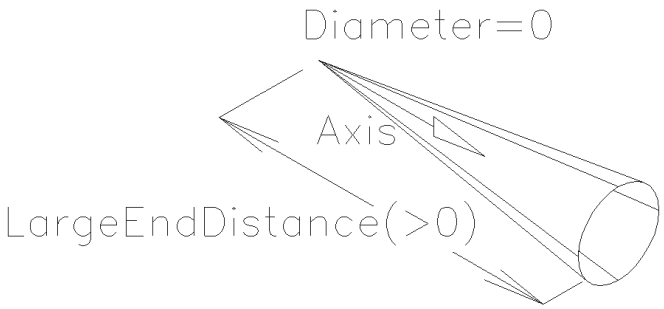


Figure 38 – A bounded pointed cone located at its vertex

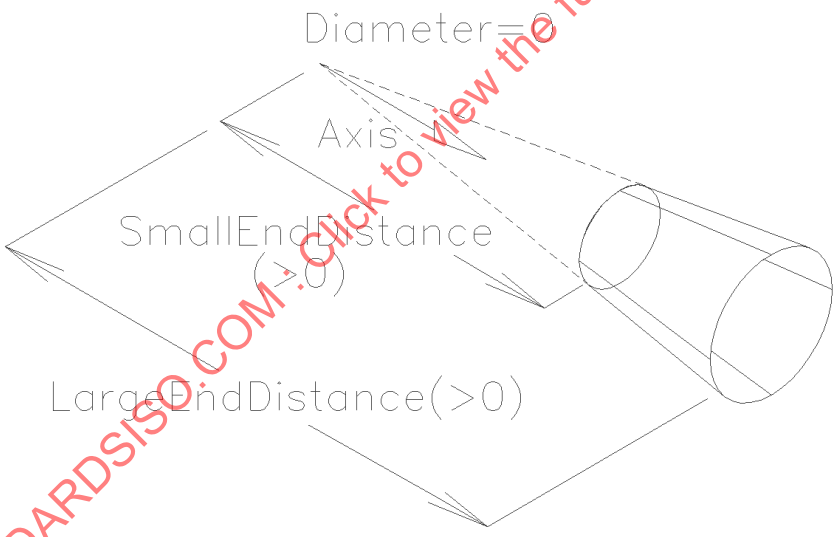


Figure 39 – A bounded truncated cone located at its vertex

5.21 Modeling pattern features in QIF

Pattern features are used for applying position controls to a set of existing features having the same shape. Pattern features are not like other QIF features. Pattern features have definition, nominal, and item aspects, but there is no measurement aspect for pattern features. The ***PatternFeatureItemBaseType*** is derived from the ***GroupFeatureItemBaseType***. The derivation is analogous for the definition and nominal aspects. The features in the pattern are identified by the **FeatureNominalIds** *element* inherited from the ***GroupFeatureNominalType***. The locations of these features must be very close to the locations specified by the pattern. All four pattern types

have a **FirstFeatureLocation** *element*; which must be very close to the location of one of the nominal features in the pattern – not necessarily the first one listed in the **FeatureNominalIds.**

There are four specific derived types of pattern feature: linear, circle, circular arc, and parallelogram (which subsumes rectangular patterns). These are described in the following four subclauses.

5.21.1 Circular pattern feature

Circular pattern features are shown in Figure 40 and Figure 41. The features are arranged with their location points equally spaced in a circle. The ***PatternFeatureCircleDefinitionType*** includes **Diameter**, **FeatureDirection**, and **NumberOfFeatures** *element*s. The **FeatureDirection** is a unit vector giving the orientation of a vector characterizing the features in the pattern, for example, the axis of a cylinder or the direction of an extrusion. The **FeatureDirection** is relative to a coordinate system whose Z axis is the normal to the plane of the circle and whose X axis is the line from the center of the circle to the feature being located. If this *element* is omitted, it means that all features in the pattern have the same orientation relative to the current coordinate system or are not orientable. The ***PatternFeatureCircleNominalType*** provides the **Center**, **Normal**, and **FirstFeatureLocation** *element*s.

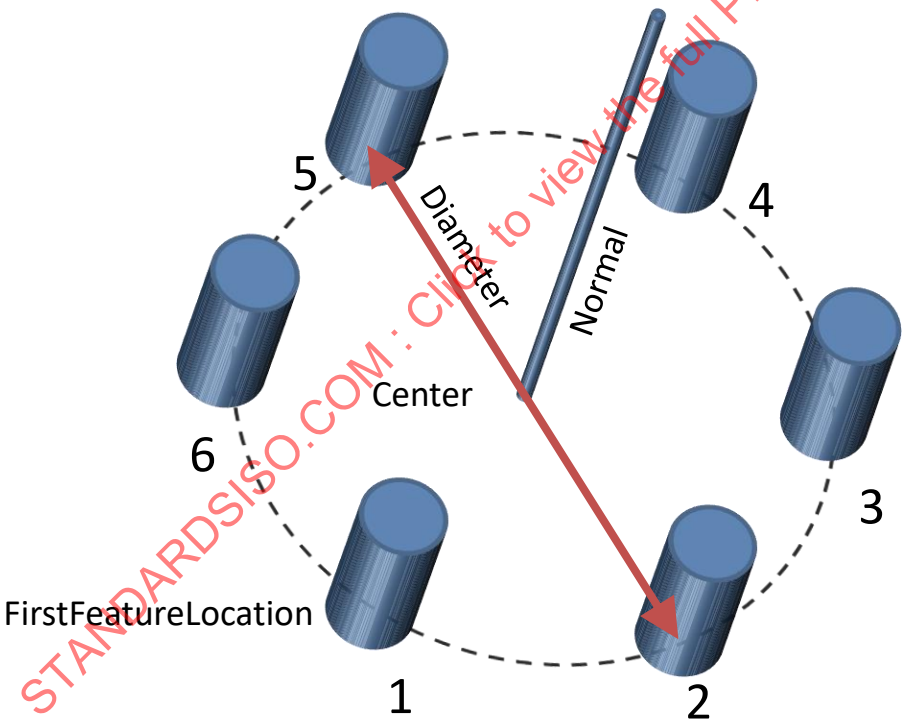


Figure 40 – **PatternFeatureCircle** with **FeatureDirection** omitted

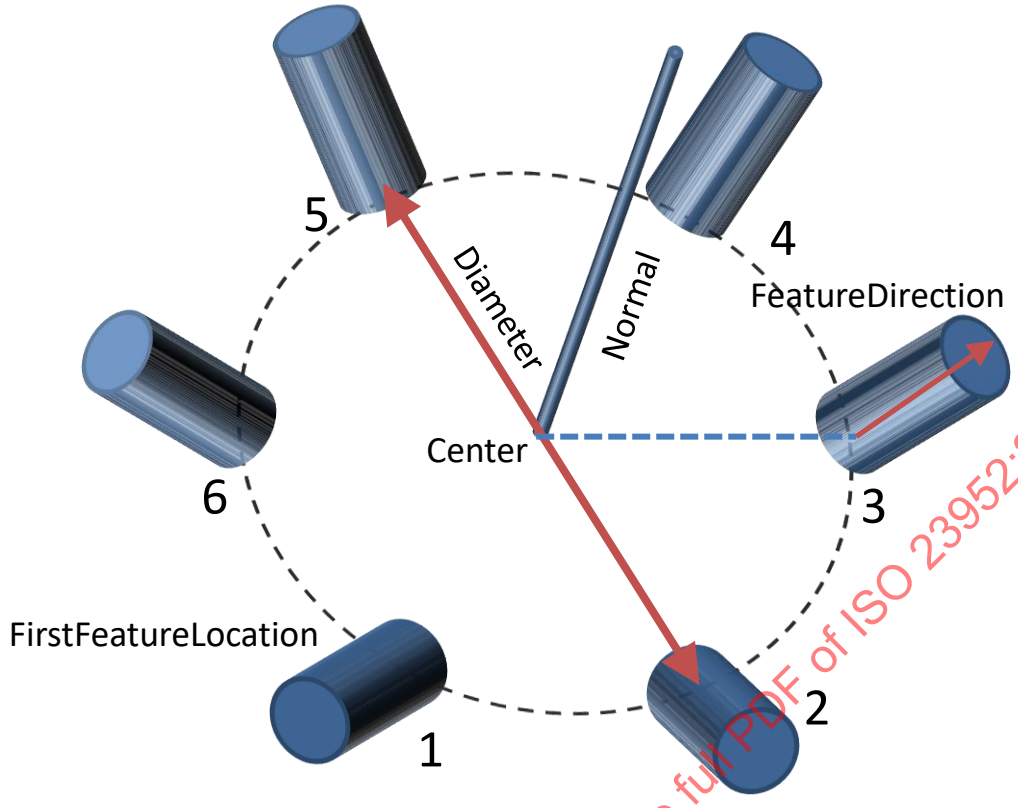


Figure 41 – **PatternFeatureCircle** with **FeatureDirection**

5.21.2 **Circular arc pattern feature**

A circular arc pattern feature is shown in Figure 42. The **Normal**, **Center**, **NumberOfFeatures**, **FirstFeatureLocation**, and **FeatureDirection** *element*s of circular arc pattern features have the same meaning as for circular pattern features. However, a circular arc pattern has an **ArcRadius** rather than a **Diameter**, and also has an **IncrementalArc** giving the angle subtended by any two adjacent features in the pattern. Looking in the direction opposite the **Normal**, the features are arranged counterclockwise from the first feature if the **IncrementalArc** is positive (and clockwise if it is negative).

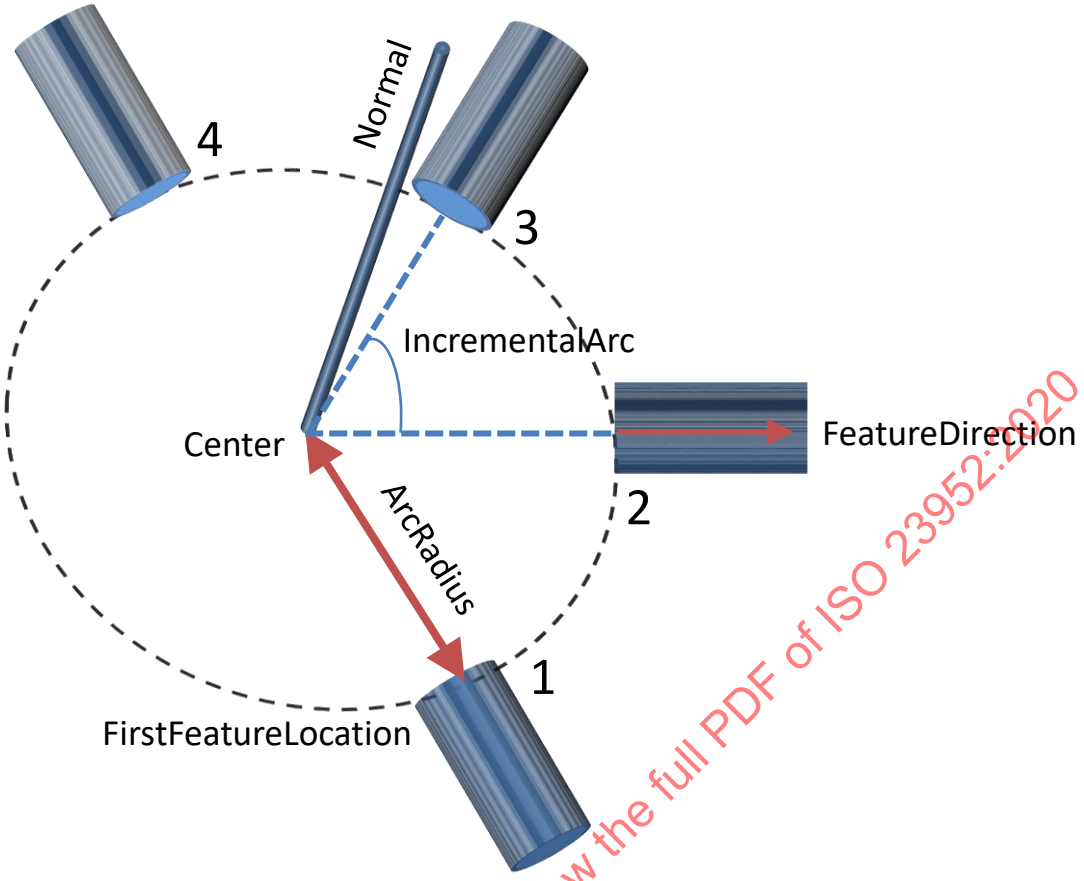


Figure 42 – **PatternFeatureCircularArc**

5.21.3 Linear pattern feature

A linear pattern feature is shown in Figure 43. The ***PatternFeatureLinearDefinitionType*** gives the **LineDirection**, **FeatureDirection**, **IncrementalDistance**, and **NumberOfFeatures** *element*s. The **FirstFeatureLocation** *element* is in the ***PatternFeatureLinearNominalType***. Unlike the circular and circular arc patterns, the **FeatureDirection** is in the same coordinate system as the feature. The **IncrementalDistance** is the distance between adjacent features.

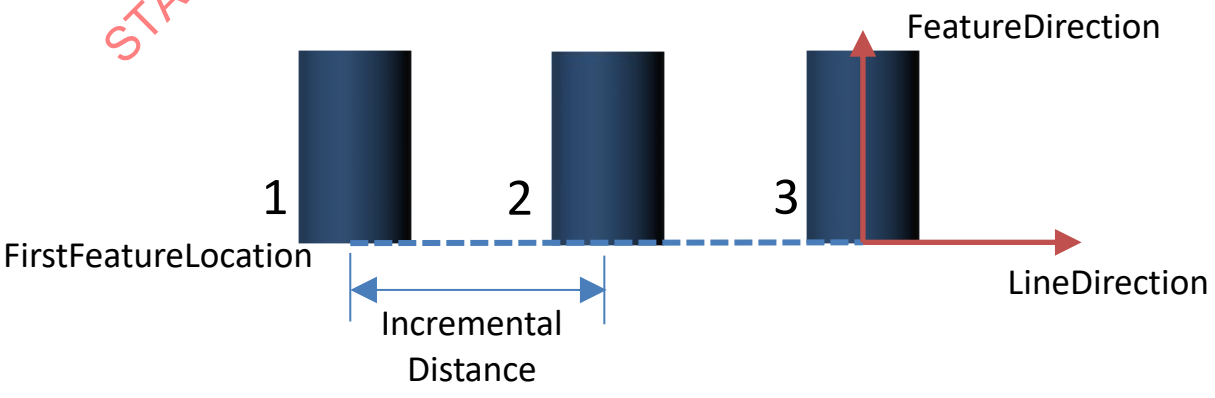


Figure 43 – **PatternFeatureLinear**

5.21.4 **Parallelogram pattern feature**

A parallelogram pattern feature is shown in Figure 44 with sample instance data. The **AlongRowDirection**, **IncrementalRowDistance**, **BetweenRowDirection**, **RowSeparationDistance**, **FeatureDirection**, **NumberOfFeaturesPerRow**, and **NumberOfRows** *element*s are given in the ***PatternFeatureParallelogramDefinitionType***. The **FirstFeatureLocation** *element* is in the ***PatternFeatureParallelogramNominalType***. If the **BetweenRowDirection** is orthogonal to the **AlongRowDirection**, the pattern is rectangular (or square).

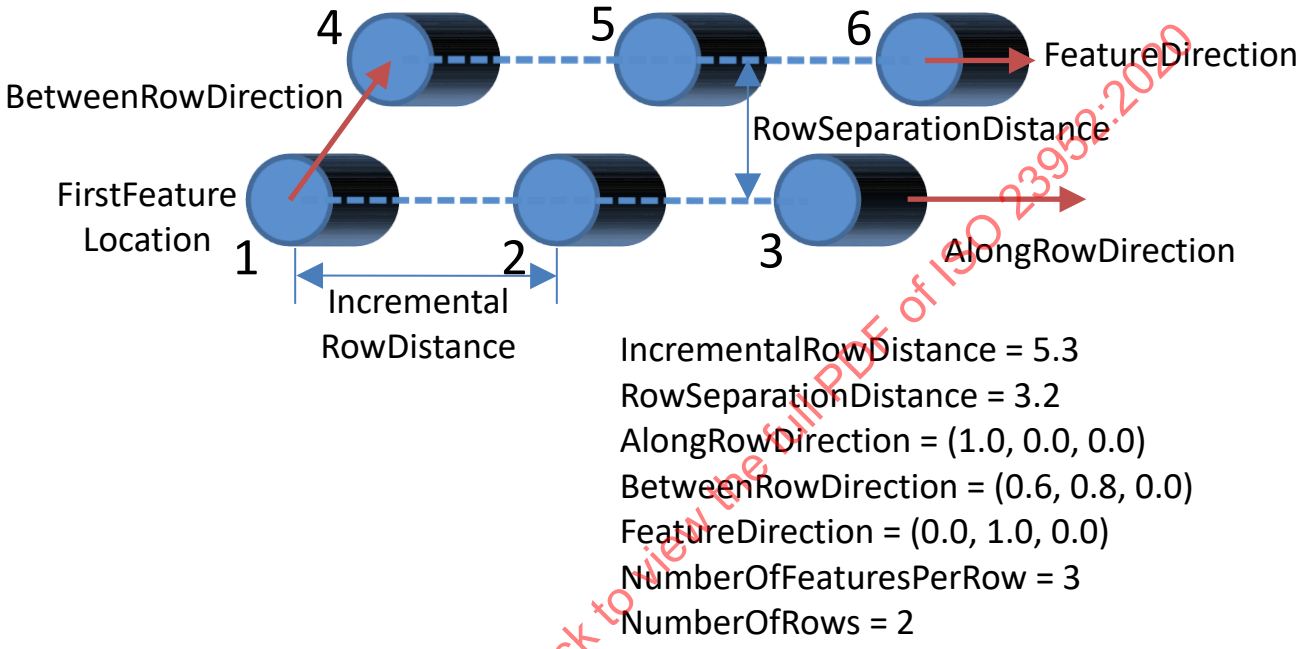


Figure 44 – **PatternFeatureParallelogram**

5.22 Modeling threads in QIF

Features and characteristics often have *element*s that essentially define the same information. For example, a cylinder has a nominal **Diameter** *element* on the ***CylinderFeatureDefinitionType*** and a diameter characteristic has a nominal **TargetValue** via the ***LinearCharacteristicNominalBaseType*** from which ***DiameterNominalCharacteristicType*** derives. Both the diameter and the target value are nominal representations of the size of the feature. Most often these two values will be identical but use cases exist where the manufacturing size and the tolerance evaluation size might be different. For example if a diameter has a specified size of 10.0 +0.5/-0.0 and the manufacturing process targeted the nominal value of 10.0 any undersize variation would result in an out of tolerance condition.

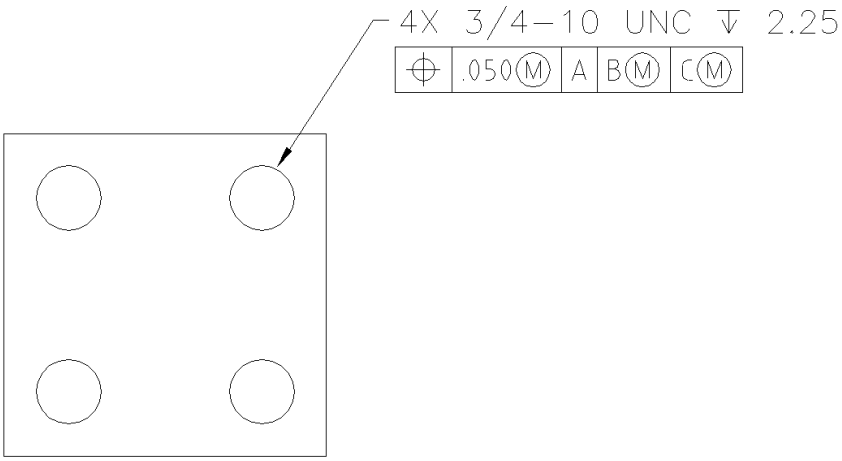


Figure 45 – Threaded features

This is not the case with threaded features. A hole is not threaded to one specification and inspected to another. Also, unlike a simple diameter, the data associated with a thread specification is quite complex as shown in Figure 45. To allow the sharing of thread specification data the threaded feature defined by the ***ThreadedFeatureDefinitionType*** and the thread characteristic defined by the ***ThreadCharacteristicDefinitionType*** both have an *element* **ThreadSpecificationId** which references an instance of ***ThreadSpecificationType*** contained in a list of ***ThreadSpecificationsType***.

5.22.1 Thread specification types

The ***ThreadSpecificationType*** provides a choice between three *elements*: **SingleLeadSpecification** of ***SingleLeadSpecificationType***, **MultiLeadSpecification** of ***MultiLeadSpecificationType***, and **TextThreadSpecification** of ***TextThreadSpecificationType***. The single lead and multi-lead types allow for the unambiguous capturing of detailed information in their *element*s, some of which resolve to enumerations based on industry standards. If these pre-defined data containers are not sufficient to capture a thread specification then that specification may be entered as a simple text string via the **TextSpecification** *element* of ***TextThreadSpecificationType***.

5.23 Feature measurement determination

The feature nominal is a representation of the design intent. The feature measurement is determined from an actual physical part. The QIF information model allows for different levels of detail for the information related to how a feature measurement is determined.

5.23.1 Checked and set features

At the most elementary level, the QIF information model distinguishes between a feature measurement which is checked and one which is set. A feature called out on a print may not be measurable. For example, a circle representing the region on the surface of a part which is inaccessible because of interference with a fixture net pad is not a measureable circle. But because the feature appears on a printed part drawing there may be the requirement for that feature to be reported. To accomplish this, the feature measurement is set to its nominal value. More commonly,

a feature will be checked by a measurement device or otherwise be evaluated using measurement data.

For a set feature, the **Set** *element* of the **DeterminationMode** *element* on the feature item is chosen. If the feature is checked then the **Checked** *element* of the **DeterminationMode** *element* on the feature item is chosen. The differentiation between set and checked features is the only required information from the **DeterminationMode** *element*.

5.23.2 Measurement and construction

More information about how a feature is checked can be added optionally with the **CheckDetails** *element*. A checked feature can be either measured directly or constructed. A directly measured feature is evaluated using data collected directly for that feature by a measurement device. A constructed feature is evaluated using data previously collected for other features referred to as base features.

For a directly measured feature, the **Measured** *element* of the **CheckDetails** *element* is chosen. For a constructed feature, the **Constructed** *element* of the **CheckDetails** *element* is chosen.

5.23.3 Measurement points

Details on the points used for measuring a feature can be added optionally with the **PointList** *element*. This *element* contains a list of references to nominal target points defined in **NominalPointSets** global *element*. Each point set can contain optional information about the measurement device and sensor used to collect each point.

5.23.4 Construction methods

Further detail about a feature construction can be optionally added by choosing between the construction methods for a given feature type.

Common to all construction methods is the optional **NominalsCalculated** *element*. This Boolean *element* is used and set to true when feature nominals are calculated from the nominals of the set of base features rather than being specified directly. This may come about if a generic macro or subroutine is used to perform a feature construction where both the feature nominal and feature measurement are determined from the input base features.

A base feature used in a construction is a reference to another feature item by its **id** in the **FeatureItemId** *element*. In addition the referenced component of the base feature is identified by the **ReferencedComponent** *element*. Usually the measured component, but sometimes the nominal component, of a base feature is used in the construction of the feature measurement. This is not to be confused with the concept of calculated nominals, where the nominal components of all base features are used to calculate the feature nominal.

The recompensated construction method references the measurement points of a base feature rather than the base feature itself. The **BaseFeaturePointList** *element* identifies an ordered list of measurement points on referenced base features. These points are identified by a single point index, a range of point indexes, or all the points on the specified base feature. Point indexes are integers that begin at 1 for the first point in a measurement point list.

The individual construction methods are described in Features.xsd. The description includes the number and types of base feature references required for the particular construction types.

The descriptions of the various construction methods can be found in *Clause 6: QIF Library - Information Model*. Some construction method types are similar; these methods are clarified and differentiated in the following subclauses.

5.23.4.1 Best-fit and recompensated construction methods

Common to many feature types are the best-fit construction method indicated by choosing the **BestFit** *element* in the **Constructed** *element*, and the recompensated construction method indicated by choosing the **Recompensated** *element* in the **Constructed** *element*. Both methods use a substitute feature fitting algorithm to determine the feature measurement from point data. In the case of best-fit construction the point data is that from point reducible base features. The point-reduced data is compensated for probe size and a substitute feature algorithm may have been applied, such as determining the center of a circle. For recompensated construction the point data is uncompensated raw measurement points from base features. The result is that a recompensated constructed feature will be the same as if it were measured with the same points.

5.23.4.2 Extract and from-scan construction methods

The extract construction method indicated by choosing the **Extract** *element* in the **Constructed** *element* and from-scan indicated by choosing the **FromScan** *element* in the **Constructed** *element* are both used to determine a feature measurement from a set of measurement points. These two methods are similar to the recompensated construction method in that all three create the feature measurement using a substitute feature algorithm on raw uncompensated point data from base features. But the extract and from-scan construction methods differ from the recompensated construction method in that even though the base feature can be identified, the point indexes on that feature cannot because the feature is typically measured with a scanning device resulting in a large point set variable in both size and density.

Both the extract and from-scan construction methods use a subset of points from the base feature. This subset is determined by the nominal extents of the feature being constructed. Therefore, the feature being constructed must be naturally bounded (e.g., an arc) or explicitly bounded (e.g., a plane with a polyline boundary).

The extract construction method is used when the dimensionality of the base feature and the feature being constructed are the same. Two-dimensional features can be extracted from a two-dimensional scan curve in the same plane; arcs and lines can be extracted from a planar curve scan inside a filleted pocket because the arcs and lines are coincident with the base feature. Similarly, three-dimensional features can be extracted from a three-dimensional scan surface. In both cases, the measurement points used in the construction are those inside the limits on the bounded feature.

But when a two-dimensional feature is derived from the measurement points of a three-dimensional scanned surface, the feature being constructed may not be coincident with the scanned data. As a result, search windows must be used to extend the two-dimensional feature out of its plane in order to capture point data sufficient to determine the feature measurement. The from-scan construction methods have *element*s like **SearchRadius** which allow for the definition of the search windows. (Some three dimensional feature types also have a from-scan construction method to be compatible with the DMIS 5.3 CONST (Input format 15) statement.)

5.23.4.3 Copy, cast, and transform construction methods

The copy, cast and transform construction methods perform the similar operation of copying measured data from one feature measurement to another feature measurement. The **NominalsCalculated** *element* is used to control the copying of nominal feature data.

In the case of the copy construction method the base feature and the feature being constructed must be of the same type and all the measured data elements from the base feature are copied unchanged to the feature being constructed. If the **NominalsCalculated** *element* is present and set to true then the nominal data elements are similarly copied from the base feature to the feature being constructed.

In the case of the cast construction method the base feature and the feature being constructed are not of the same type and all the measured data elements from the base feature may or may not have corresponding data elements in the feature being constructed. Only those measured data elements shared between the two feature types are copied unchanged from the base feature to the feature being constructed. If the **NominalsCalculated** *element* is present and set to true then any shared nominal data elements are similarly copied from the base feature to the feature being constructed.

In the case of the transform construction method the base feature and the feature being constructed must be of the same type and all the measured data elements from the base feature are copied to the feature being constructed. All location and orientation elements are transformed by the actual transform matrix of the specified coordinate system. All size, form, and other dimensional data remain unchanged. If the **NominalsCalculated** *element* is present and set to true then the nominal data elements are similarly copied from the base feature to the feature being constructed with the nominal location and orientation data being transformed by the nominal transform matrix of the specified coordinate system.

5.24 CharacteristicDesignators - encoding "balloon" numbers in QIF

Figure 10 shows a plate with ballooned tolerances. Characteristics that are key to a manufacturing process or to a part's usability (or have some other level of criticality) are often indicated on a drawing with an identifier in a circle or another shape. These balloon numbers and levels of criticality are accommodated in QIF with the concept of a characteristic designator.

The characteristic designator, defined by the ***CharacteristicDesignatorType***, performs four tasks. First, the required **Designator** *element* captures the balloon number. Second, the optional **UUID** *element* enables assigning a persistent identifier. Third, the optional **Criticality** *element* is used to indicate the level of importance of the characteristic. This is either one of an enumerated set of criticalities (MINOR, MAJOR, CRITICAL, KEY, and UNDEFINED) or a user-defined token that may have any value. Fourth, the optional **Balloon** *element* enables specifying the location and style of the balloon for visualization.

An *element* **CharacteristicDesignator** of ***CharacteristicDesignatorType*** can be found in three locations in the characteristics aspect hierarchy: on the definition, on the nominal and on the item. This allows for the sharing of a characteristic designator among several instances of a characteristic. The **CharacteristicDesignator** is also an optional *element* in the ***CompositeSegmentDefinitionBaseType***.

In Figure 10 the balloon number "1" is shared by the four holes in much the same way that the nominal diameter is shared. This relationship can be shown by using the optional

CharacteristicDesignator *element* on the characteristic nominal. This is the normal location for indicating the characteristic designator and criticality except in special circumstances.

When the characteristic designator is shared among several items and it is necessary to assign augmented labels or balloon numbers to each item, this is accomplished with the **CharacteristicDesignator** *element* on the characteristic item. A balloon number of “1” may result in individual characteristic designators with designators like 1_1, 1_2, etc.; 1A, 1B, etc.; or 1.1, 1.2, etc. depending on company standards.

When a box tolerance is used (like that shown in subclause 5.9.4) and such a tolerance is ballooned then, in practice, that designator may be shared among several different characteristics and even different characteristic types. In this case, the **CharacteristicDesignator** *element* on the characteristic definition is used.

If the characteristic designator spans several different characteristic types then the same characteristic designator must be re-defined for each characteristic type; QIF does not allow for characteristic definitions to be shared among characteristics of different types.

5.25 Attributes and Part Notes

The QIF information model enables users to insert otherwise unmodeled information in many places in QIF instance files by using **Attributes** and/or **PartNotes**. Note the capital **A** and bold font on **Attributes**; this is entirely different from XSDL *attribute*s.

An **Attributes** *element* is a list of **AttributeXXX** *element*s where the **XXX** indicates the type of data. The full list of **AttributeXXX** *element*s is shown in Figure 46.

Element Name	Data Type
AttributeBool	Boolean
AttributeQPId	QPIdType
AttributeI1	integer
AttributeI2	*list* of two integers
AttributeI3	*list* of three integers
AttributeD1	double
AttributeD2	*list* of two doubles
AttributeD3	*list* of three doubles
AttributeStr	string
AttributeTime	XML dateTime
AttributeUser	user defined data in a binary array or XML structure

Figure 46 – **Attributes** element names and types

Attributes are intended to be used to convey important information that is not representable elsewhere in the QIF model. Items that can be represented in the model (the nominal diameter of a circle, for example) should not be put into **Attributes**. The **AttributeStr** *element* can be used to convey any sort of information in natural language. The types that have an **Attributes** *element* in the QIF model are shown in Figure 47. Where there is an **Attributes** *element* in a base type, all derived types will have one, too.

Characteristics.xsd
- ***CharacteristicBaseType***
- ***CharacteristicGroupType***

Features.xsd
- ***FeatureBaseType***
- ***FeatureZoneBaseType***
- ***MeasuredPointSetType***

IntermediatesPMI.xsd
- ***AlgorithmType***
- ***AlignmentOperationBaseType***
- ***AngularToleranceDefinitionType***
- ***AngularToleranceType***
- ***AreaToleranceType***
- ***CoordinateSystemType***
- ***DatumType***
- ***DatumDefinitionType***
- ***DatumReferenceFrameType***
- ***DatumTargetType***
- ***ForceToleranceType***
- ***LinearToleranceDefinitionType***
- ***LinearToleranceType***
- ***MassToleranceType***
- ***MaterialType***
- ***OrganizationType***
- ***PressureToleranceType***
- ***SoftwareType***
- ***SpeedToleranceType***
- ***StandardType***
- ***SubstituteFeatureAlgorithmType***
- ***TemperatureToleranceType***
- ***ThreadSpecificationDetailedBaseType***
- ***TimeToleranceType***
- ***TransformInstanceType***

PrimitivesPD.xsd
- ***NodeWithIdBaseType***

QIFDocument.xsd
- ***QIFDocumentType***

QIFMeasurementResources.xsd
- ***EnvironmentalRangeType***
- ***MeasurementResourceBaseType***
- ***QualificationType***
- ***TemperatureType***
- ***UserDefinedAxisType***

QIFPlan.xsd
- ***ActionMethodBaseType***
- ***MeasurandBaseType***
- ***PlanElementBaseType***
- ***WorkInstructionBaseType***

QIFResults.xsd
- ***ActualComponentType***
- ***MeasurementResultsType***

QIFStatistics.xsd
- ***ControlMethodType***
- ***CorrectiveActionPlanType***
- ***StatisticalStudyPlanBaseType***
- ***StatisticalStudyResultsBaseType***

Statistics.xsd
- ***AssignableCauseType***
- ***CharacteristicStatsEvalBaseType***
- ***CorrectiveActionType***
- ***StatsBaseType***
- ***StatsNumericalBaseType***
- ***StatsWithTolNumericalBaseType***
- ***SummaryStatisticsType***

Traceability.xsd
- ***ActualProductTraceabilityType***
- ***EnvironmentType***
- ***InspectionTraceabilityType***
- ***ManufacturingProcessTraceabilityType***
- ***PreInspectionTraceabilityType***
- ***ProductTraceabilityType***

Visualization.xsd
- ***FontType***
- ***PMIDisplayType***

Figure 47 – Types with **Attributes** element

The ***PartNoteType*** models a note that needs to be displayed graphically and may contain only text. The ***ProductType*** has a **PartNoteSet** *element* of ***PartNoteSetType*** that is a list of **PartNote** *elements* of ***PartNoteType***. These are referenceable by **id**. The ***ProductDefinitionBaseType*** has a **PartNoteIds** *element* that is a list of **id**s of part notes. This allows part notes to be connected to specific products. The ***PartNoteType*** also has a **PartNoteIds** *element* so that part notes may be nested.

5.26 Detailed requirements

5.26.1 XML naming and design rules (NDR)

XML technology was chosen for QIF encoding because the basic XML specifications are supported as open, public domain, royalty-free standards, and because XML technology is very widely used.

The QIF information model is built using the XML Schema Definition Language(XSDL). That language was chosen because:

- XSDL has adequate expressive power for the basic structure of the model. It includes the ability to define complex types with *attributes* and *elements*.
- XSDL allows more specialized complex types to be derived from less specialized complex types, so that type hierarchies can be defined.
- XSDL has built-in data types and the ability to specialize them.
- XSDL permits the modular construction of models via an "include" capability.
- XSDL has a default instance file format (XML) with a set of rules for determining if an instance file conforms to a model. Moreover, XML instance files are human-readable as well as machine-readable.
- XSDL enables the model builder to define constraints that extend the rules for determining whether an instance file conforms to a model.
- XSDL is a widely accepted language, and the XML file format of instance files is even more widely accepted.
- Tools for determining whether a model is syntactically correct and consistent and whether a given instance file conforms to a given model are available free or for a moderate price.

Tools for generating computer code that may be incorporated in an application from a model built in XSDL are available free or for a moderate price, thus lowering the cost of implementation.

QIF version 3.0 uses all the capabilities of XSDL just mentioned:

- over 3000 complex types are defined.
- derivation hierarchies are built up to five levels deep.
- over 180 specialized data types are defined.
- the QIFDocument model is built from 22 modules.
- QIF includes hundreds of *key* and *keyref* constraints.

In this subclause, the prefix xs: is used to indicate terms that are part of the XML schema definition language (XSDL). The xs: prefix is also used in the schema files. The term xs:type used here is not part of XSDL but means either xs:complexType or xs:simpleType.

5.26.1.1 Naming conventions

Certain naming conventions have been used in the development of the QIF XML schemas.

With few exceptions names are descriptive and formed by concatenation without abbreviation. One exception to the rule against abbreviation is that "identifier" is shortened everywhere it appears to "id" or "Id". All concatenated words in a name except possibly the first start with an upper case letter. The rules for the case of the first letter of the first word are:

- All names of XML items except xs:attribute names start with an upper case letter. This includes names for xs:type, xs:element, xs:key, and xs:keyref. Example xs:type name: ***ArcFeatureNominalType***

- All xs:attribute names start with a lower case letter. Example xs:attribute name: id. All xs:type names end in "Type".
- All xs:key names end in "Key".
- All xs:keyref names end in "Keyref".
- All names of instantiable feature xs:types end in "FeatureItemType", "FeatureDefinitionType", "FeatureNominalType", or "FeatureMeasurementType".
- All names of instantiable characteristic xs:types end in "CharacteristicItemType", "CharacteristicDefinitionType", "CharacteristicNominalType", or "CharacteristicMeasurementType".
- All names of enumerated xs:types end in "EnumType".
- Names of xs:types that are parent xs:types not intended to be instantiated end in "BaseType", and the types are abstract.
- To a great extent, the names of xs:elements are formed from the name of the xs:type of the xs:element by removing the "Type" at the end. For example, the name of the xs:element whose xs:type is ***PlaneFeatureNominalType*** is **PlaneFeatureNominal**.

The name of an xs:element that is a reference to a single QIF **id** almost always ends in "Id" and is always of one of the derived types of ***QIFReferenceBaseType***. If the value of an xs:element is a list or array of **id**s, the xs:element name almost always ends in "Ids".

5.26.1.2 Design rules

A number of design rules have been followed in building the QIF schema files.

- All xs:type definitions are declared globally, i.e., as direct children of an xs:schema. In other words, no xs:type definition is embedded inside another xs:type definition or inside an xs:element. This convention is commonly called using the venetian blind pattern.
- Although the names of QIF xs:elements and xs:types are very descriptive, the precise meaning almost always requires explanation. The XML schema definition language includes an xs:documentation node type that may be used to put documentation into a schema. Documentation nodes must be preserved by XML tools. Comments may also be inserted in schema files but are not necessarily preserved by XML tools. Further details of documentation are given in subclause 7.2.
- All xs:types not intended to be instantiated are made abstract so as to be explicitly non-instantiable.
- When an xs:element is (or could be) declared to be of an abstract xs:type, there are three ways under the W3C rules in which the xs:element can be declared and instances of it put into instance files.
    - First, in the schema file, the xs:element may be declared to be of the abstract xs:type. In an instance file, instances of it may use the xs:element name for the abstract xs:type followed by an xsi:type declaration identifying one of the derived types. The xsi prefix is used for the standard XML instance namespace, http://www.w3.org/2001/XMLSchema-instance.
    - Second, in the schema file, rather than having a single xs:element declaration, an xs:choice of xs:elements of the various instantiable derived types may be used instead. In an instance file, one of the xs:elements in the xs:choice is used.
    - Third, in the schema file, the xs:element may be declared globally to be of the abstract xs:type and made to be the head of a substitution group. The instantiable derived xs:types are used as the xs:types for the other xs:elements in that substitution group. The xs:element for the abstract xs:type is then used via "ref" elsewhere in the schema file. In an instance file, the xs:element for a member of the substitution group headed by the abstract type is used.

The first method can make writing *key*/*keyref* constraints difficult since (1) those constraints are expressed using xs:element names, (2) the constraints need to distinguish among xs:types, and (3) the same xs:element name is used with different xs:types. It also requires the instance file to be more verbose (because the xsi:type declaration is needed). Hence, the first method is not used in QIF.

The second and third methods have a different xs:type for each xs:element name, so they support writing *key*/*keyref* constraints. The third method is used overwhelmingly but not exclusively in QIF. The second method is used in several places.

- QIF uses the namespace "http://www.qifstandards.org/xsd/qif3". The schema files in version 3.0 of QIF, use both no prefix and the prefix “t” for this namespace. The “t” is to satisfy the XSDL rule that xpaths require a prefix when a namespace is used.

5.26.1.3 Other naming and design items

XML instance files that conform to QIF must follow the rules for the conformance of instance files to XML schemas as defined by the W3C.

5.26.2 Annotation conventions

A great number of annotations (xs:annotation) containing documentation (xs:documentation) have been included in the QIF schema files in order to explain the full meaning of the various elements in the QIF models. The meanings of the documentation nodes are essential parts of the model, not merely suggestions that may be taken or not. The documentation nodes have been prepared according the following conventions:

- Every schema file has a documentation node near the top that provides a general description of the file.
- Every xs:type has a documentation node.
- Every xs:element outside of a substitution group has a documentation node.
- The xs:element at the head of every substitution group has a documentation node.
- Every different sort of xs:key and xs:keyref has a documentation node. In the case of features and characteristics, the xs:keys and xs:keyrefs occur in large (over 35) batches that are very similar. For these batches, only the first member is documented.
- Only one documentation node is used under an annotation node, unless there is an exceptional reason to include more than one.
- The text of each documentation node consists of one or more complete sentences. The text can stand alone without the context of the XML schema code. The name of the schema file, *attribute*, type, *element*, *key*, or *keyref* is repeated in the sentence.
- All abstract types are annotated as abstract. QIF BaseTypes are always abstract.
- If an element has multiple pieces of information, then "defines" or "gives" is used to describe the element. If the element is one piece of information (even if it has substructure), then "is" is used.

- When an element has maxOccurs=”0”, that indicates the element is optional. In this case, the word “optional” immediately precedes the name of the element in the documentation.

- When an element has maxOccurs="unbounded", or maxOccurs=”N” (where N is an integer that is 2 or greater) there may be more than one such element in an instance file. In these cases, the usage is "Each XXX element defines" rather than "The XXX element defines". Note: The default for elements is exactly one.

- If an attribute is optional, which is the default for attributes, the word “optional” immediately precedes the attribute name in the documentation.

- If an attribute has use=”required”, that means the attribute is required. In this case, the word “required” immediately precedes the attribute name in the documentation.

- If a type is a list of specific values, the word "enumerated" or "enumerates" is always in the annotation, e.g., "The optional, enumerated TimeDescription element describes the time relative to the inspection, at which the environment data is measured."

- Individual enumeration values in an enumerated type do not have documentation nodes, but if the meaning of the values is not obvious, either they are explained in the documentation node of the type, or a reference is given to a document giving the meanings.

- Documentation nodes do not describe structural aspects of the information model.

- When text notes can assist people in reading and understanding the schema file, text describing the file structure is placed in XML comments.

- The sentence, “This element is in an optional choice.” describes a situation in which the element itself is not optional, but the element is effectively optional because the choice it is in is optional.

6 QIF Library

6.1 Introduction

The QIF Library contains fifteen schema files that support the QIF Application schema files. The files and the area covered by each are:

- Auxiliary.xsd – types for auxiliary geometry
- Characteristics.xsd – characteristic types
- Expressions.xsd – expression types specific to QIF, used in the QIF rules and QIF plan applications
- Features.xsd – feature types
- GenericExpressions.xsd – domain-independent arithmetic, string, and Boolean expression types
- Geometry.xsd – geometric types (half of the core of CAD)
- IntermediatesPMI.xsd – intermediate PMI types
- Primitives.xsd – basic types used throughout QIF
- PrimitivesPD.xsd – basic numerical types used in product description
- PrimitivesPMI.xsd – basic PMI types
- Statistics.xsd – types that support QIF statistics
- Topology.xsd – topology types (the other half of the core of CAD)
- Traceability.xsd – quality measurement traceability information
- Units.xsd – types for units of measurement and values with units
- Visualization.xsd – types for graphical display of products, including PMI

The information covered by files in the QIF Library encompasses:

- boundary representation models of the sort found at the core of commercial computer-aided design (CAD) systems
  - geometry
  - topology
  - mesh representations
  - visualization
- product and manufacturing information (PMI)
  - geometric dimensioning and tolerancing information as expressed in ASME Y14.5M-2009
  - digital product definition data practices in ASME Y14.41-2012
  - geometrical product specifications expressed in ISO GPS related standards such as: ISO 1101.3
  - information expressed by ISO 22093:2011 and ANSI/DMIS 105.3 (DMIS 5.3) standards
  - first article inspection report information as defined in the AS9102B standard.
  - welding information
- rules to use in generating quality plans and programs
  - basic generic expressions
  - expressions specific to QIF rules
- quality statistics.

The QIF Library, and QIF in general cover both pre-measurement and post-measurement PMI information.

The sizes of these files vary widely, from 300 lines for Auxiliary.xsd to well over 23,000 lines for Features.xsd. The sizes of the subclauses of this clause vary similarly.

The descriptions of the contents of the schema files in this clause are high-level to give the reader a general understanding of each library file. For full details of every type, the HTML data dictionary (which provides two-way navigation) is most convenient to use. The XML schema files themselves contain equivalent information and for some uses are preferable to the HTML.

6.1.1 Changes in the QIF Library from QIF Version 2.1

The QIF Library for version 3.0 contains revised versions of the identically named fifteen schema files in the version 2.1 Library. Some of these files were changed radically and some only slightly, as described in this clause.

One significant change in terminology (which did not change meaning) appears in over a thousand places. That is, the term for the results of inspection has changed from “actual” to “measured”. As a result, the names of most *element*s and types containing "Actual" have been changed to instead contain "Measured" or “Measurement”. This change in itself has little impact on the capability of QIF but has a large impact on existing software implementations. Most of these changes were in components of *element* names and type names; for example, ***DiameterCharacteristicActualType*** became ***DiameterCharacteristicMeasurementType***. The two QIF library files most affected by that change are Characteristics.xsd and Features.xsd.

A significant change was made to the relationships between the four aspects of features. Previously a measured (actual) feature was required to reference a feature item, now the reference to the feature item is optional; previously a feature item could optionally reference a feature nominal, now the reference to the feature nominal is required. This change brings the QIF feature aspects more into harmony with manufacturing and inspection stages: the nominal and definition aspects come into being at the design stage, the item aspect comes into being at the inspection planning stage, and the measurement aspect comes into being at inspection execution stage.

A modest change expands the identification of the GD&T standard used to include the ability to define associated standards and specifications. Associated is the ability to explicitly define software application and algorithms for substitute feature algorithms and statistical calculations.

A minor addition is the ability to optionally include a simple list of measured values used in a statistical calculation in place of references to measured characteristics that contain those values.

A detailed list of all changes in each schema file will be made available.

6.2 Auxiliary.xsd

The Auxiliary.xsd schema file contains types for:

- auxiliary geometry (points, lines and planes)
- CAD coordinate systems

Auxiliary geometry is geometry items that a user might want to see that are not in the boundary representation of an object, such as the axis of a cylinder. This includes, for example, ***PointAuxiliaryType***, ***LineAuxiliaryType***, and ***PlaneReferenceType***.

CAD coordinate systems are coordinate systems defined in CAD and included in QIF to support CAD to CAD and CAD to DME transfer.

6.3 Characteristics.xsd

6.3.1 **Characteristics *element***

Characteristic information in the QIF data model is modeled in the Characteristics.xsd schema file. The top level type defined in the file is the ***CharacteristicAspectsListsType***. This is used in the QIFDocument schema as the type of the **Characteristics** *element*, as shown in Figure 48.

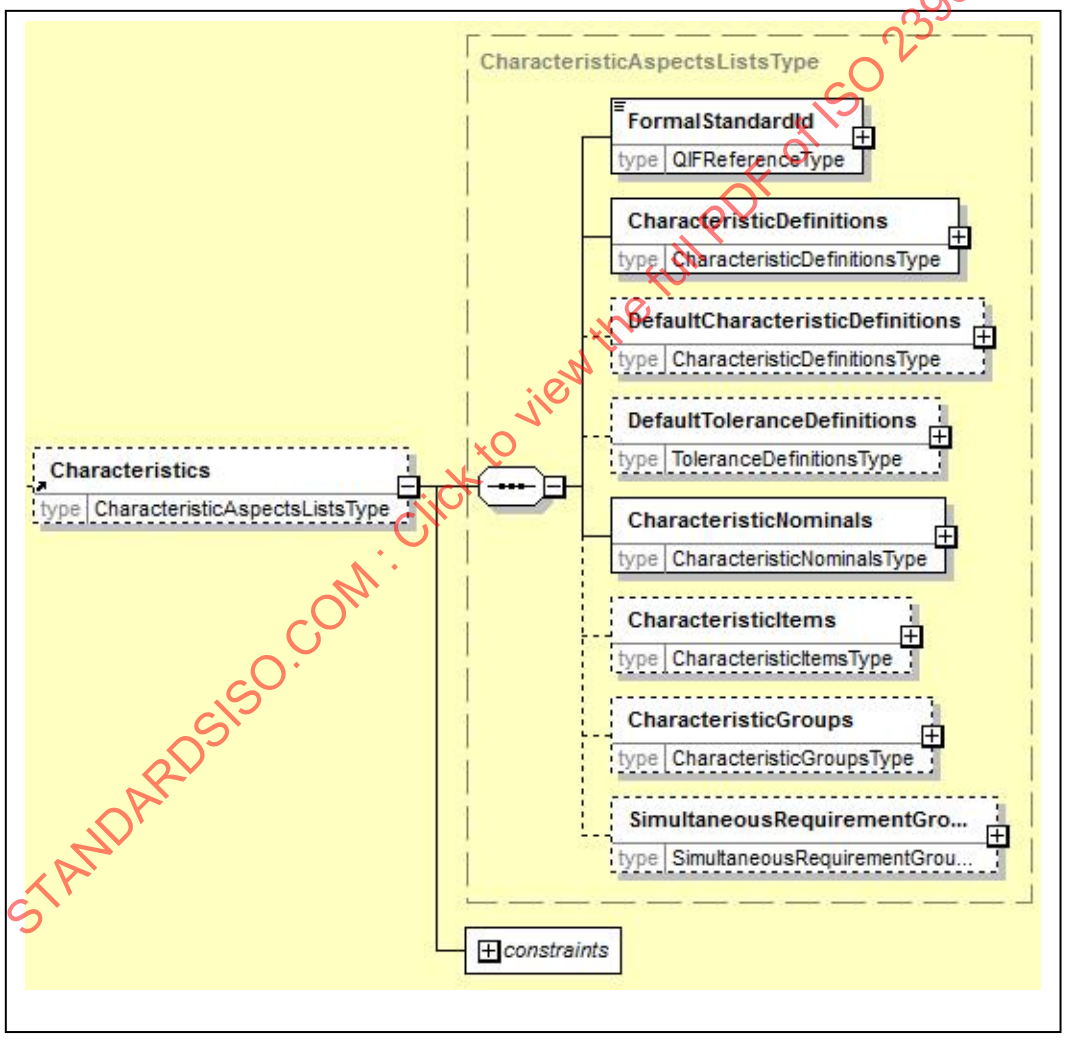


Figure 48 – **Characteristics** element

6.3.2 **Characteristic definitions, nominals, items, and measurements**

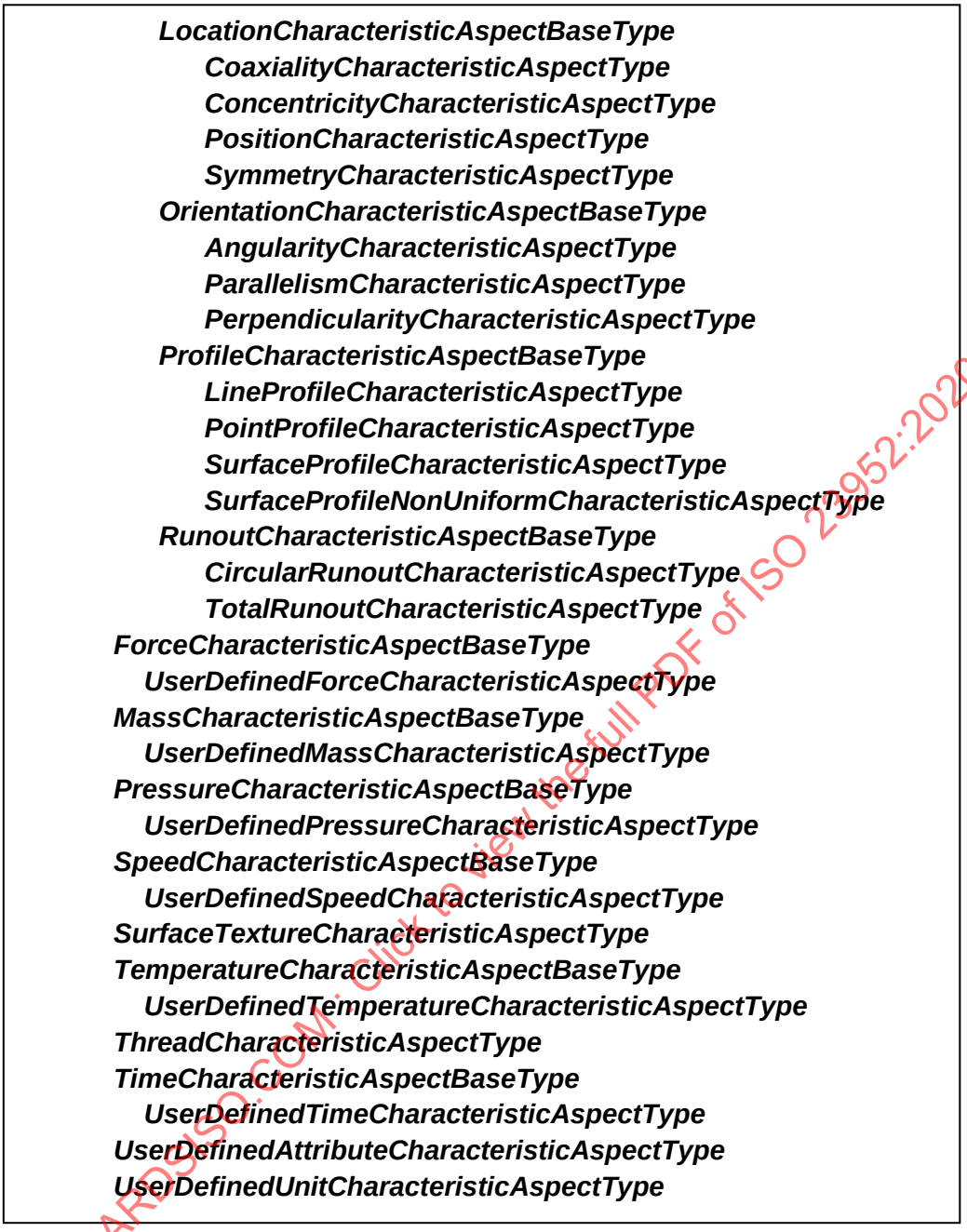
Characteristics have four aspects: *definition*, *nominal*, *measurement*, and *item*. How the aspects relate to one another is described in Subclause 6.9.3. Lists of three of the four aspects are in the

Characteristics *element*. The missing aspect, *measurement*, is in measurement results. The **Characteristics** *element* also contains a formal standard identifier, default characteristics, default tolerances, characteristic groups, and simultaneous requirement groups.

The XML Schema language definitions for characteristics form a type hierarchy by using XSDL *extension*s to derive more specialized types from more general types. There are four hierarchies, one for each aspect. Each of the four is headed by ***CharacteristicBaseType***. The prototype followed by each of the four hierarchies is shown in Figure 49. The word ***Aspect*** is shown in the names in the figure where one of the four aspect names (***Item***, ***Definition***, ***Nominal***, or ***Measurement***) would appear. In the figure, inheritance is indicated by indenting an additional level. For example, the inheritance path to ***CylindricityCharacteristicAspectType*** is:

CharacteristicBaseType
CharacteristicAspectBaseType
GeometricCharacteristicAspectBaseType
FormCharacteristicAspectBaseType
CylindricityCharacteristicAspectType

The ***CharacteristicBaseType*** and other types whose name includes ***Base*** are abstract types and cannot be instantiated. The hierarchy has 73 leaf types that are not abstract and can be instantiated. Most of the leaf types exist because they are included in ASME GD&T, ISO GPS, or AWS A2.4:2012 (for welding). Nine of the types exist only so that users can create user-defined characteristics for which there is a defined unit (***Angular***, ***Area***, ***Force***, ***Linear***, ***Mass***, ***Pressure***, ***Speed***, ***Temperature***, and ***Time***). User-defined characteristics not using any of those units may be defined with either ***UserDefinedAttributeCharacteristicAspectType*** or ***UserDefinedUnitCharacteristicAspectType***.



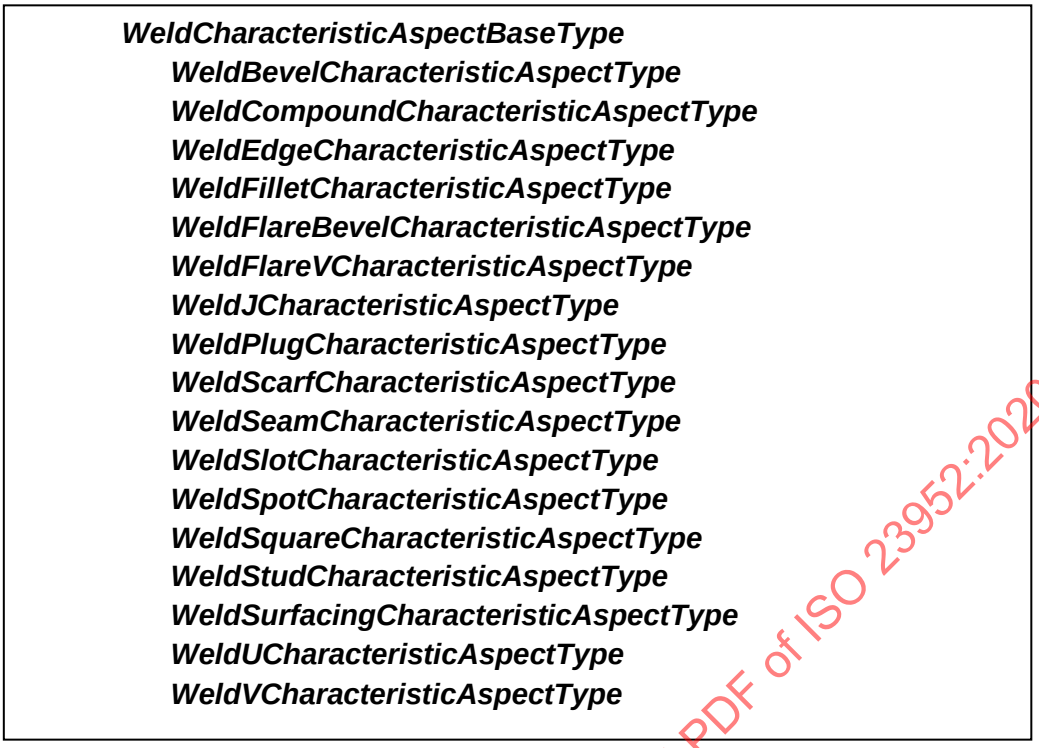


Figure 49 – Characteristic types

A small deviation from the hierarchy shown in Figure 49 is that ***DiameterCharacteristicDefinitionType*** is derived directly from ***DimensionalCharacteristicAspectBaseType*** in order that an option to use limits and fits may be included.

6.3.3 DefaultCharacteristicDefinitions

The **DefaultCharacteristicDefinitions** *element* shown in Figure 48 is list of default or “unless otherwise specified” characteristic definitions.

6.3.4 DefaultToleranceDefinitions

The **DefaultToleranceDefinitions** *element* shown in Figure 48 is a list of tolerance definitions with ids. The list may be a mix of **LinearTolerance** and **AngularTolerance** *element*s. A tolerance in the list can be referenced by id from a characteristic definition that uses a tolerance of the correct type. *Key*/*keyref* checks are included for this.

6.3.5 CharacteristicGroups

The **CharacteristicGroups** *element* shown in Figure 48 is a list of characteristic groups. Each of these is of ***CharacteristicGroupType*** or its only derived type, ***CharacteristicManufacturingProcessGroupType***. A characteristic group may be transformed as a whole and have a status as a whole.

6.3.6 Constraint checking for characteristics

Between QIFDocument.xsd and Characteristics.xsd there are over 200 *keyref*s for checking references between the aspects of characteristics. The three sorts of references between characteristics that these check are:

- measurement to item
- item to nominal
- nominal to definition

For example, if the **NominalId** *element* of an instance of ***SurfaceProfileCharacteristicItemType*** is populated, its value must be the **id** of a ***SurfaceProfileCharacteristicNominalType***.

The nominal-to-definition *keyref* checks are in Characteristics.xsd. The other two types of *keyref* checks are in QIFDocument.xsd.

QIF enables referencing objects in external QIF instance files. *Keyref*s apply only within an instance file. All of the 200-plus *keyref* constraints between aspects of characteristics within an instance file have been implemented between instance files using XSLT, as described in Clause 5.

6.3.7 ToleranceZones

Characteristics.xsd defines types that describe the shapes of tolerance zones used in the characteristics as shown in Figure 50.

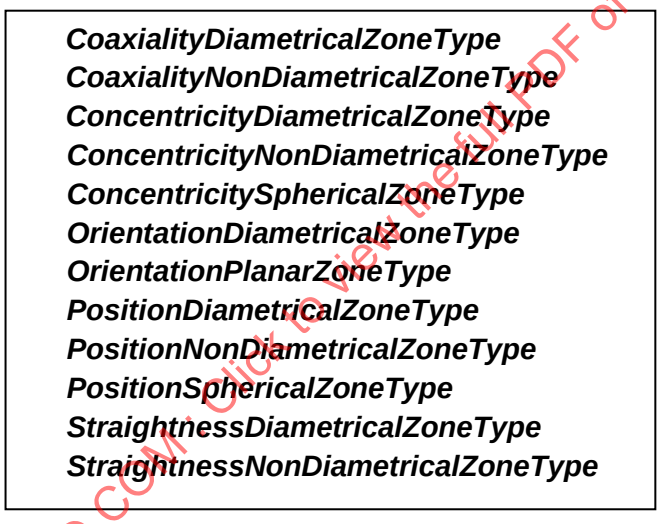


Figure 50 – Tolerance zone types

6.3.8 Substitution groups in Characteristics.xsd

Characteristics.xsd defines five *substitutionGroup*s. One of these is for **CharacteristicGroup**, and has only one substitutable *element*. Four *substitutionGroup*s are for characteristic aspects. Each of those has a head whose type is the base type for the aspect plus 73 substitutable *element*s (one for each instantiable characteristic type).

6.4 Expressions.xsd

The Expressions.xsd schema file defines QIF-specific expressions for supporting QIFRules.xsd. Expressions.xsd *include*s GenericExpressions.xsd (discussed in subclause 6.6), so the capabilities of GenericsExpressions.xsd are available for rules.

Expressions are involved in QIF rules when a feature (possibly associated with a characteristic) is being examined for one of three purposes: (1) picking the number of target hit points on the feature for probing, (2) picking the strategy (pattern) for hit points, or (3) picking a substitute feature algorithm for fitting the feature to hit points. The (possibly automated) system that is making the decisions is working in an environment in which sampling category has been set. The system is expected to have access to information about the feature and the characteristic (if there is one). Sampling category is discussed further in Clause 10.

Expressions are also used in rules for selecting dimensional metrology equipment (DMEs).

6.4.1 Types Defined in Expressions.xsd

The following expression types are derived from ***BooleanExpressionBaseType*** (which is defined in GenericExpressions.xsd).

CharacteristicIsType
FeatureIsDatumType
FeatureIsInternalType
FeatureTypeIsType
SamplingCategoryIsType
ShapeClassIsType

The following expression types are derived from ***ArithmeticExpressionBaseType*** (which is defined in GenericExpressions.xsd).

ArithmeticParameterBaseType
- ***ArithmeticCharacteristicParameterType***
- ***ArithmeticDMEParameterType***
- ***ArithmeticFeatureParameterType***
- ***ArithmeticPartParameterType***

CharacteristicToleranceType
FeatureAreaType
FeatureLengthType

6.4.2 Substitution groups in Expressions.xsd.

The *substitutionGroup*s of **BooleanExpression** and **ArithmeticExpression** defined in GenericExpressions.xsd are extended in Expressions.xsd. There is an *element* for each of the types listed in subclause 6.4.1 above that is not a base type. In all cases, the name of the *element* is the name of the type with “Type” removed from the end of the name.

6.5 Features.xsd

6.5.1 Features *element*

Measurement features are generated by analyzing the association between (1) geometric dimensions and tolerances (GD&T) and (2) product design geometric elements as specified in design models or drawings. In the QIF realm and widely accepted in the metrology and manufacturing quality fields, features are defined as the underlying targets to which GD&T is applied. These features are often defined as points, curves, planes, spheres, etc. that can actually

be seen on the physical part to be measured. The characteristics defined in Characteristics.xsd apply primarily to features; for example a diameter characteristic might apply to a circle or cylinder, and a perpendicularity characteristic might apply to two planes.

The Features.xsd schema file defines types that describe dimensional metrology features. The top level type defined in the file is the ***FeatureAspectsListsType***. This is used in the QIFDocument schema as the type of the **Features** *element*, as shown in Figure 51.

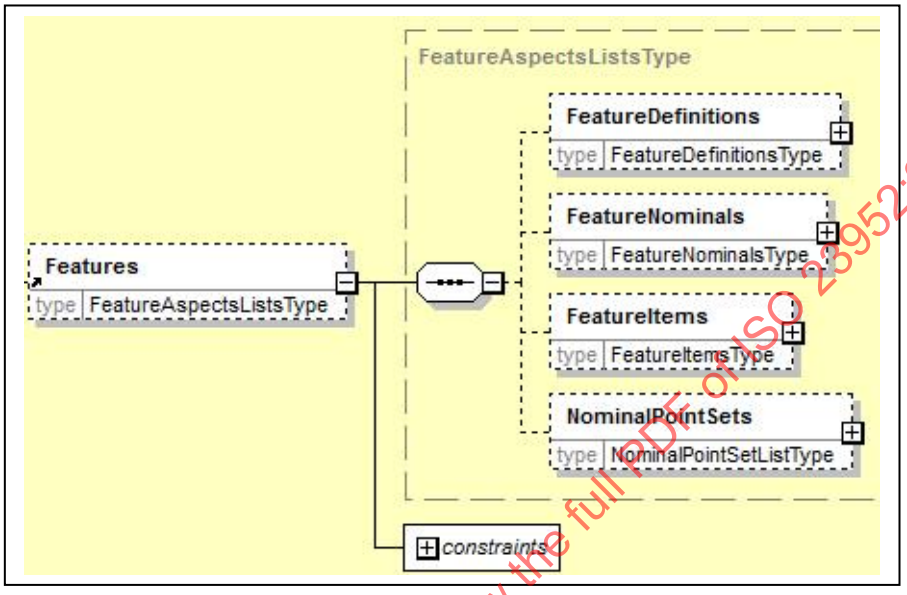


Figure 51 – **Features** element

6.5.2 Feature types

Features.xsd defines instantiable feature types plus base feature types, as shown in Figure 52. Features have four aspects: *definition*, *nominal*, *measurement*, and *item*. Each feature type in Figure 52 that has “Aspect” in its name has a type for each of the four aspects, except that there is no measurement type for the five pattern feature types. Lists of three of the aspects are included in the ***FeatureAspectsListsType***, as shown in Figure 51. A list of the fourth aspect, *measurement*, is included in measurement results.

Figure 52 – Feature Types (below)

FeatureBaseType
- ***NonShapeFeatureBaseType***
  - ***NonShapeFeatureAspectBaseType***
    - ***MarkingFeatureAspectType***
    - ***OtherNonShapeFeatureAspectType***
- ***ShapeFeatureBaseType***
  - ***ShapeFeatureAspectBaseType***
    - ***CurveFeatureAspectBaseType***
      - ***CircleFeatureAspectType***
      - ***CircularArcFeatureAspectType***
      - ***EllipseFeatureAspectType***
      - ***EllipticalArcFeatureAspectType***
      - ***ElongatedCircleFeatureAspectType***
      - ***LineFeatureAspectType***
      - ***OppositeAngledLinesFeatureAspectType***
      - ***OppositeParallelLinesFeatureAspectType***
      - ***OtherCurveFeatureAspectType***
      - ***PointDefinedCurveFeatureAspectType***
    - ***GroupFeatureAspectType***
      - ***PatternFeatureAspectBaseType***
        - ***PatternFeatureCircleAspectType***
        - ***PatternFeatureCircularArcAspectType***
        - ***PatternFeatureLinearAspectType***
        - ***PatternFeatureParallelogramAspectType***
    - ***OtherShapeFeatureAspectType***
    - ***PointFeatureAspectBaseType***
      - ***EdgePointFeatureAspectType***
      - ***PointFeatureAspectType***
    - ***SpecifiedFeatureAspectBaseType***
      - ***ThreadedFeatureAspectType***
    - ***SurfaceFeatureAspectBaseType***
      - ***ConeFeatureAspectType***
      - ***ConicalSegmentFeatureAspectType***
      - ***CylinderFeatureAspectType***
      - ***CylindricalSegmentFeatureAspectType***
      - ***ElongatedCylinderFeatureAspectType***
      - ***ExtrudedCrossSectionFeatureAspectType***
      - ***OppositeAngledPlanesFeatureAspectType***
      - ***OppositeParallelPlanesFeatureAspectType***
      - ***OtherSurfaceFeatureAspectType***
      - ***PlaneFeatureAspectType***
      - ***PointDefinedSurfaceFeatureAspectType***
      - ***SphereFeatureAspectType***
      - ***SphericalSegmentFeatureAspectType***
      - ***SurfaceOfRevolutionFeatureAspectType***
      - ***ToroidalSegmentFeatureAspectType***
      - ***TorusFeatureAspectType***

As shown in Figure 51, the **Features** *element* also contains the **NominalPointSets** *element*.

Feature descriptions in natural language are given in the in-line documentation of the feature types. Diagrams of some feature types are provided in Clause 5. The Features.xsd file also:

- defines methods of constructing features
- contains the *key*/*keyref* pairs that constrain references from feature nominals to feature definitions
- defines the *subsitutionGroup*s for feature aspects.

All but one of the feature types in DMIS 5.3 have an equivalent in QIF. QIF has some that do not have an equivalent in the DMIS 5.3 standard. A comparison of the shape feature definitions in QIF and DMIS is shown in Figure 53. QIF non-shape features are not shown.

QIF Feature	Equivalent DMIS Feature
Circle	CIRCLE
CircularArc	ARC (format 1)
Cone	CONE
ConicalSegment	CONRADSEGMNT
Cylinder	CYLNDR
CylindricalSegment	CYLRADSEGMNT
EdgePoint	EDGEPT
Ellipse	ELLIPS
EllipticalArc	*no equivalent*
ElongatedCircle	*no equivalent*
ElongatedCylinder	ELONGCYL
ExtrudedCrossSection	*no equivalent*
Generic	GEOM, OBJECT
Group	COMPOUND
Pattern	PATERN
Line	LINE
OppositeAngledLines	CPARLN
OppositeParallelLines	CPARLN
OppositeAngledPlanes	PARPLN, SYMPLN
OppositeParallelPlanes	PARPLN, SYMPLN
OtherCurveFeature	GEOM, OBJECT
OtherShapeFeature	GEOM, OBJECT
OtherSurfaceFeature	GEOM, OBJECT
Plane	PLANE
Point	POINT
PointDefinedCurve	GCURVE
PointDefinedSurface	GSURF
Sphere	SPHERE
SphericalSegment	SPHRADSEGMNT
SurfaceOfRevolution	REVSURF
Threaded	*no equivalent*
ToroidalSegment	TORRADSEGMNT
Torus	TORUS
no equivalent	RCTNGL

Figure 53 – Comparison of feature definitions in QIF and DMIS

QIF features are described using Cartesian coordinates. Notes may be attached to any of the feature aspects.

6.5.3 Constraint checking for features

Between QIFDocument.xsd and Features.xsd there are over 100 *keyref*s for checking references between the aspects of features. The three sorts of references between features that these check are:

- measurement to item
- item to nominal
- nominal to definition.

For example, if the **FeatureNominalId** *element* of an instance of ***CircleFeatureItemType*** is populated, its value must be the **id** of a ***CircleFeatureNominalType***.

The nominal to definition *keyref* checks are in Features.xsd. The other two types of *keyref* checks are in QIFDocument.xsd.

QIF enables referencing objects in external QIF instance files. *Keyref*s apply only within an instance file. All of the 100-plus *keyref* constraints between aspects of features within an instance file have been implemented between instance files using XSLT, as described in Clause 5.

6.5.4 Feature construction methods

As described in subclause 6.5.4, the Features.xsd schema file provides construction methods for features. These are defined for all instantiable shape features except GroupFeature and its derived types. There are several stereotypical construction methods that may be applied to the shape feature types that may be constructed.

- copy – all
- transform – all except OtherCurve, OtherShape, and OtherSurface
- cast – all except PointDefinedCurve, PointDefinedSurface, OtherCurve, OtherShape, and OtherSurface
- best fit – all except Point, EdgePoint, OtherCurve, OtherShape, and OtherSurface
- recompensated – all except Point and EdgePoint, OtherCurve, OtherShape, and OtherSurface
- from scan – Circle, CircularArc, Cone, Cylinder, EdgePoint, Ellipse, EllipticalArc, Line, OppositeAngledLines, OppositeParallelLines, OppositeAngledPlanes, OppositeParallelPlanes, PointDefinedCurve, Point, Sphere, Torus
- extract – CircularArc, Line, Plane, PointDefinedCurve, PointDefinedSurface
- intersection – Circle, CircularArc, Ellipse, EllipticalArc, Line, OppositeParallelLines, OppositeAngledLines, Point
- projection – CircularArc, Circle, Ellipse, EllipticalArc, Line, OppositeAngledLines, OppositeParallelLines, Point
- tangent through – Circle, Line, Plane

In addition to these generally applicable construction methods, several other construction methods are applicable to one or a few feature types.

- Circle – FromCone, Tangent
- Line – Midline, Parallel, Perpendicular
- Plane – Midplane, Offset, Parallel, Perpendicular, Through
- Point – FromCone, CenterOfGravity, Pierce, Midpoint, MovePoint, MovePointVector, MovePointAxis, Extreme
- Threaded – FromCylinder

6.5.5 Substitution groups for features

The Features.xsd schema file defines four *substitutionGroup*s (one for each feature aspect), each of which has a head that is the base type for the aspect, plus 37 members (one for each instantiable feature type).

6.6 GenericExpressions.xsd

The GenericExpressions.xsd schema file defines Boolean expression types, simple arithmetic expression types, and very simple string expression (actually *token*) types. The generic expressions are not specific to QIF.

Expressions defined in GenericExpressions.xsd are used in Expressions.xsd, QIFPlan.xsd, and QIFRules.xsd. Expressions.xsd extends the *substitutionGroup*s of **BooleanExpression** and **ArithmeticExpression**, as described in Subclause 6.4.2. QIFPlan.xsd extends the *substitutionGroup* of **TokenExpression**.

All expressions are modeled in XSDL (the same way everything else is modeled), not in condensed symbolic notation. This makes instances of the expressions verbose but very easy to process. No separate expression parser is needed.

6.6.1 Arithmetic Expressions

All instantiable arithmetic expression types are derived directly or indirectly from the *abstract* (non-instantiable) ***ArithmeticExpressionBaseType***. Instantiable arithmetic expression types defined in the file that represent arithmetic operators are:

- ***NegateType***
- ***MaxType***
- ***MinType***
- ***PlusType***
- ***MinusType***
- ***TimesType***
- ***DividedByType***

ArithmeticConstantType is also an arithmetic expression type.

A *substitutionGroup* headed by the **ArithmeticExpression** *element* is defined containing an *element* for each of the types listed above. The *element* names are the type names with “Type” removed.

MaxType, ***MinType***, ***PlusType***, and ***TimesType*** take two or more arguments. ***MinusType*** and ***DividedByType*** take exactly two arguments. ***NegateType*** takes one argument.

6.6.2 String Expressions

String expressions are defined using the *token* type, which is a string in which extra white space is ignored (so that, for example, “ a string ” is considered to be the same as “a string”). There is an *abstract* ***TokenExpressionBaseType*** from which ***TokenConstantType*** is derived. There is a *substitutionGroup* headed by the **TokenExpression** *element* and containing the **TokenConstant** *element*.

It is intended to extend the range of string expressions in future versions of QIF.

6.6.3 Boolean Expressions

The following common comparisons between arithmetic expressions that return a Boolean result are defined:

- ***ArithmeticEqualType***
- ***GreaterThanType***
- ***GreaterOrEqualType***
- ***LessThanType***
- ***LessOrEqualType***

Boolean expression types taking Boolean arguments are defined:

- ***BooleanEqualType***
- ***ConstantIsType***
- ***NotType***
- ***AndType***
- ***OrType***

One Boolean expression taking two *token* (i.e., string) arguments is defined; that is ***TokenEqualType***.

The Boolean expression types listed above are all instantiable. In addition to those types, non-instantiable *abstract* base types are defined, headed by ***BooleanExpressionBaseType***. A *substitutionGroup* headed by the **BooleanExpression** *element* is defined. Its members are *element*s corresponding to the types listed above. The names of the *element*s are the same as the names of the types, with “Type” removed.

The expression types ***ConstantIsType***, ***NegateType***, and ***NotType*** take one argument. ***AndType*** and ***OrType*** take two or more arguments. All other Boolean expression types take exactly two arguments.

6.7 Geometry.xsd

The Geometry.xsd schema file defines the geometry types shown in Figure 54 and the geometry set types shown in Figure 55. Subhierarchies of the derivation hierarchy shown in Figure 54 are also shown in Clause 7. The role of the geometry types in product definition is covered thoroughly in Clause 7 and is not described here.

The Geometry.xsd file also defines a mathematical core type for each of the instantiable (leaf) types in Figure 54 and two enumerations.

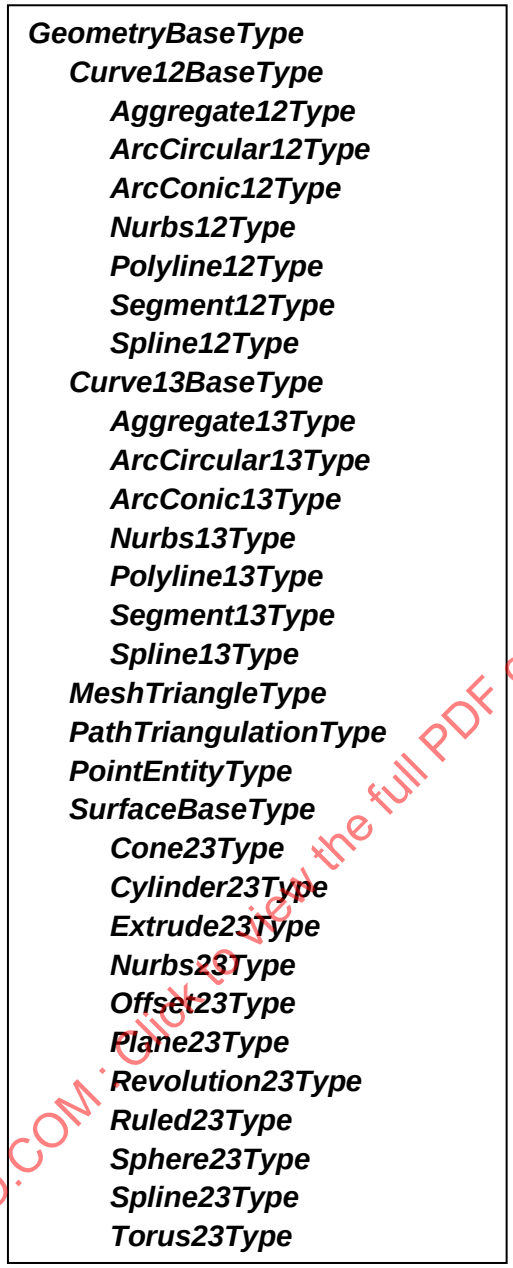


Figure 54 – Individual geometry types

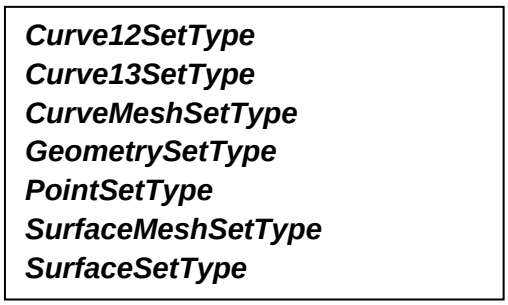


Figure 55 – Geometry set types

The ***GeometrySetType*** in Figure 55 consists of *element*s of the other six types.

6.8 IntermediatesPMI.xsd

The IntermediatesPMI.xsd schema file defines over 130 miscellaneous types used in other schema files. Many of these types either define ids or use ids. Twenty-four of them are enumerations.

The hierarchy of alignment operation types shown in Figure 56 is included along with a *substitutionGroup* for them.

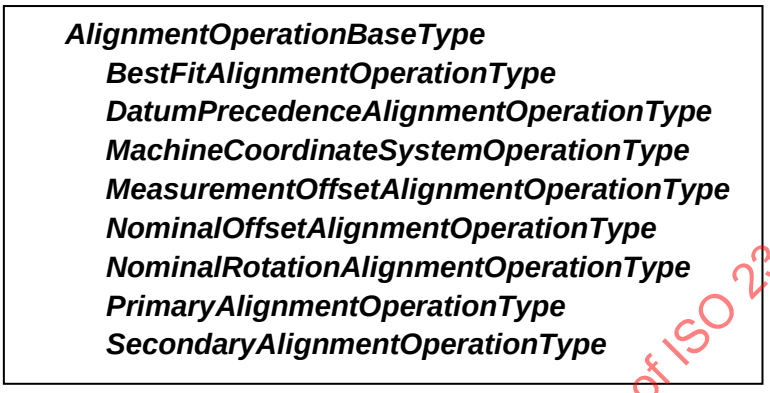


Figure 56 – Alignment operations

The types dealing with datums and datum reference frames shown in Figure 57 are defined.

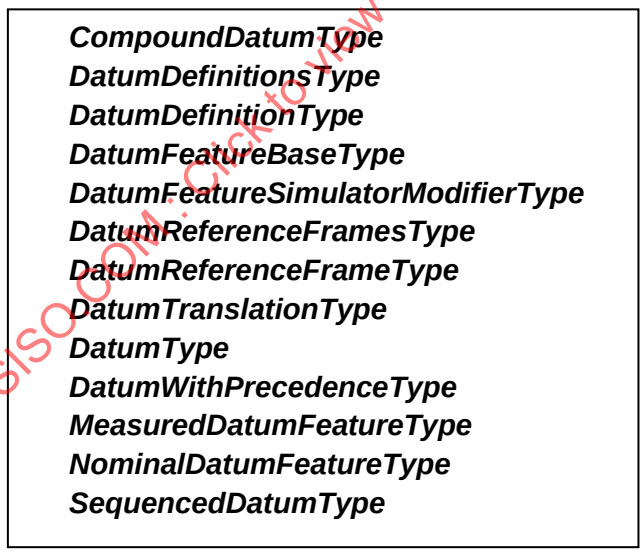


Figure 57 – Datums and datum reference frames

A number of ISO-specific types are defined as shown in Figure 58. In addition to these ISO-specific types, there are many ISO-specific *element*s in various types. For example, the DatumType has the following ISO-Specific *element*s: **ProjectedDatum**, **DiameterModifier**, **SectionModifier**, **ContactingFeature**, **DistanceVariable**, **DatumFixed**, **ReducedDatum**, **ConstrainOrientation**, **ConstrainSubsequent**. Further information about ISO-specific items is given in Clause 5.

CollectionPlaneType
DiameterModifierEnumType
IntersectionPlaneEnumType
IntersectionPlaneType
ISODegreeOfFreedomEnumType
ModifyingPlaneEnumType
OrientationPlaneType
ReducedDatumEnumType
SectionModifierEnumType

Figure 58 – ISO-specific types

Intermediates.xsd defines other types dealing, among other things, with the following: algorithms, events, external references, threads, standards, coordinate systems, tolerances, transforms, materials, and points that are to be measured or were measured.

6.9 Primitives.xsd

The Primitives.xsd schema file defines 90 miscellaneous types used in other schema files. Types in Primitives.xsd might be used in CAD as well as in quality measurement. This includes, for example: ***PointType***, ***UnitVectorType***, and ***TransformMatrixType***. Types in this file do not have an **id** *attribute*.

The QIF types for identifiers and references to identifiers, both internal to a QIF instance file and external to it, are defined in Primitives.xsd.

Nineteen of the types have an *attribute* named **n** or **count** and represent collections such as arrays and lists. In these types, **n** or **count** is the number of items.

The definitions of the ***AttributeBaseType*** and eleven derived types are included along with the definitions of ten global *element*s (one of each type) in a *substitutionGroup*. These are made available to users for including data in an instance file that is not modeled elsewhere in QIF. The eleven *element*s are:

- **AttributeBool** – an xs:boolean
- **AttributeStr** – an xs:string
- **AttributeQPId** – a QPId for identifying the containing *element*
- **AttributeTime** – an xs:dateTime
- **AttributeI1** – a single xs:integer
- **AttributeI2** – two xs:integers
- **AttributeI3** – three xs:integers
- **AttributeD1** – one xs:double
- **AttributeD2** – two xs:doubles
- **AttributeD3** – three xs:doubles
- **AttributeUser** – binary data or any user-defined XML data.

An **Attributes** *element* including any number of these *element*s may be inserted in an instance file at many places. All the **Attributes** *element*s are optional.

6.10 PrimitivesPD.xsd

The PrimitivesPD.xsd schema file defines 3 miscellaneous *simpleTypes* and 8 *complexTypes* that are used in CAD product definition but not in PMI.

6.11 PrimitivesPMI.xsd

The PrimitivesPMI.xsd schema file defines miscellaneous types that do not have ids and are not used in geometry or topology but are used elsewhere. This includes 15 enumerations and 45 *complexType*s.

Over a dozen types support statistics and have “Stats” in their names. Four of the *complexType*s are scale types. Another four are used for datum targets. Nine *complexType*s offer a choice between an enumeration value and a user-defined value. These are shown in Figure 59.

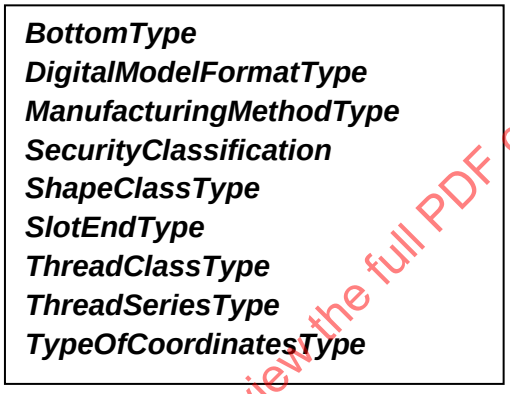


Figure 59 – Types with enumeration or user definition

6.12 Statistics.xsd

The Statistics.xsd schema file supports the QIFStatistics application. It defines type hierarchies and miscellaneous other types used in quality statistics. This includes 178 *complexType*s and 9 *simpleType*s. Further information about QIFStatistics is given in Clause 12.

6.12.1 Basic Statistics Types

The Statistics.xsd file defines a hierarchy of basic statistics types as shown in Figure 60.

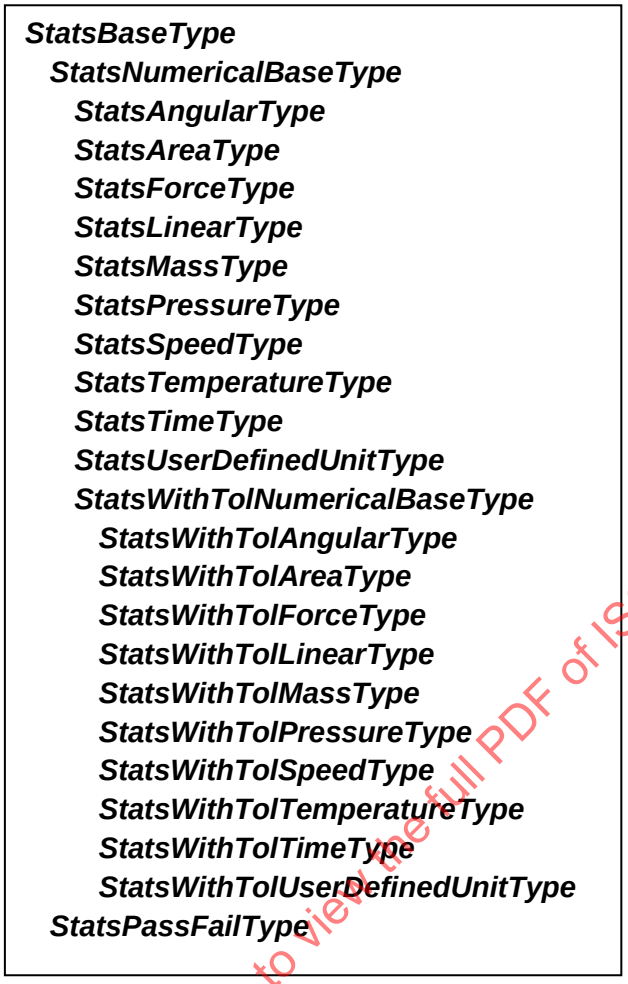


Figure 60 — Basic Statistics Types

6.12.2 Characteristic Statistics Evaluation Types

The file defines a hierarchy of characteristic statistics evaluation types as shown in Figure 61. This is very similar to the characteristics hierarchy but omits some of the intermediate types of that hierarchy. Statistics may be reported for each one of these whose name does not include ***Base***, thus offering various aggregated statistics in addition to statistics for the leaf node types.

Figure 61 – **CharacteristicStatsEval** types (continued on next page)

CharacteristicStatsEvalBaseType
- ***AngularCharacteristicStatsEvalType***
  - ***AngularCoordinateCharacteristicStatsEvalType***
  - ***AngleCharacteristicStatsEvalType***
  - ***AngleFromCharacteristicStatsEvalType***
  - ***AngleBetweenCharacteristicStatsEvalType***
- ***GeometricCharacteristicStatsEvalType***
  - ***FormCharacteristicStatsEvalBaseType***
    - ***CircularityCharacteristicStatsEvalType***
    - ***ConicityCharacteristicStatsEvalType***
    - ***CylindricityCharacteristicStatsEvalType***
    - ***EllipticityCharacteristicStatsEvalType***
    - ***FlatnessCharacteristicStatsEvalType***
    - ***OtherFormCharacteristicStatsEvalType***
    - ***SphericityCharacteristicStatsEvalType***
    - ***StraightnessCharacteristicStatsEvalType***
    - ***ToroidicityCharacteristicStatsEvalType***
  - ***LocationCharacteristicStatsEvalType***
    - ***CoaxialityCharacteristicStatsEvalType***
    - ***ConcentricityCharacteristicStatsEvalType***
    - ***PositionCharacteristicStatsEvalType***
    - ***SymmetryCharacteristicStatsEvalType***
  - ***OrientationCharacteristicStatsEvalType***
    - ***AngularityCharacteristicStatsEvalType***
    - ***ParallelismCharacteristicStatsEvalType***
    - ***PerpendicularityCharacteristicStatsEvalType***
  - ***ProfileCharacteristicStatsEvalBaseType***
    - ***LineProfileCharacteristicStatsEvalType***
    - ***PointProfileCharacteristicStatsEvalType***
    - ***SurfaceProfileCharacteristicStatsEvalType***
    - ***SurfaceProfileNonUniformCharacteristicStatsEvalType***
  - ***RunoutCharacteristicStatsEvalBaseType***
    - ***CircularRunoutCharacteristicStatsEvalType***
    - ***TotalRunoutCharacteristicStatsEvalType***
- ***LinearCharacteristicStatsEvalType***
  - ***ChordCharacteristicStatsEvalType***
  - ***ConicalTaperCharacteristicStatsEvalType***
  - ***CurveLengthCharacteristicStatsEvalType***
  - ***DepthCharacteristicStatsEvalType***

LinearCharacteristicStatsEvalType (continued)
- ***DiameterCharacteristicStatsEvalType***
- ***DistanceBetweenCharacteristicStatsEvalType***
- ***DistanceFromCharacteristicStatsEvalType***
- ***FlatTaperCharacteristicStatsEvalType***
- ***HeightCharacteristicStatsEvalType***
- ***LengthCharacteristicStatsEvalType***
- ***LinearCoordinateCharacteristicStatsEvalType***
- ***RadiusCharacteristicStatsEvalType***
- ***SphericalDiameterCharacteristicStatsEvalType***
- ***SphericalRadiusCharacteristicStatsEvalType***
- ***SquareCharacteristicStatsEvalType***
- ***ThicknessCharacteristicStatsEvalType***
- ***WidthCharacteristicStatsEvalType***

SurfaceTextureCharacteristicStatsEvalType
ThreadCharacteristicStatsEvalType
UserDefinedAngularCharacteristicStatsEvalType
UserDefinedAreaCharacteristicStatsEvalType
UserDefinedAttributeCharacteristicStatsEvalType
UserDefinedForceCharacteristicStatsEvalType
UserDefinedLinearCharacteristicStatsEvalType
UserDefinedMassCharacteristicStatsEvalType
UserDefinedPressureCharacteristicStatsEvalType
UserDefinedSpeedCharacteristicStatsEvalType
UserDefinedTemperatureCharacteristicStatsEvalType
UserDefinedTimeCharacteristicStatsEvalType
UserDefinedUnitCharacteristicStatsEvalType

CompositeSegmentStatsEvalBaseType
- ***CompositeSegmentPositionStatsEvalType***
- ***CompositeSegmentProfileStatsEvalType***
- ***CompositeSegmentPSymmetryStatsEvalType***

WeldBevelCharacteristicStatsEvalType
- ***WeldCompoundCharacteristicStatsEvalType***
- ***WeldEdgeCharacteristicStatsEvalType***
- ***WeldFilletCharacteristicStatsEvalType***
- ***WeldFlareBevelCharacteristicStatsEvalType***
- ***WeldFlareVCharacteristicStatsEvalType***
- ***WeldJCharacteristicStatsEvalType***
- ***WeldPlugCharacteristicStatsEvalType***
- ***WeldScarfCharacteristicStatsEvalType***
- ***WeldSeamCharacteristicStatsEvalType***
- ***WeldSlotCharacteristicStatsEvalType***
- ***WeldSpotCharacteristicStatsEvalType***
- ***WeldSquareCharacteristicStatsEvalType***
- ***WeldStudCharacteristicStatsEvalType***
- ***WeldSurfacingCharacteristicStatsEvalType***
- ***WeldUCharacteristicStatsEvalType***
- ***WeldVCharacteristicStatsEvalType***

Figure 61 - **CharacteristicStatsEval** types (continued from previous page)

6.12.3 **Criterion types**

The file defines three Criterion types, one of which has a derived type for each of ten unit types, as shown in Figure 62. A Criterion type defines a numerical limit outside of which an issue (e.g., a process control issue) will exist.

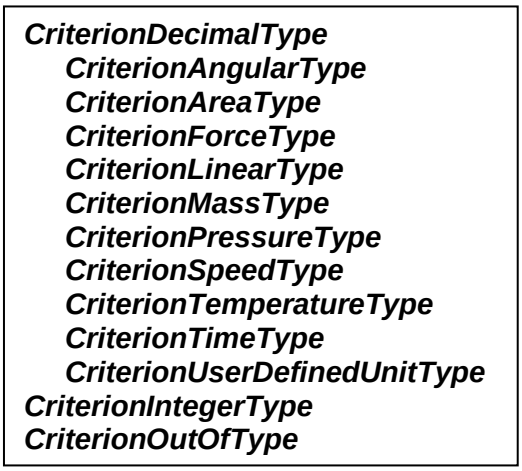


Figure 62 – Criterion Types

6.12.4 Summary Statistics

The Statistics.xsd schema file defines summary statistics types as shown in Figure 63. For each of the derived types there is a set type with “Summaries” replacing “Summary” in the name. For example, there is a ***SummariesStatisticsAngularType***.

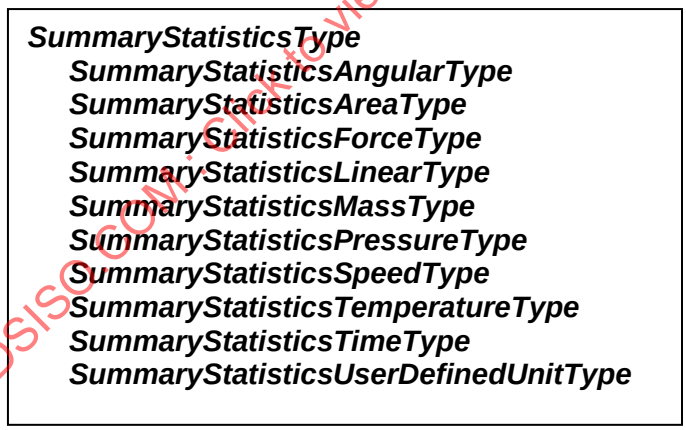


Figure 63 – Summary Statistics Types

6.12.5 Other items

The Statistics.xsd schema file defines a dozen enumeration types for statistics. For three of these, a list type is defined.

6.13 Topology.xsd

The topology.xsd schema file defines the individual topology types shown in Figure 64 and the topology set types shown in Figure 65 plus supporting enumeration types. In addition, validation types are defined for edge, face, and body. Key/keyref constraints are defined that require

references to be of the correct type. For example, a VertexId *element* in a body must be the id of a vertex. A global *element* is defined for each non-base type in Figure 64 and Figure 65.

The role of the topology types in product definition is covered thoroughly in Clause 0 and is not described here.

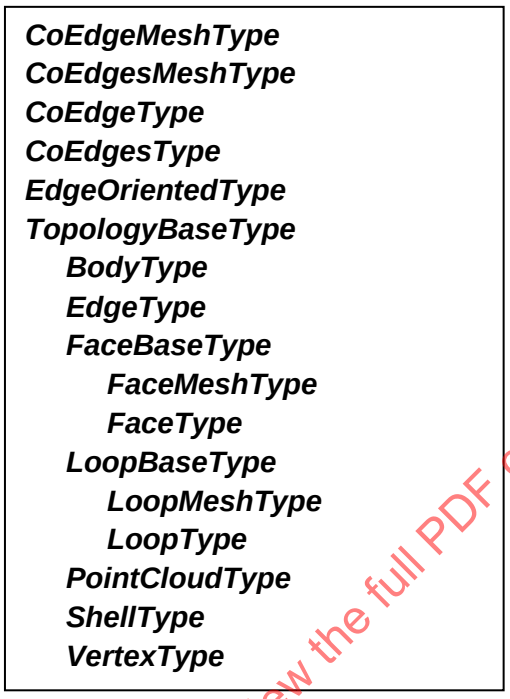


Figure 64 – Individual topology types

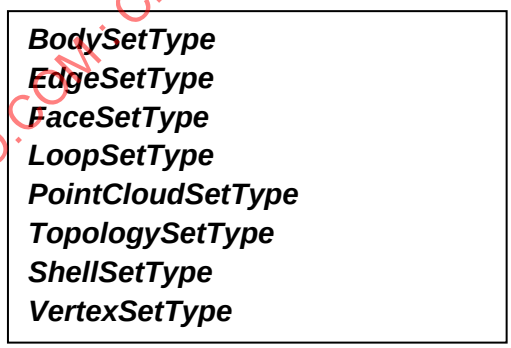


Figure 65 – Topology set types

The ***TopologySetType*** in Figure 65 consists of *element*s of the other seven types.

6.14 Traceability.xsd

The purpose of traceability information is to make it possible to associate QIF data files with the product inspected and the resources used to design it, manufacture it, program its inspection, and inspect it. The Traceability.xsd schema file defines types that describe the circumstances of a planned or completed quality measurement.

6.14.1 *InspectionTraceabilityType*

The ***InspectionTraceabilityType*** contains information applicable to an entire set of inspection results. An instance of the ***InspectionTraceabilityType*** is used (optionally) as the value of the **InspectionTraceability** *element* in both the ***ResultsType*** (multiple results sets) and the ***MeasurementResultsType*** (single results set) of the QIFResults schema. It is also used as the value of the **InspectionTraceability** *element* in the ***StatisticalStudyResultsBaseType*** (and, hence, all derived types) in the QIFStatistics schema. All *element*s of ***InspectionTraceabilityType*** are optional. The *element*s are shown in Figure 66.

Figure 66 – Elements of ***InspectionTraceabilityType***

6.14.2 *PreInspectionTraceabilityType*

The ***PreInspectionTraceabilityType*** contains information similar to that of the ***InspectionTraceabilityType***, except that is it tailored for use when inspection has not yet occurred. ***PreInspectionTraceabilityType*** is the type of the **PreInspectionTraceability** *element* of the ***QIFDocumentType*** in QIFDocument.xsd and the ***StatisticalStudyPlanBaseType*** (and, hence, all its derived types) in QIFStatistics.xsd. All *element*s of ***PreInspectionTraceabilityType*** are optional except for the **FormalStandardId**, which is required. The AsmPathIds indicate what is to be inspected. The *element*s are shown in Figure 67.

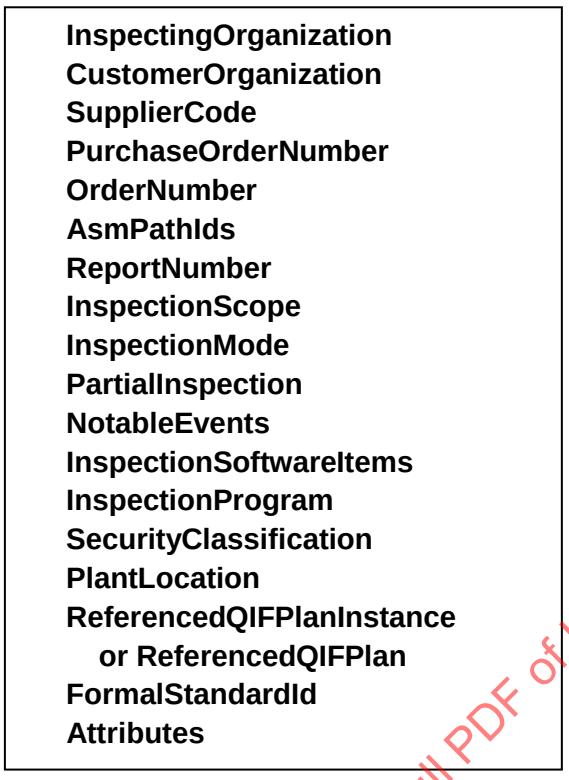


Figure 67 – Elements of ***PreInspectionTraceabilityType***

6.14.3 *ProductTraceabilityType*

The ***ProductTraceabilityType*** defines traceability information for a component (i.e., part or assembly). It is used (optionally) as the value of the **Traceability** *element* of the ***ComponentType*** in Product.xsd. The *element*s of ***ProductTraceabilityType*** are all optional. They are shown in Figure 68.

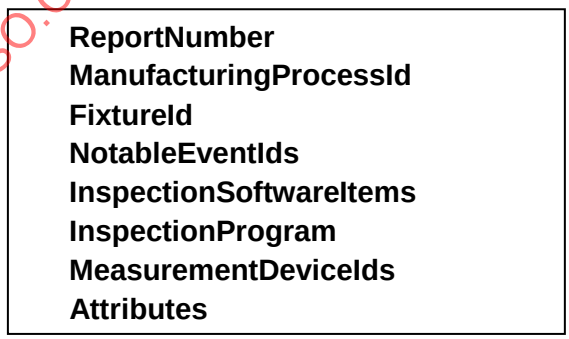


Figure 68 – Elements of ***ProductTraceabilityType***

6.14.4 *ActualProductTraceabilityType*

The ***ActualProductTraceabilityType*** defines traceability information for an actual component. It is used (optionally) as the **Traceability** *element* of the ***ActualComponentType*** in QIFResults.xsd. Its *element*s are all optional. They are shown in Figure 69.

Figure 69 – Elements of ***ActualProductTraceabilityType***

6.14.5 *ManufacturingProcessTraceabilityType*

The ***ManufacturingProcessTraceabilityType*** defines traceability information for a manufacturing process. It is used in the **ManufacturingProcessTraceabilities** *element* of the ***QIFDocumentType***. Once a **ManufacturingProcessTraceability** is listed there, it may be referenced using the **ManufacturingProcessId** *element* found at several places in the QIF model. The *element*s of ***ManufacturingProcessTraceabilityType*** are all optional. They are shown in Figure 70. There is also a required id *attribute* which may be used for referencing.

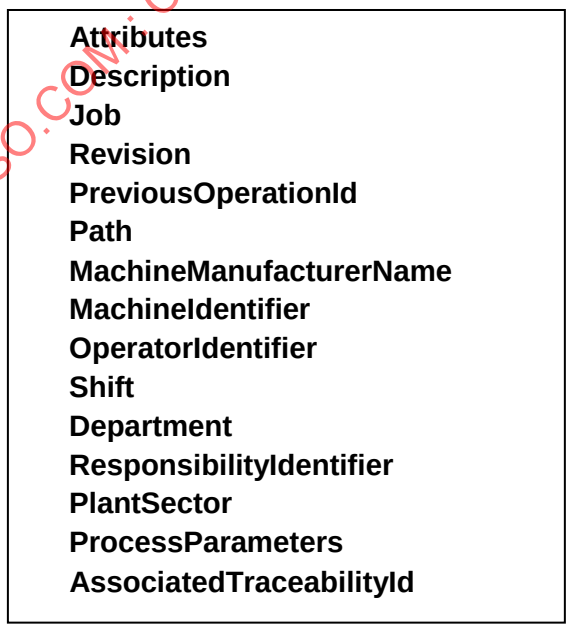


Figure 70 – Elements of ***ManufacturingProcessTraceabilityType***

Significant supporting types defined in Traceability.xsd include ***EnvironmentType***, ***InspectionProgramType***, and ***PartialInspectionType***.

6.15 Units.xsd

See also subclause 5.18 QIF Handling of Units.

6.15.1 FileUnits

The top level type defined in the file is the ***FileUnitsType***. This is used in the QIFDocument.xsd schema as the type of the **FileUnits** *element*, as shown in Figure 71. The **FileUnits** *element* gives explicit primary units for the file and optionally gives other units and user defined units that may be used in the file. The **FileUnits** *element* is optional, but it is strongly recommended that it be used. In the absence of a **FileUnits** *element*, the default units are SI units.

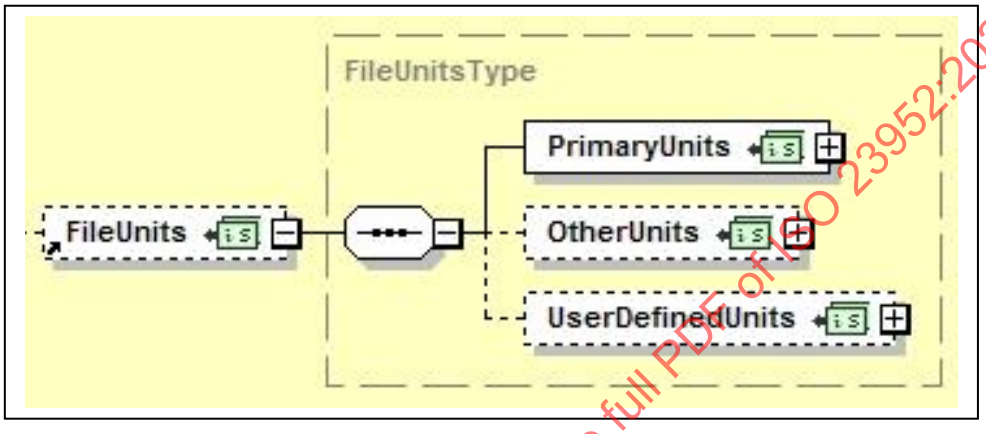


Figure 71 – **FileUnits** element

The Units.xsd schema file defines units for values of angle, area, force, length, mass, pressure, speed, temperature, and time. The length unit is called *linear* unit to avoid confusion with the length characteristic. The file also defines the following corresponding value types that have units. The default unit is shown following each value type.

AngleValueType radian
AreaValueType square meter
ForceValueType newton
LinearValueType meter
MassValueType kilogram
PressureValueType pascal
SpeedValueType meter per second
TemperatureValueType kelvin
TimeValueType second

A user defined characteristic type is defined in Characteristics.xsd for each of the defined unit types, and the **TargetValue** in each of those is the corresponding value type.

The Units.xsd file uses optional *attribute*s extensively in order that numbers appearing in instance files do not require a lot of accompanying markup.

The derivation hierarchy of values with units is shown in

Figure 72. The items shown after the type names are the *attribute*s that are added in the type. *Attribute*s that may be used with a value with any type of unit in an instance file include the name of the unit, the number of decimal places in the value, and the number of significant figures in the value. If a value represents a measured value, it may also have *attribute*s giving combined uncertainty and mean error. All of these *attribute*s are optional.

xs:decimal
- ***SpecifiedDecimalType*** – adds **decimalPlaces**, **significantFigures**
  - ***AngularValueType*** – **angularUnit**
  - ***AreaValueType*** – **areaUnit**
  - ***ForceValueType*** – **forceUnit**
  - ***LinearDualValueType*** – **linearUnit**
  - ***LinearValueType*** – **linearUnit**
  - ***MassValueType*** – **massUnit**
  - ***PressureValueType*** – **pressureUnit**
  - ***SpeedValueType*** – **speedUnit**
  - ***TemperatureValueType*** – **temperatureUnit**
  - ***TimeValueType*** – **timeUnit**
  - ***UserDefinedUnitValueType*** – name of a ***UserDefinedUnitType***
  - ***MeasuredDecimalType*** – adds **combinedUncertainty**, **meanError**
    - ***MeasuredAngularValueType*** – **angularUnit**
    - ***MeasuredAreaValueType*** – **areaUnit**
    - ***MeasuredForceValueType*** – **forceUnit**
    - ***MeasuredLinearValueType*** – **linearUnit**
    - ***MeasuredMassValueType*** – **massUnit**
    - ***MeasuredPressureValueType*** – **pressureUnit**
    - ***MeasuredSpeedValueType*** – **speedUnit**
    - ***MeasuredTemperatureValueType*** – **temperatureUnit**
    - ***MeasuredTimeValueType*** – **timeUnit**
    - ***MeasuredUserDefinedUnitValueType*** – name of a ***UserDefinedUnitType***

Figure 72 – Derivation hierarchy of values with units

The Units.xsd schema file also includes a ***UserDefinedUnitType***. The ***UserDefinedUnitValueType*** and ***MeasuredUserDefinedUnitValueType*** may be used in an instance file in a user defined unit characteristic.

6.15.2 Conversions

The ***UnitConversionType*** has *element*s **Factor** and **Offset**. These are the parameters for a conversion from non-SI units to SI units. To convert a non-SI unit value X to an SI unit value S, use the equation: S = ((X plus **Offset**) times **Factor**). Figure 73 shows values of **Factor** and **Offset** for common units. More precise values of some **Factor**s may be needed in some applications.


```
Angle conversion to radians
  degree            Factor=0.017453293  Offset=0
Area conversion to square meters
  square inch       Factor=0.00064516   Offset=0
  square foot       Factor=0.09290304   Offset=0
  square millimeter Factor=0.000001     Offset=0
Force conversion to newtons
  kilogram          Factor=9.80665      Offset=0
  ounce             Factor=0.2780139    Offset=0
  pound             Factor=4.448222     Offset=0
Length conversion to meters
  foot              Factor=0.3048       Offset=0
  inch              Factor=0.0254       Offset=0
  millimeter        Factor=0.001        Offset=0
Mass conversion to kilograms
  gram              Factor=0.001        Offset=0
  ounce             Factor=0.02834952   Offset=0
  pound             Factor=0.4535924    Offset=0
Pressure conversion to pascals
  kilopascal        Factor=1000.0       Offset=0
  psi               Factor=6894.757     Offset=0
Speed conversion to meters per second
  feetPerSecond     Factor=0.3048       Offset=0
  inchesPerSecond   Factor=0.0254       Offset=0
  mmPerSecond       Factor=0.001        Offset=0
Temperature Conversion to kelvin
  Fahrenheit        Factor=0.555555556  Offset=459.67
  Celsius           Factor=1.0          Offset=273.15
  Rankine           Factor=0.555555556  Offset=0
Time conversion to seconds
  hour              Factor=3600.0       Offset=0
  minute            Factor=  60.0       Offset=0
```

Figure 73 – Conversion of units

6.15.3 **FileUnitsExample**

Figure 74 shows a snippet of an instance file containing a **FileUnits** *element*. The **SIUnitName** and the **UnitConversion** *element*s are optional. In the snippet, the **AngleUnit** uses them but the **LinearUnit** does not. The snippet shows one entry in the **OtherUnits** and one entry in the **UserDefinedUnits**.

```
<FileUnits>
  <PrimaryUnits>
    <AngularUnit>
      <SIUnitName>radian</SIUnitName>
      <UnitName>degree</UnitName>
      <UnitConversion>
        <Factor>0.017453293</Factor>
      </UnitConversion>
    </AngularUnit>
    <LinearUnit>
      <UnitName>inch</UnitName>
    </LinearUnit>
    <TemperatureUnit>
      <SIUnitName>kelvin</SIUnitName>
      <UnitName>Fahrenheit</UnitName>
      <UnitConversion>
        <Factor>0.555555556</Factor>
        <Offset>459.67</Offset>
      </UnitConversion>
    </TemperatureUnit>
  </PrimaryUnits>
  <OtherUnits>
    <LinearUnit>
      <UnitName>mm</UnitName>
    </LinearUnit>
  </OtherUnits>
  <UserDefinedUnits>
    <UserDefinedUnit>
      <WhatIsMeasured>electrical resistance</WhatIsMeasured>
      <UnitName>ohm</UnitName>
    </UserDefinedUnit>
  </UserDefinedUnits>
</FileUnits>
```

Figure 74 – **FileUnits** snippet

6.15.4 **Instance File Example Using Units**

Three snippets from an instance file are shown in Figure 75. Assuming that the **FileUnits** shown in Figure 74 is in the instance file, the snippets are all valid.

The first cylinder has a diameter of 19 mm; the **linearUnit** mm is given explicitly. That is valid because mm is given as one of the **OtherUnits**. The **decimalPlaces** value of 1 is valid even though no decimal places appear in “19”.

The second cylinder has a diameter of 2.53 inches. The diameter has no explicit **linearUnit**, but the **linearUnit** for the diameter is implicitly inch because inch was given as the **LinearUnit** in the **PrimaryUnits** of the **FileUnits**.

The nominal angularity characteristic has an angle of 10.00145 degrees, although only the first four figures are significant. The angle has no explicit **angularUnit**, but the **angularUnit** for the Angle is implicitly degree because degree was given as the **AngularUnit** in the **PrimaryUnits** of the **FileUnits**.


```
<CylinderFeatureDefinition id="62">
  <Diameter linearUnit="mm" decimalPlaces="1">19</Diameter>
</CylinderFeatureDefinition>
  ...
<CircleFeatureDefinition id="72">
  <Diameter>2.53</Diameter>
</CircleFeatureDefinition>
...
<AngularityCharacteristicNominal id="45">
  <CharacteristicDefinitionId>42</CharacteristicDefinitionId>
  <Angle significantFigures="4">10.00145</Angle>
</AngularityCharacteristicNominal>
```

Figure 75 – Instance file snippets using units

6.16 Visualization.xsd

The Visualization.xsd schema file defines types needed for visualizing products, including PMI items. The file also includes data specifying views of a product.

Most of the information required for PMI visualization is in the PMI item being displayed. The way it should be visualized is prescribed by ASME GD&T and ISO GPS related standards, and American Welding Society standards. However the placement of the PMI item on a 2D drawing or in a 3D model based design is controlled by the user, as are the lines connecting the visualized PMI with the visualized feature. Additional text may also accompany a PMI item and may use various fonts. These items are what the Visualization.xsd schema file defines. Any QIF application providing PMI visualization should be capable of displaying PMI from this visualization information.

Visualization and views are described in much more detail in subclause 6.16.

The top level type for displaying PMI items is the ***VisualizationSetType***, which is the type of the **VisualizationSet** *element* of the ***ProductType*** as shown in Figure 76. The ***VisualizationSetType*** has (1) a **Fonts** *element* that is a list of **Font** *element*s and (2) a **PMIDisplaySet** *element* that is a list of **PMIDisplay** *elements*.

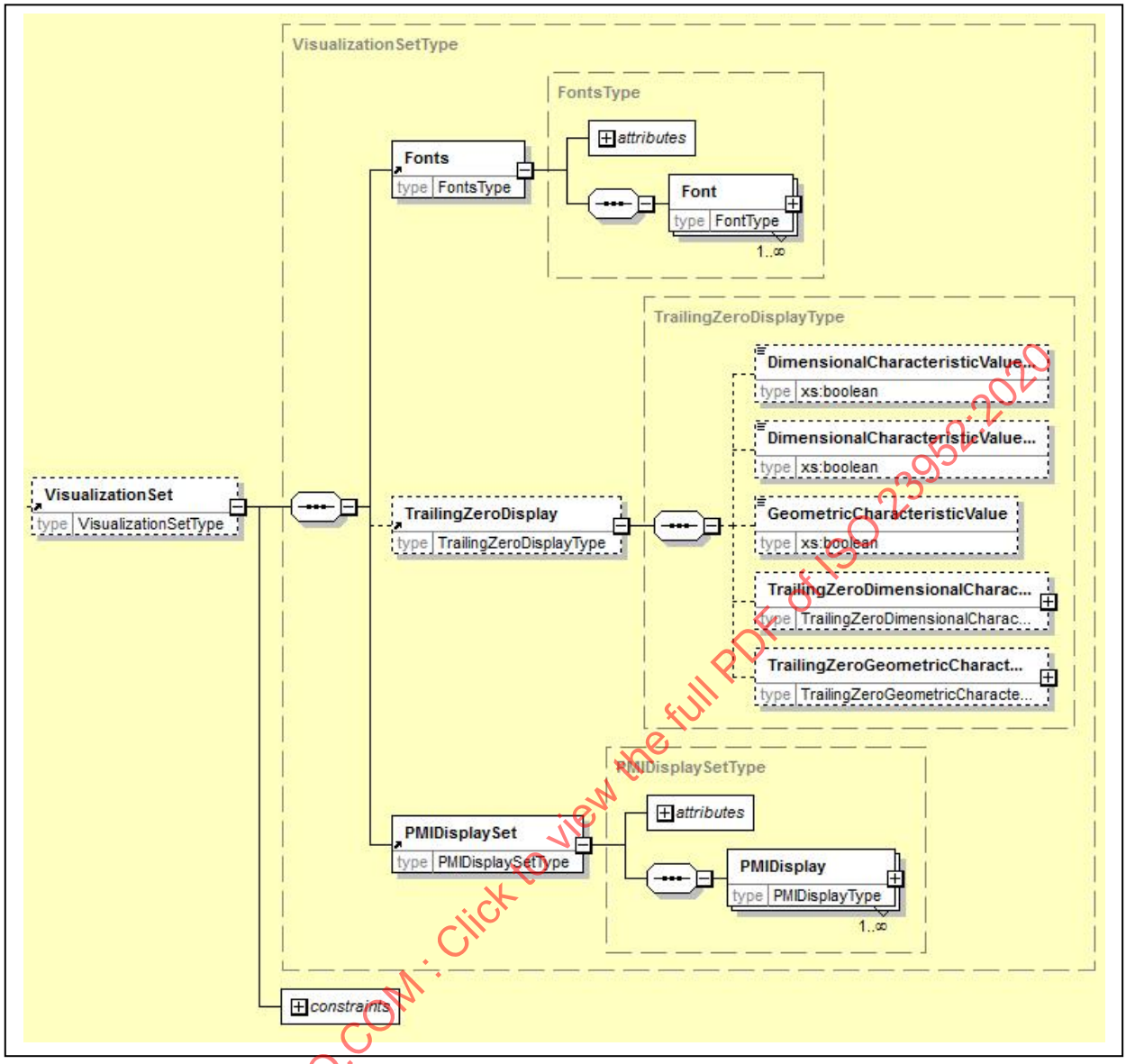


Figure 76 – **VisualizationSet** *element*

The **Font** *element*s are of ***FontType***. A ***FontType*** is described by **Name** and **Size** *element*s as well as an optional **Attributes** *element*. It also has *attribute*s indicating whether the font is bold and/or italic and/or underlined. Another *attribute* gives an index number for the font.

The **PMIDisplay** *element*s are of ***PMIDisplayType***. The *element*s of the ***PMIDisplayType*** include a required **Reference** *element* that is the id of a PMI item to be displayed and the following optional *element*s:

- **Attributes** (user data)
- **Color**
- **Plane**
- **Texts**
- **Leader** (or one of 5 substitutes that are specialized types of leader)
- **WitnessLines**
- **Frames**
- **Graphics**

The **Frames** *element* is a list of **Frame** *element*s of type ***FrameType***. The *substitutionGroup* of the **Frame** *element* allows any of the following *element*s to appear in the list.

- **FrameCircular**
- **FrameFlag**
- **FrameHexagonal**
- **FrameIrregularForm**
- **FrameOoctagonal**
- **FramePentagonal**
- **FrameRectangular**
- **FrameTriangular**
- **FrameWeldSymbol**

The top level type for specifying views is the ***ViewSetType***, which is the type of the **ViewSet** *element* of the ***ProductType*** as shown in Figure 77.

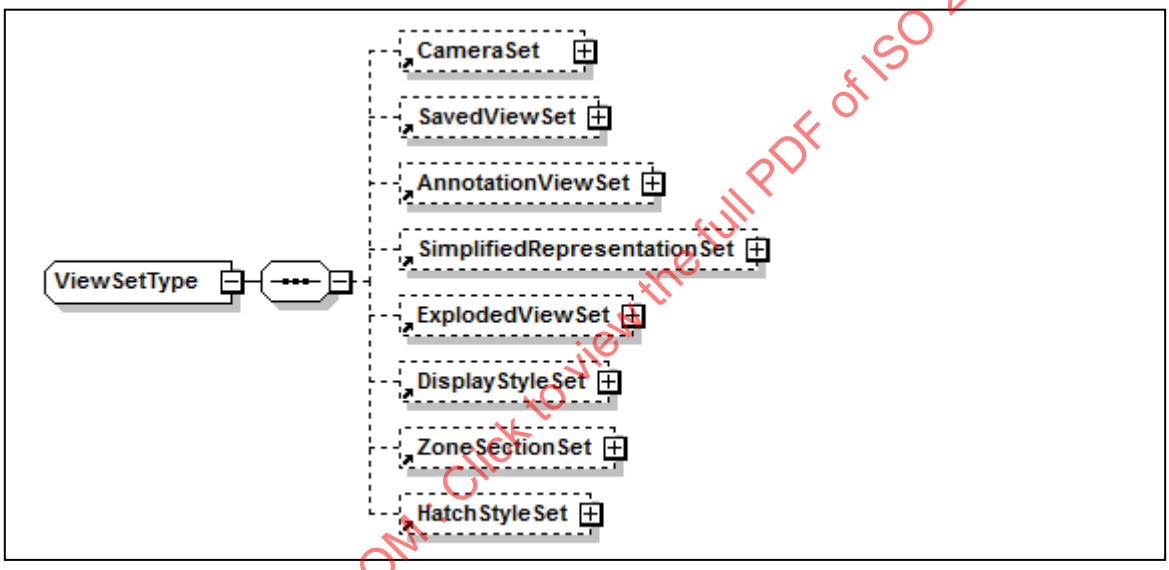


Figure 77 – ***ViewSetType***

7 QIF Model Based Definition (MBD) information model

7.1 Foreword

The Quality Information Framework (QIF) information model was developed by domain experts from the manufacturing quality (that is metrology) community representing a wide variety of industries and quality measurement needs. Specifically for the QIF Model-Based Definition (QIF MBD) work, contributors include:

- Capvidia
- Mitutoyo America
- National Institute of Standards and Technology
- Origin International Inc.

The bulk of the work on this clause was performed by the Complete and Accurate Model-Based Definition (CAMBD) Working Group, approved and revised as needed by the Quality Information Framework (QIF) Working Group, and given final approval for ANSI review by the DMSC's Quality Management Systems (QMS) Committee.

7.2 Introduction

The QIF MBD information model defines an open, 3D model-based product definition (MBD). The model is independent of any CAD system. It includes a full suite of formally modeled characteristics (dimensions, tolerances, etc.) and features, associated to a complete model of parts and assemblies (topology, views, etc.) and metadata. It supports interchange of part and assembly models between CAD systems and transfer of those models to downstream processes.

A QIF MBD instance file can replace a traditional 2D drawing or be used together with a drawing.

QIF MBD defines a digital data format to convey part geometry (typically called the "CAD model") and information to be consumed by downstream manufacturing quality processes, about the essential characteristics of the product. Information defined in a QIF MBD model is more useful than is possible on a 2D drawing. For example, each product manufacturing information (PMI) object can have associated geometry, thereby conveying information to downstream processes. PMI objects include, but are not limited to: dimensions, geometric tolerances, datums, and datum reference frames.

The QIF MBD data format defines manufacturing information to form a "digital bridge" connecting CAD processes for product design and downstream manufacturing quality processes that generate physical products. The QIF MBD data model provides a vendor-neutral format and ensures that software solutions that support it, by either writing or reading QIF instance files, can exchange manufacturing quality data efficiently and accurately.

7.3 Scope

7.3.1 Contents of this clause

This standard defines the QIF MBD data model for a portion of the QIF manufacturing quality information model focused on model-based definition and designated as QIF MBD. QIF MBD defines a digital data format to convey part geometry (typically called the "CAD model") and information to be consumed by downstream manufacturing quality processes. The QIF MBD model

consists of definitions for data types, elements, the logical relationships between them, and the semantics of the quality information. The QIF MBD model is defined using the XML Schema Definition Language (XSDL). The QIF MBD model is a digital data exchange mechanism that can be easily incorporated in software developed by commercial solution vendors that implements manufacturing quality systems.

The top level of the QIF MBD model is defined in the QIFApplications/QIFDocument.xsd XML schema file. That file includes, directly or indirectly, all of the 15 XML schema files in the QIF Library (the QIFLibrary directory). An XML instance file describing a model-based definition with PMI will contain information items drawn from QIFApplications/QIFDocument.xsd and many other QIF schema files, particularly QIFApplications/QIFProduct.xsd, QIFLibrary/Characteristics.xsd, and QIFLibrary/Features.xsd. Clause 7 describes primarily the CAD model and assembly model embodied in QIFApplications/QIFProduct.xsd, QIFLibrary/Topology.xsd, and QIFLibrary/Geometry.xsd.

XSDL also supports the definition of rules and validation checks of QIF instance files. Almost all of these check that a connection made by reference to an object that is supposed to be of a particular type does, in fact, reference an object of the correct type. Hundreds of these are included in the overall QIF model, and most of them are applicable to QIF MBD instance files. This clause describes these only briefly.

7.3.2 QIF MBD Information Model Application Architecture

The QIF MBD model is designed to support transfer of model-based definition information between a variety of types of applications as shown in Figure 78. Systems that can generate QIF MBD files are shown on the left, and systems that can consume QIF MBD files on the right.

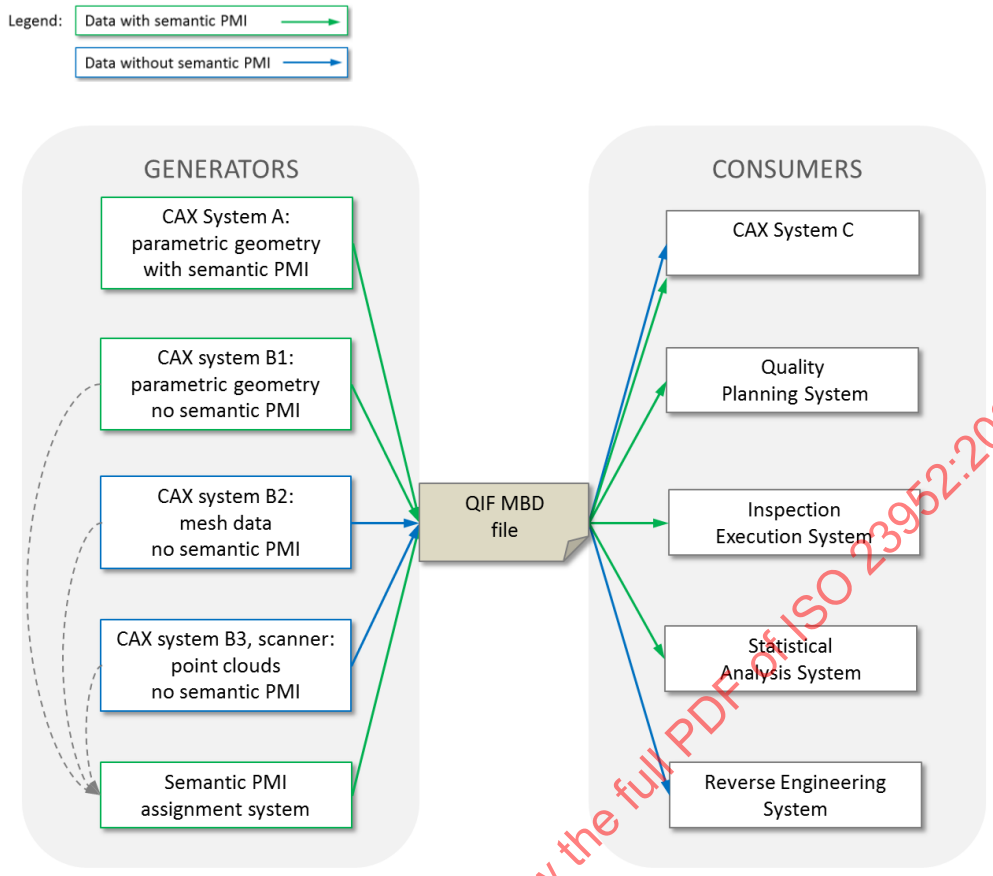


Figure 78 – Workflow of QIF MBD Information

A CAX system such as System A in Figure 78 that has semantic PMI and is QIF-enabled will be able to output a QIF MBD file with semantic PMI.

CAX systems such as Systems B1/B2/B3 in the Figure 78 that do not have semantic PMI and are QIF-enabled will be able to output a QIF MBD file containing a product definition without PMI. Although QIF MBD is designed to include PMI, it is not required.

Output from Systems B1/B2/B3 may be fed into a QIF-enabled semantic PMI assignment system, which would be able to write a QIF MBD file containing the product definition with semantic PMI.

Once a QIF MBD file is generated, by whatever means, it may be fed into any QIF-enabled system.

QIF enabled CAX System C can read QIF MBD files. If System C fully supports semantic PMI, then it can consume all the information from a QIF MBD file. If not, it can only consume product definition without PMI from QIF MBD files. Thus, QIF MBD enables CAD-to-CAD transfers.

A QIF-enabled Quality Planning System can read QIF MBD files. If it does high level planning (the plan specifies only what to inspect), it may only need the PMI stored in a file. If it does low level planning (the plan specifies what and how to inspect) it may need all the information in the file.

Other QIF-enabled quality systems, such as inspection execution systems and statistical analysis systems, will be able to consume a QIF MBD file, produce graphical displays for the user, and use the PMI for calculations.

QIF MBD was designed for the uses described above. QIF MBD files may have additional uses, which are not defined here.

7.4 QIF MBD information model requirements

All CAD systems generate models of the nominal shape of products. Designers and/or manufacturing quality experts add information to the nominal shape information to instruct downstream manufacturing quality functions how to perform actions. This combination of information is called Product Definition with Product Manufacturing Information (PDPMI).

The basic requirement of QIF MBD is that it must be able to represent PDPMI. The QIF MBD model must be able to capture the product definition information coming from a CAX authoring system. The QIF MBD model must also be able to capture the PMI that comes from the CAX authoring system.

A second requirement is that the QIF MBD model must represent PMI in a form that can be consumed by downstream application software. That is, the representation must be able to graphically present the PMI. The representation must also enable processing the PMI without graphical presentation of it. Being able to display the PMI is essential for human use. Being able to use the PMI without graphical presentation is essential for computer use.

Many products are assemblies of parts. A third requirement of the QIF MBD model is that it be able to represent assemblies.

The representation of PMI used in QIF MBD is described in Clause 5, Clasue 6, and Clause 7 of the standard as well as in the XML schema files containing the formal model. This Clause of the standard focuses on the representation of the definition of the nominal shape of a product and the representation of assemblies.

7.5 Overview of the product data model

7.5.1 Design principles

QIF defines a complete and unambiguous model structure implementing a clear and efficient way for representing simple parts and complex assemblies.

QIF product definition implements Boundary Representation (BREP) — a method for defining shapes using their boundaries. A distinctive characteristic of Boundary Representation is the clear separation of the shape definition data in the following two types: topology and geometry. Topology defines the relationship between the geometric entities that can be linked together to produce complex shapes. In contrast to the geometry, topology does not consider the metric properties of objects. Geometry defines the shape of the model entities, independent of their topological relationships.

The QIF data model supports two modeling concepts – exact modeling (with use of one global model tolerance) and tolerant modeling (the model tolerance can vary from one entity to another). In both cases, the tolerance concept applies to the numerical accuracy of numbers in the file. This is distinct from the tolerance concept in PMI, which applies to manufacturing tolerances.

QIF geometric objects can be categorized in different ways. The most natural way is to separate objects by dimension: zero-dimensional (points), one-dimensional (curves) and two-dimensional (surfaces) objects.

In order to avoid any potential interoperability issues, the QIF data model defines all geometric entities explicitly. No entities such as blends, which can be interpreted and implemented differently by different systems, are allowed.

QIF supports a hybrid model representation allowing data of different natures within one model. This includes items such as parametric, mesh, and point cloud data.

For any model entity, a variety of traditional CAD attributes can be specified (such as: color, label, layer, visibility status, etc.) with possible arbitrary extensions implemented as user-defined attributes.

7.5.1.1 Binary data blocks

To store binary data blocks in an xml file the Base-64 encoding is used; it is implemented as one of the standard xml types, xs:base64Binary.

In QIF there are two application areas where the binary data can be used.

The first use is one type of user-defined attributes (AttributeUser) - BinaryDataType.

The second use of binary data is for providing an alternative, more efficient and compact way of storing large amounts of simple-type data such as 3D points. This noticeably reduces file size and significantly increases file parsing speed. These two factors can be crucial in working with such objects as point clouds or tessellated data.

The BinaryDataType represents a Base64-encoded arbitrary binary data block. For base64Binary data, the entire binary stream is encoded using the Base64 Alphabet in RFC 2045.

Fields of BinaryDataType:

Field Name	Data Type	Description
<body>	xs:base64Binary	The binary data encoded in the Base-64 format.
@count	xs:unsignedInt	The count attribute shows the size of the binary block in bytes.

Example:

```
<Edge id="459">
  <Attributes count="3">
    <!-- A user defined attribute STD02 - 14 bytes of binary data -->
    <AttributeUser name="STD02" nameUserAttribute="T1">
      <UserDataBinary n="14">
        c2FtcGxlIHN0cmluZwA=
      </UserDataBinary>
    </AttributeUser>
  </Attributes>
```


. . .
</Edge>

The fields where the binary alternatives can be used are shown in the table below. The corresponding non-binary field names are the same but without the "Binary" suffix.

Field Name (binary array)	Array Element Type
Nurbs12CoreType/CPsBinary	2D point (pair of doubles)
Nurbs13CoreType/CPsBinary	3D point (triplet of doubles)
Nurbs23CoreType/CPsBinary	3D point (triplet of doubles)
Polyline12CoreType/PointsBinary	2D point (pair of doubles)
Polyline13CoreType/PointsBinary	3D point (triplet of doubles)
PathTriangulationCoreType/EdgesBinary	Pair of integers
MeshTriangleCoreType/TrianglesBinary	Triplet of integers
MeshTriangleCoreType/NeighboursBinary	Triplet of integers
MeshTriangleCoreType/VerticesBinary	3D point (triplet of doubles)
MeshTriangleCoreType/NormalsBinary	3D unit vector (triplet of doubles)
MeshTriangleType/NormalsSpecialBinary	Unsigned integer, unsigned byte and 3D point (triplet of doubles)
FaceMeshType/TrianglesBinary	Unsigned integer
FaceMeshType/TrianglesVisibleBinary	Unsigned integer
FaceMeshType/TrianglesHiddenBinary	Unsigned integer
FaceMeshType/TrianglesColorBinary	Triplet of unsigned bytes
PointCloudType/PointsBinary	3D point (triplet of doubles)
PointCloudType/NormalsBinary	3D unit vector (triplet of doubles)
PointCloudType/PointsVisibleBinary	Unsigned integer
PointCloudType/PointsHiddenBinary	Unsigned integer
PointCloudType/PointsColorBinary	Triplet of unsigned bytes
ValidationPointsType/PointsBinary	3D point (triplet of doubles)
ValidationPointsType/DirectionsBinary	3D unit vector (triplet of doubles)
Polyline2dType/PointsBinary	2D point (pair of doubles)
Triangulation2dType/VerticesBinary	2D point (pair of doubles)
Triangulation2dType/TrianglesBinary	Triplet of integers

Table 4 – Binary Arrays

The binary representation:

Array Element Type	Binary Representation	Bytes	Endianness
double	IEEE 754-2008 binary64 double precision floating point format.	8	Little-endian
integer	32-bit integer	4	Little-endian
unsigned integer	32-bit unsigned integer	4	Little-endian
unsigned byte	8-bit unsigned integer	1	Little-endian

Table 5 – Binary Representation

The ArrayBinaryType represents an array of Base64-encoded binary elements. For data the entire binary stream is encoded using the Base64 Alphabet in RFC 2045.

Fields of ArrayBinaryType:

Field Name	Data Type	Description
<body>	xs:base64Binary	Binary data encoded in Base64 format.
@count	xs:unsignedInt	The count attribute shows how many elements are present in this array.
@sizeElement	xs:unsignedInt	The sizeElement attribute shows the size (in bytes) of one element stored in the array. The total size of the binary array can be calculated as: n*sizeElement.

Example:

```
<!-- Binary -->
<PointCloud id="3">
  <PointsBinary count="31" sizeElement="24">
    ANwUZdaTPcD9///9B/QzwCx7CR2wKypAZHlAwox0PcAFAADwH+gzwAzaS+jAVipAwMzMQlZV
    PcAFAADKR9wzwBsAABAzgSpAsZfQ3jI2PcAJAACAf9AzwJvQXkIHqypAfJtijiIXPcD+//8F
    x8QzwHkvoS0+1CpAo5mZSSX4PMAMAABQHrkzwPf//3/Y/CpAalOMCDvZPMAEAABSha0zwAEm
    tOfWJCtAbYpRw2O6PMAOAAAA/KEzwKOE9hI6TCtA9P//cZ+bPMD7//9NgpYzwP7//68CcytA
    mXWuDO58PMDy//8vGIszwFh7CW0xmStAiKxzi09ePMD+//+ZvX8zwCvaS/jGvitAWmZm5sM/
    PMAMAACAcnQzwPL////D4ytAdGSdFUshPMAMAADWNmkzwJnQXjIpCCxAVGgvEeUCPMAIAACQ
    Cl4zwGMvoT33KyxAJTMz0ZHkO8AKAACi7VIzwAMAANAuTyxAn4a/TVHGO8AAAAAA4EczwPUl
    tJfQcSxACCYrfiOoO8D2//+d4TwzwLSE9kLdkyxAw8zMXAiKO8D1//9v8jEzwPT//39VtSxA
    a0J73/9pO8AIAABqEiczwC97Cf051ixALEYN/wlOO8ACAACAQRwzwBnaS2iL9ixAoJmZsyYw
    O8AIAACmfxEzwPX//29KFi1ADv429VUSO8AKAADQzAYzwK/QXsJ3NS1AEjX8u5f0OsD2///x
    KPwywFEvoQ0UVC1ADgAAAOzWOsAOAAAAlPEywAAAAAAgci1ASSBZuVK5OsAJAADuDecywP4l
    tEecjy1AQ1ce4MubOsANAACwltwywMeE9pKJrC1AXmZmbFd+OsAIAAA6LtIywA8AAJDoyC1A
    NQ9IVvVgOsAFAACA1McywE97Ce255C1A8xLalaVDOsDz//91ib0ywAXaS1j+/y1ANDMzI2gm
    OsD4//8PTbMywBoAAIC2Gi5AWzFq9jwJOsACAABCH6kywJvQXhLjNC5A
  </PointsBinary>
</PointCloud>

<!-- Text -->
<PointCloud id="3">
  <Points count="31">
    -29.5774901557852 -19.9532469511032 13.0853280138086
    -29.4552728087814 -19.9067373275757 13.1694405167191
    -29.3333474874496 -19.8604704141617 13.2523427009583
    -29.211713720251 -19.8144454956055 13.3340397587529
```



```
    -29.0903710356465 -19.7686618566513 13.4145368823299
    -28.9693189620972 -19.7231187820435 13.493839263916
    -28.848557028064 -19.6778155565262 13.5719520957382
    -28.7280847620081 -19.6327514648438 13.6488805700231
    -28.6079016923904 -19.5879257917404 13.7246298789978
    -28.4880073476721 -19.5433378219604 13.799205214889
    -28.368401256314 -19.4989868402481 13.8726117699235
    -28.2490829467773 -19.4548721313477 13.9448547363281
    -28.130051947523 -19.4109929800034 14.0159393063298
    -28.0113077870122 -19.3673486709595 14.0858706721553
    -27.8928499937057 -19.3239384889603 14.1546540260315
    -27.7746780960648 -19.28076171875 14.2222945601852
    -27.6567916225504 -19.2378176450729 14.2887974668432
    -27.5391901016235 -19.1951055526733 14.3541679382324
    -27.4218730617453 -19.1526247262955 14.4184111665796
    -27.3048400313766 -19.1103744506836 14.4815323441117
    -27.1880905389786 -19.068354010582 14.5435366630554
    -27.0716241130122 -19.0265626907349 14.6044293156377
    -26.9554402819386 -18.9849997758865 14.6642154940852
    -26.8395385742188 -18.9436645507813 14.722900390625
    -26.7239185183137 -18.9025563001633 14.7804891974838
    -26.6085796426844 -18.8616743087769 14.8369871068884
    -26.4935214757919 -18.8210178613663 14.8923993110657
    -26.3787435460974 -18.7805862426758 14.946731002245
    -26.2642453820617 -18.7403787374496 14.9999873726456
    -26.150026512146 -18.7003946304321 15.052173614502
    -26.0360864648113 -18.6606332063675 15.1032949200383
  </Points>
</PointCloud>
```

7.5.1.2 Entity attributes

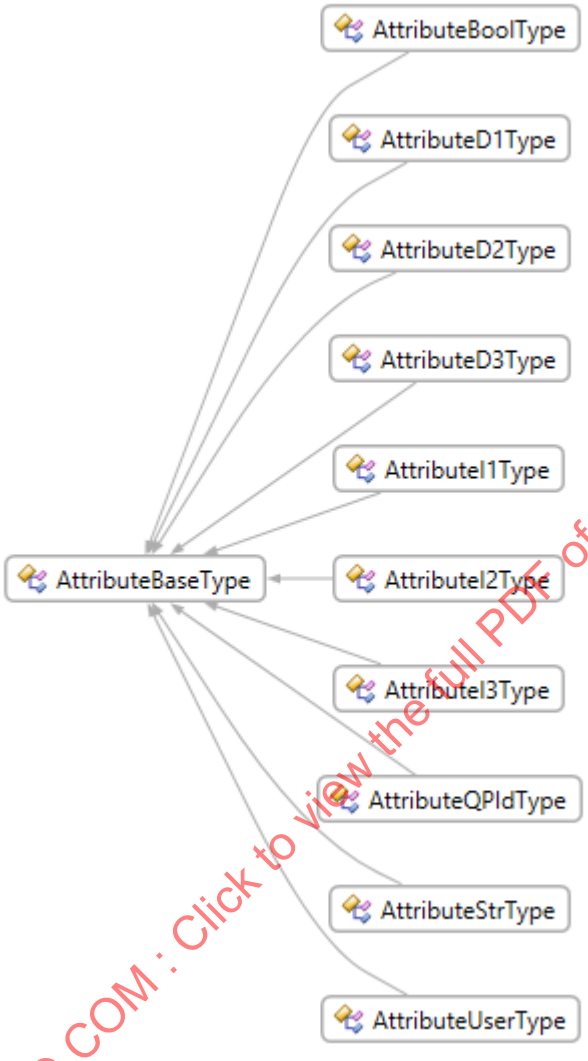


Figure 79 – Entity attributes

All entity attributes contain the following fields:

Field Name	Data Type	Description
@name	xs:string	The name of the entity attribute. This name is a unique identifier of an attribute within the entity.

7.5.1.2.1 *Boolean entity attribute*

The AttributeBool describes an entity attribute of Boolean type.

Fields:

Field Name	Data Type	Description

Value	xs:boolean	The Boolean value.

Example:

```
<AttributeBool name="bool_attr" value="1"/>
```

7.5.1.2.2 *Double entity attribute*

The AttributeD1 describes an entity attribute of double type.

Fields:

Field Name	Data Type	Description
@value	xs:double	The double value.

Example:

```
<AttributeD1 name="D1_attr" value="-12.230243"/>
```

7.5.1.2.3 *Pair of doubles entity attribute*

The AttributeD2 describes an entity attribute of ‘pair of doubles’ type.

Fields:

Field Name	Data Type	Description
@value	D2Type	The pair of double values.

Example:

```
<AttributeD2 name="D2_attr" value="223.13 -34.244"/>
```

7.5.1.2.4 *Triplet of doubles entity attribute*

The AttributeD3 describes an entity attribute of ‘triplet of doubles’ type.

Fields:

Field Name	Data Type	Description
@value	D3Type	The triplet of double values.

Example:

```
<AttributeD3 name="D3_attr" value="23.434 -0.927 37.321"/>
```

7.5.1.2.5 *Integer entity attribute*

The AttributeI1 describes an entity attribute of integer type.

Fields:

Field Name	Data Type	Description
@value	xs:integer	The integer value.

Example:

```
<AttributeI1 name="int_attr" value="1987"/>
```

7.5.1.2.6 *Pair of integers entity attribute*

The AttributeI2 describes an entity attribute of 'pair of integers' type.

Fields:

Field Name	Data Type	Description
@value	I2Type	The pair of integer values.

Example:

```
<AttributeI2 name="I2_attr" value="223 -34"/>
```

7.5.1.2.7 *Triplet of integers entity attribute*

The AttributeI3 describes an entity attribute of 'triplet of integers' type.

Fields:

Field Name	Data Type	Description
@value	I3Type	The triplet of integer values.

Example:

```
<AttributeI3 name="I3_attr" value="323 -92 373"/>
```

7.5.1.2.8 *QPId entity attribute*

The AttributeQPid describes an entity attribute of QPId type.

Fields:

Field Name	Data Type	Description
Value	QPIdType	The QPId value.

Example:

```
<AttributeQPId name="qpid_attr">
  <Value>4b61ec75-ab3c-4129-b88a-d9e2ee415322</Value>
```



```
</AttributeQPId>
```

7.5.1.2.9 *String entity attribute*

The AttributeStr describes an entity attribute of string type.

Fields:

Field Name	Data Type	Description
@value	xs:string	The string value.

Example:

```
<AttributeStr name="str_attr" value="Value of string attribute"/>
```

7.5.1.2.10 *Custom entity attribute*

The AttributeUser defines a user-defined entity attribute that contains a binary array or any user-defined structured XML data.

Fields:

Field Name	Data Type	Description
@nameUserAttribute	xs:string	The name of user-defined attribute type. The structure of the user-defined attribute can be identified by this name.
UserDataXML or UserDataBinary	xs:any or BinaryDataType	UserDataXML element: each element is a user-defined XML element from any namespace that is not the target namespace. The XML processor will validate elements if the corresponding schema is present, if the schema is not present, no errors will occur. or UserDataBinary element: a binary block of user data.

Example 1: attribute with user defined XML content:

```
<AttributeUser name="user_attr" nameUserAttribute="user_type_structure">
  <UserDataXML>
    <Structure xmlns="http://userschema.org">
      <TextField>Some text</TextField>
      <IntegerField>2332</IntegerField>
    </Structure>
  </UserDataXML>
</AttributeUser>
```

Example 1: user defined XML schema:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
```

```
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://userschema.org"
  targetNamespace="http://userschema.org"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  version="1">

  <xs:complexType name="StructureType">
    <xs:sequence>
      <xs:element name="TextField" type="xs:string">
      </xs:element>
      <xs:element name="IntegerField" type="xs:integer">
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Structure" type="StructureType"/>

</xs:schema>
```

Example2: attribute with binary content:

```
<AttributeUser name="user_attr" nameUserAttribute="user_type_image">
  <UserDataBinary n="240">
    iVBORw0KGgoAAAANSUhEUgAAACAAAAAUCAIAAABj86gYAAAABmJLR0QAAAAAAAD5Q7t/AAAA
    CXBIWXMAAA7EAAAOxAGVKw4bAAAAkElEQVR4nO2RyxXAIAwFqYd6qId6qId6PChMIDFBPqPx
    xJ4QHlkTzKGMGQXesgVDfhIEZyrWx+d49LYfEFQBXFuvQME6FS5ohqZZFaR2ABfYaV54R8ZK
    5lzSDQEZUboASdylgnKMwV4Hwp6K4p9cifxx74A3PTWigrqAj4gN4QMBfeR+LVzDrMUjK7IF
    Q9QFJ99Bg+O37n7tAAAAAElFTkSuQmCC
  </UserDataBinary>
</AttributeUser>
```

7.5.2 **Geometry**

QIF geometry defines the shape of the model entities and includes the subtypes shown in Figure 80.

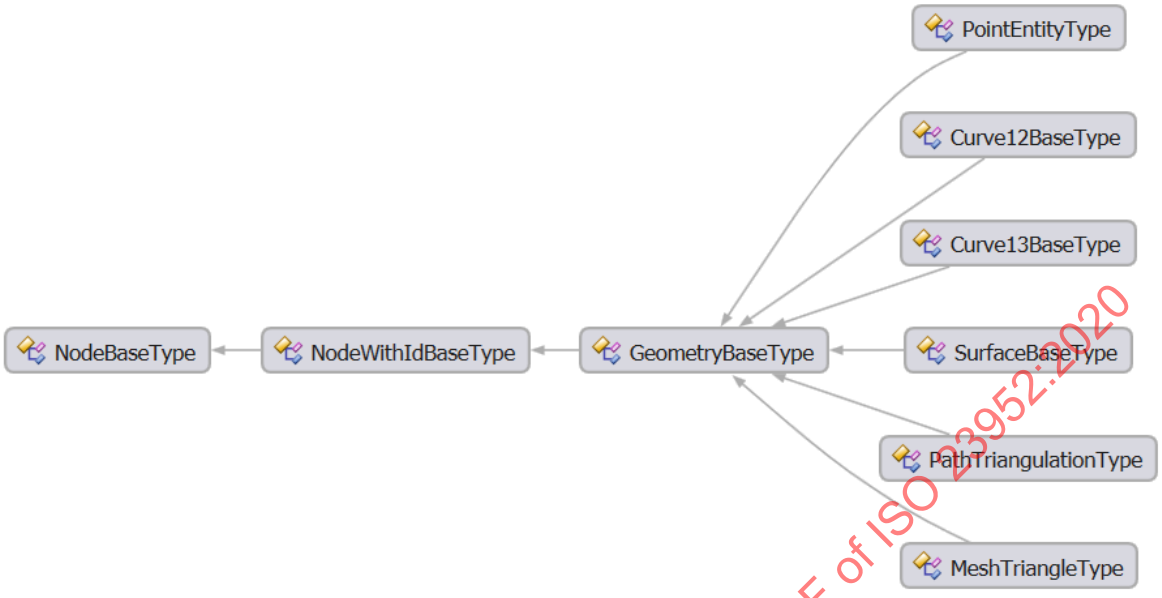


Figure 80 – Geometry Types

All geometric entities contain the following fields:

Field Name	Data Type	Description
@id	QIFIdType	The unique model entity identifier.
@label	xs:string	The model entity "nameplate". Normally it can be seen at the entity item in the model tree.
Attributes	AttributesType	User defined attributes (typified, binary array, or XML structured).

7.5.2.1 Point

The Point describes a point in 3D space. It is normally used as underlying geometry for vertices.

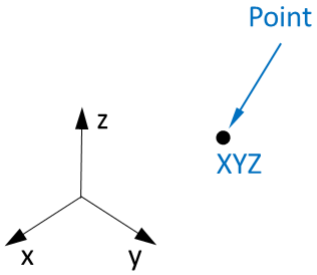


Figure 81 – **Point**

Fields:

Field Name	Data Type	Description
XYZ	PointSimpleType	The Cartesian three-dimensional coordinates of the 3D point.

Example:

```
<Point id="200">
  <XYZ>-0.2032 -0.0666 -0.0684</XYZ>
</Point>
```


7.5.2.2 2D Curves

This chapter describes curves defined in 2D space, curve(t):R1 → R2, where

- R1 is curve parameter space - the 1D real number space, where each point is defined as a single coordinate (t)
- R2 is the two-dimensional Euclidean space, where each point is defined as a coordinate pair (u, v).

The correspondence between 1D parameter space and 2D Euclidean space is illustrated in Figure 82.

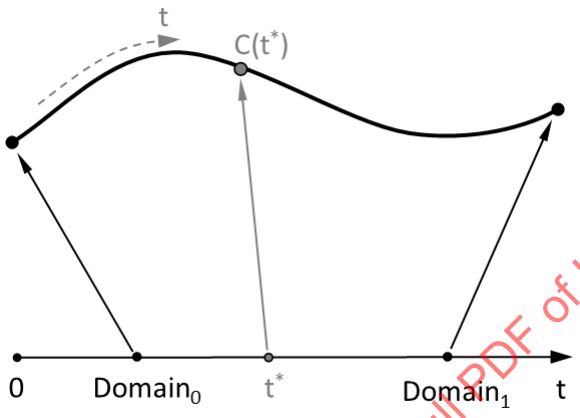


Figure 82 – 2D Parametric Curve

All types of 2D curves are derived from Curve12BaseType as shown in Figure 83.

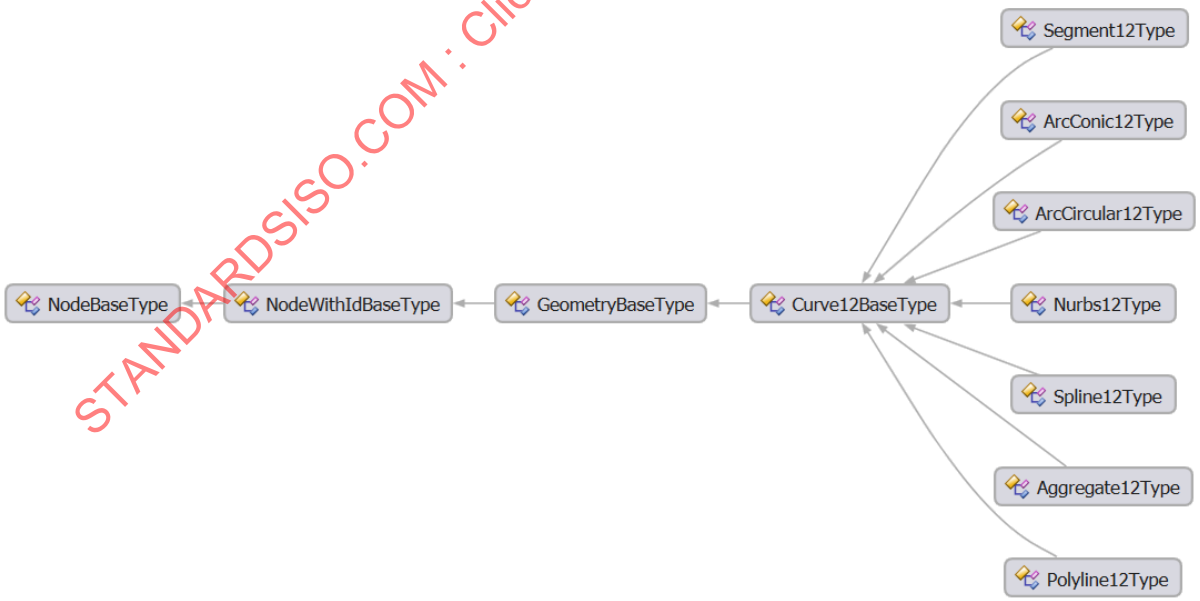


Figure 83 – 2D Curves Types

All 2D geometric curves are represented using a geometric core and a "wrapper" that includes an id for referencing.

The domain of every 2D curve is given in the domain attribute of the core, which is required.

The turned attribute of 2D curves is optional. If it does not appear, its default value is false.

2D curves are normally used to define a trimming curve in the parametric space of a surface.

All 2D curves contain the following fields:

Field Name	Data Type	Description
{Curve}/{Curve}Core	{Curve}CoreType	The curve core.
{Curve}/{Curve}Core/@domain	ParameterRangeType	The domain of the parameter space of the curve.

7.5.2.2.1 *Segment*

Segment12 describes a linear segment in 2D space as shown in Figure 84.

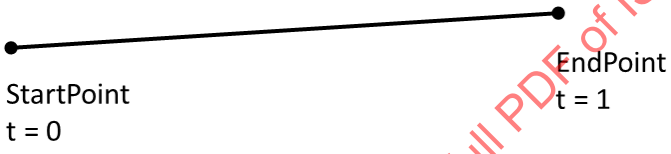


Figure 84 – 2D Segment

A linear segment is defined and positioned in 2D space with start and end points.

Function:

$$Segment12(t)\colon R_1 \rightarrow R_2$$

$$Segment12(t) = StartPoint + t(EndPoint - StartPoint), \qquad t \in [0,1]$$

Fields:

Field Name	Data Type	Description
Segment12Core/StartPoint	Point2dSimpleType	The beginning point of the linear segment.
Segment12Core/EndPoint	Point2dSimpleType	The end point of the linear segment.

Example:

```
<Segment12 id="350">
   <Segment12Core domain="0 1">
      <StartPoint>10.1 -2431.7</StartPoint>
      <EndPoint>34.1 4542.4</EndPoint>
   </Segment12Core>
 </Segment12>
```

7.5.2.2.2 *Polyline*

Polyline12 describes a polyline in 2D space as shown in Figure 85.

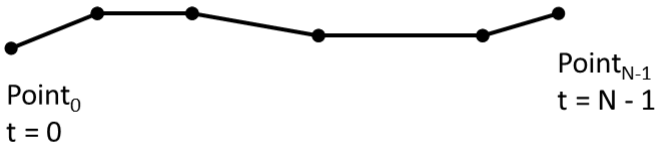


Figure 85 – 2D Polyline

A polyline is defined as a series of connected linear segments.

Function:

$$Polyline12(t)\colon R_1 \rightarrow R_2$$

$$Polyline12(t) = Point_i + (t - i)(Point_{i+1} - Point_i)$$

$$t \in [i, i+1], \qquad i \in [0, N-2], \qquad N - number\ of\ points$$

Fields:

Field Name	Data Type	Description
Polyline12Core/Points or Polyline12Core/PointsBinary	ArrayPoint2dType or ArrayBinaryType	The array of 2D polyline points.

Example:

```
<Polyline12 id="32">
  <Polyline12Core domain="0 4">
    <Points n="5">
      1.2 11.5
      4.1 13.5
      7.3 16.2
      8.9 23.5
      9.2 29.8
    </Points>
  </Polyline12Core>
</Polyline12>
```

7.5.2.2.3 *Circular Arc Curve*

ArcCircular12 describes a circular arc in 2D space as shown in Figure 86 and Figure 87.

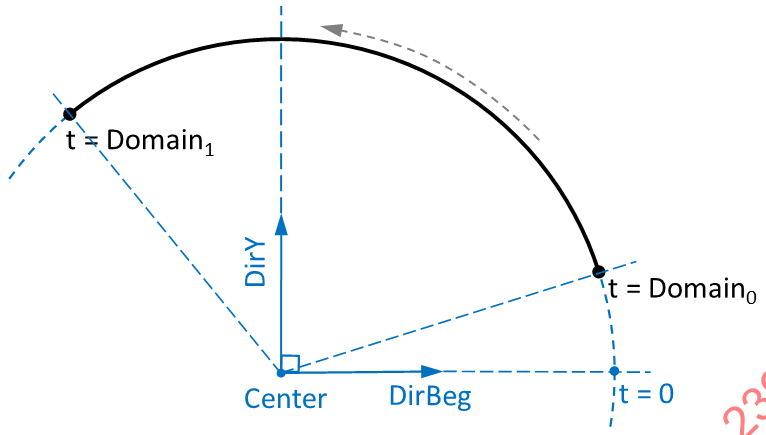


Figure 86 – 2D Circular Arc

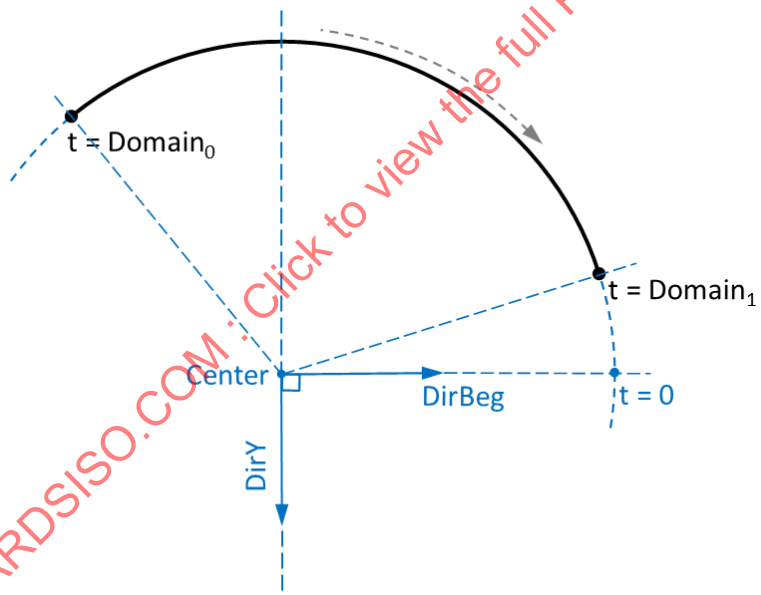


Figure 87 – 2D Circular Arc (turned)

A circular arc is defined as a portion of a circle.

Function:

$$ArcCircular12(t)\colon R_1 \rightarrow R_2$$

$$ArcCircular12(t) = Center + Radius(\cos(t)\, DirBeg + \sin(t)\, DirY)$$

$$DirY = \begin{cases} \begin{bmatrix} -DirBeg_y \\ DirBeg_x \end{bmatrix}, & turned = false \\ \begin{bmatrix} DirBeg_y \\ -DirBeg_x \end{bmatrix}, & turned = true \end{cases}$$

where t is in radians, $t \in (-\infty, +\infty)$

Fields:

Field Name	Data Type	Description
ArcCircular12Core/@turned	xs:boolean	This flag shows if the curve is turned.
ArcCircular12Core/Radius	xs:double	The arc radius.
ArcCircular12Core/Center	Point2dSimpleType	The center of arc.
ArcCircular12Core/DirBeg	UnitVector2dSimpleType	The unit vector representing the beginning of the circular arc.

Example:

```
<ArcCircular12 id="10">
  <ArcCircular12Core domain="0 3.1415926">
    <Radius>10.2</Radius>
    <Center>1.4 6.7</Center>
    <DirBeg>0.0 1.0</DirBeg>
  </ArcCircular12Core>
</ArcCircular12>
```

7.5.2.2.4 *Conic Arc Curve*

ArcConic12 describes a conic arc in 2D space as shown in Figure 88 through Figure 93.

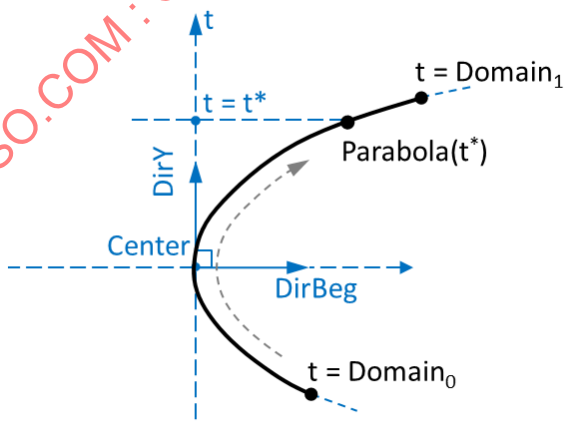


Figure 88 – 2D Conic Arc (form = PARABOLA)

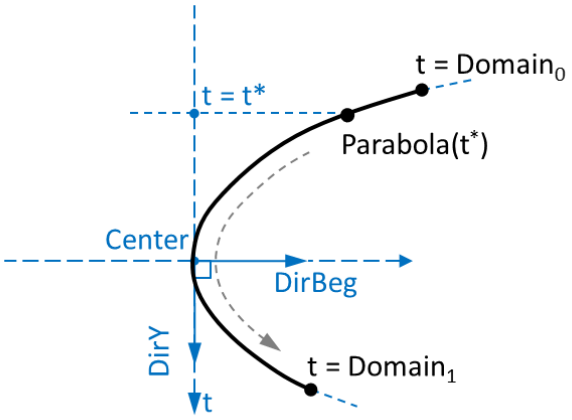


Figure 89 – 2D Conic Arc (form = PARABOLA, turned = true)

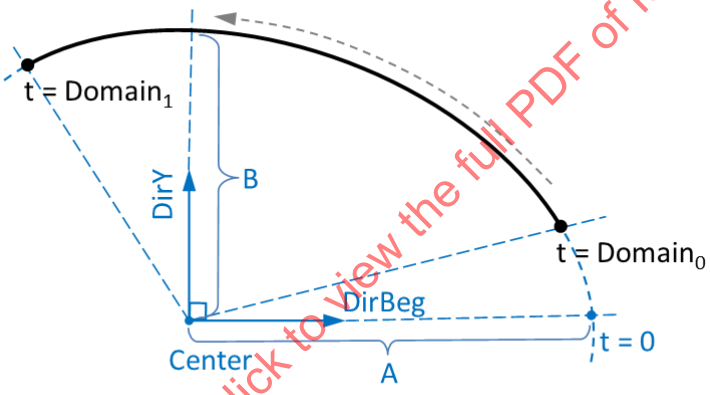


Figure 90 – 2D Conic Arc (form = ELLIPSE)

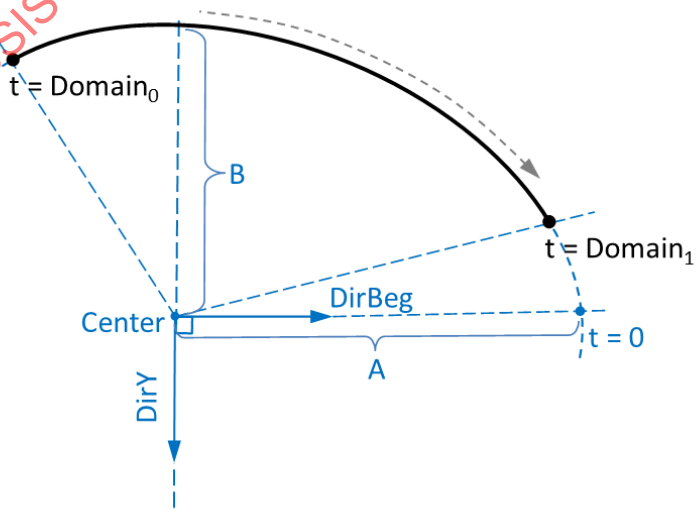


Figure 91 – 2D Conic Arc (form = ELLIPSE, turned = true)

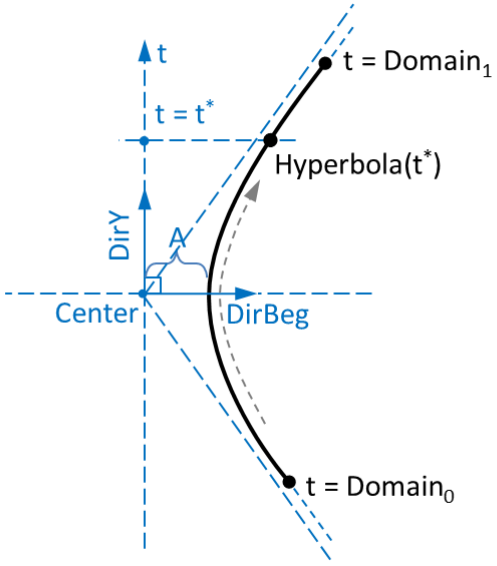


Figure 92 – 2D Conic Arc (form = HYPERBOLA)

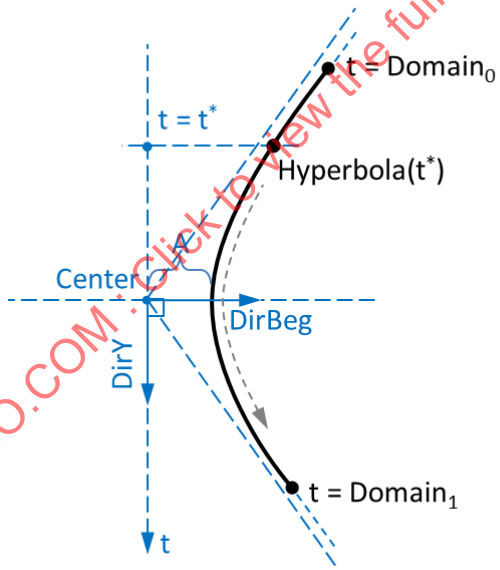


Figure 93 – 2D Conic Arc (form = HYPERBOLA, turned = true)

A conic arc is defined as a portion of a conic curve. The conic curve can have one the following forms: an ellipse, a parabola, or a hyperbola.

Function:

$$ArcConic12(t)\colon R_1 \rightarrow R_2$$

$$ArcConic12(t) = Center + x(t)DirBeg + y(t)DirY$$

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{cases} \begin{bmatrix} At \\ Bt^2 \end{bmatrix}, & form = PARABOLA \\ \begin{bmatrix} A\cos(t) \\ B\sin(t) \end{bmatrix}, & form = ELLIPSE \\ \begin{bmatrix} A\sqrt{1 + t^2/B^2} \\ t \end{bmatrix}, & form = HYPERBOLA \end{cases}$$

$$DirY = \begin{cases} \begin{bmatrix} -DirBeg_y \\ DirBeg_x \end{bmatrix}, & turned = false \\ \begin{bmatrix} DirBeg_y \\ -DirBeg_x \end{bmatrix}, & turned = true \end{cases}$$

$$t \in (-\infty, +\infty)$$

Fields:

Field Name	Data Type	Description
ArcConic12Core/@form	ArcConicFormEnumType	The form of the conic arc.
ArcConic12Core/@turned	xs:boolean	This flag shows if the curve is turned.
ArcConic12Core/A	xs:double	Ellipse: the major radius of the arc. Parabola: the coefficient “A”. Hyperbola: the semi-axis (the distance from the center to the extreme point).
ArcConic12Core/B	xs:double	Ellipse: the minor radius of the arc. Parabola: the coefficient “B”. Hyperbola: the implicit parameter.
ArcConic12Core/Center	Point2dSimpleType	The center point.
ArcConic12Core/DirBeg	UnitVector2dSimpleType	The unit vector representing the beginning of the conic arc.

Example:

```
<ArcConic12 id=”123”>
  <ArcConic12Core form=”PARABOLA” domain=”-0.2 3.5”>
    <A>4.5</A>
    <B>3.1</B>
    <Center>13.3 -43.2</Center>
    <DirBeg>1.0 0.0</DirBeg>
  </ArcConic12Core>
</ArcConic12>

<ArcConic12 id=”123”>
  <ArcConic12Core form=”ELLIPSE” domain=”10.3 18.9” turned=”1”>
    <A>14.2</A>
    <B>33.1</B>
    <Center>63.7 -34.0</Center>
    <DirBeg>1.0 0.0</DirBeg>
  </ArcConic12Core>
</ArcConic12>

<ArcConic12 id=”123”>
  <ArcConic12Core form=”HYPERBOLA” domain=”1.2 7.0”>
    <A>14.5</A>
```



```
    <B>30.1</B>
    <Center>0.3 3.2</Center>
    <DirBeg>1.0 0.0</DirBeg>
  </ArcConic12Core>
</ArcConic12>
```

7.5.2.2.5 *Spline Curve*

Spline12 describes a spline curve in 2D space as shown in Figure 94.

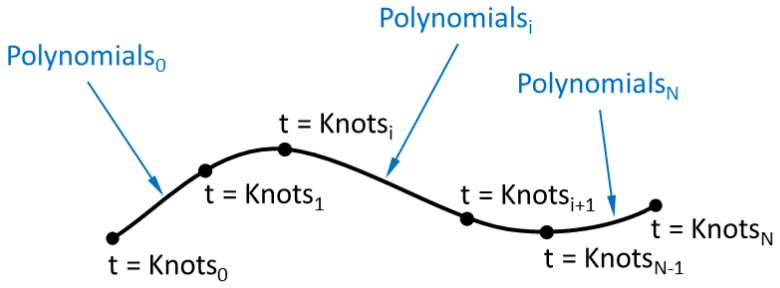


Figure 94 – 2D Spline Curve

A spline curve is defined as a sequence of parametric polynomial segments.

Function:

$$Spline12(t)\colon R_1 \rightarrow R_2$$

$$Spline12(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{Degree_p} (C_i^p)_x (t_p)^i \\ \sum_{i=0}^{Degree_p} (C_i^p)_y (t_p)^i \end{bmatrix}$$

$$N - number\ of\ polynomials$$

$$N = number\ of\ knots - 1$$

$$Coefficients = \{C^0, \ldots C^{N-1}\}$$

$$p - index\ of\ polynomials$$

$$C^p = \{c_0 \ldots c_{Degree_p}\}, \quad Degree_p = Orders_p - 1$$

$$t_p - parameter\ of\ polynomials\ p$$

$$t_p \in \begin{cases} [Knots_p, Knots_{p+1}], & normalized = false \\ [0,1], & normalized = true \end{cases}$$

$$t_p = \begin{cases} t - Knots_p, & Normalized = false \\ (t - Knots_p)/(Knots_{p+1} - Knots_p), & Normalized = true \end{cases}$$

$$t \in [Knots_p, Knots_{p+1}], \qquad p \in [0, N-1]$$

Fields:

Field Name	Data Type	Description

Spline12Core/@normalized	xs:boolean	This flag shows if the curve is normalized.
Spline12Core/Knots	ArrayDoubleType	The spline breakpoints.
Spline12Core/Orders	ArrayNaturalType	The orders of the polynomial segments. The order is 'the degree of the polynomial' + 1. The size of this array is 'the number of spline breakpoints' – 1.
Spline12Core/Coefficients	ArrayPoint2dType	The coefficients of the polynomial segments are pairs. For each segment the number of coefficients is equal to the order on this segment. The total size of this array is the sum of all orders.

Example:

```
<Spline12 id="302">
  <Spline12Core domain="0 2">
    <Knots n="3">
      0 1 2
    </Knots>
    <Orders n="2">
      4 4
    </Orders>
    <Coefficients n="8">
      87.5 237.5
      14.0625 -7.8125
      0 0
      -1.5625 1.5625
      100 231.25
      9.375 -3.125
      -4.6875 4.6875
      1.5625 -1.5625
    </Coefficients>
  </Spline12Core>
</Spline12>
```

7.5.2.2.6 *NURBS Curve*

Nurbs12 describes a NURBS curve in 2D space as shown in Figure 95.

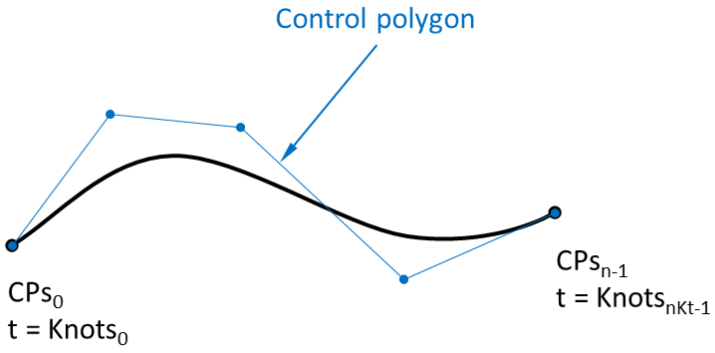


Figure 95 – 2D NURBS Curve

A NURBS curve is a freeform curve built on the B-spline basis functions and defined by its order (= degree + 1), a knot vector (an increasing sequence of real numbers which divides the parametric space in the intervals called knot spans), and an array of control points with an optional set of associated weights (positive real numbers). If the weights are not defined or if the weights are equal, the curve is a polynomial one (otherwise rational).

Function:

$$Nurbs12(t)\colon R_1 \rightarrow R_2$$

$$Nurbs12(t) = \frac{\sum_{i=0}^{n-1} N_{i,Degree}(t) Weights_i CPs_i}{\sum_{i=0}^{n-1} N_{i,Degree}(t) Weights_i}$$

$$Degree = Order - 1$$

$$N_{i,j}(t) - the\ bspline\ basis\ functions$$

$$N_{i,0}(t) = \begin{cases} 1, & t \in [Knots_i, Knots_{i+1}) \\ 0, & otherwise \end{cases}$$

$$N_{i,p}(t) = \frac{t - Knots_i}{Knots_{i+p} - Knots_i} N_{i,p-1}(t) + \frac{Knots_{i+p+1} - t}{Knots_{i+p+1} - Knots_{i+1}} N_{i+1,p-1}(t)$$

$$t \in [Knots_0, Knots_{nKt-1}], \qquad nKt - number\ of\ knots$$

$$nKt = n + Order, \qquad n - number\ of\ control\ points$$

Fields:

Field Name	Data Type	Description
Nurbs12Core/Order	NaturalType	The order (= degree + 1).

Nurbs12Core/Knots	ArrayDoubleType	The knot vector is an increasing sequence of real numbers. The size of the knot vector is “number of control points” + “order”.
Nurbs12Core/CPs or Nurbs12Core/CpsBinary	ArrayPoint2dType or ArrayBinaryType	The array of 2D control points.
Nurbs13Core/Weights	ArrayDoubleType	The array of weights associated with control points (positive real numbers).

Example:

```
<Nurbs12 id="607">
  <Nurbs12Core domain="0 1">
    <Order>3</Order>
    <Knots N="10">
       0 0 0 0.33333333 0.33333333 0.66666667 0.66666667 1 1 1
    </Knots>
    <CPs N="7">
      -386.61 -44.00
      -386.61 -54.39
      -377.61 -49.19
      -368.61 -44.00
      -377.61 -38.80
      -386.61 -33.60
      -386.61 -44.00
    </CPs>
    <Weights N="7">
       1 0.5 1 0.5 1 0.5 1
    </Weights>
  </Nurbs12Core>
</Nurbs12>
```

7.5.2.2.7 *Aggregate Curve*

Aggregate12 describes an aggregate curve in 2D space as shown in Figure 96.

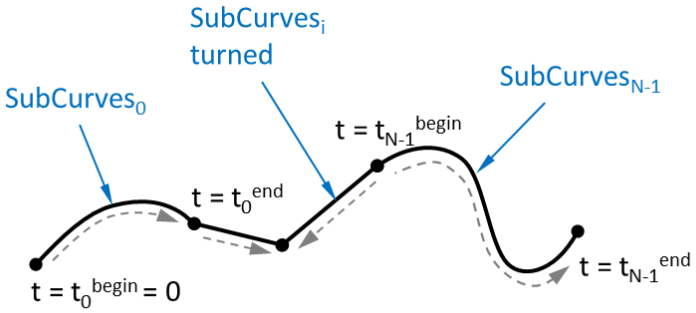


Figure 96 – 2D Aggregate Curve

An aggregate curve is a continuous curve defined as a sequence of oriented parametric sub-curves.

Function:

$$Aggregate12(t)\colon R_1 \to R_2$$

$$Aggregate12(t) = SubCurves_i(t_i)$$

$$t \in [t_i^{begin}, t_i^{end}], \qquad i \in [0, N-1], \qquad N - number\ of\ subcurves$$

$$Domain_i^j := Domain_j\ of\ SubCurves_i, \qquad j \in \{0,1\}$$

$$t_i^{begin} = \sum_{j<i} \left(domain_j^1 - domain_j^0\right)$$

$$t_i^{end} = t_i^{begin} + (domain_i^1 - domain_i^0)$$

$$t_i = \begin{cases} domain_i^0 + (t - t_i^{begin}), & SubCurves_i\ is\ not\ turned \\ domain_i^1 - (t - t_i^{begin}), & SubCurves_i\ is\ turned \end{cases}$$

Fields:

Field Name	Data Type	Description
Aggregate12Core/SubCurves	ArraySubCurve12Type	The array of oriented sub-curves.
Aggregate12Core/SubCurves/SubCurve/@turned	xs:boolean	The orientation of sub-curve.

Example:

```
<Aggregate12 id="419">
  <Aggregate12Core domain="0 4">
    <SubCurves n="4">
      <SubCurve turned="1">
        <Segment12Core domain="0 1">
          <StartPoint>-367.36 -214.75</StartPoint>
```



```
          <EndPoint>-367.36 -207.50</EndPoint>
        </Segment13Core>
      </SubCurve>
      <SubCurve>
        <Nurbs12Core domain="0 1">
          <Order>3</Order>
          <Knots n="6">
            0 0 0 1 1 1
          </Knots>
          <CPs n="3">
            -367.36 -214.75
            -366.61 -214.75
            -366.61 -214.75
          </CPs>
          <Weights n="3">
            1 0.70710678 1
          </Weights>
        </Nurbs12Core>
      </SubCurve>
      <SubCurve>
        <Segment12Core domain="0 1">
          <StartPoint>-366.61 -214.75</StartPoint>
          <EndPoint>-366.61 -207.50</EndPoint>
        </Segment12Core>
      </SubCurve>
      <SubCurve>
        <Nurbs12Core domain="0 1">
          <Order>4</Order>
          <Knots n="10">
            0 0 0 0 0.91406368 0.91406368 1 1 1 1
          </Knots>
          <CPs n="6">
            -366.61 -207.50
            -366.61 -207.50
            -366.88 -207.50
            -367.29 -207.50
            -367.33 -207.50
            -367.36 -207.50
          </CPs>
        </Nurbs12Core>
      </SubCurve>
    </SubCurves>
  </Aggregate12Core>
</Aggregate12>
```


7.5.2.3 3D Curves

This chapter describes curves defined in 3D Model Space, curve(t):R1→ R3, where

- R1 is curve parameter space – the 1D real number space, where each point is defined as a single coordinate (t)
- R3 is model space – the three-dimensional Euclidean space, where each point is defined as a coordinate triple (x, y, z) specified in a right-handed Cartesian coordinate system

The correspondence between 1D parameter space and 3D Euclidean space is illustrated in Figure 97.

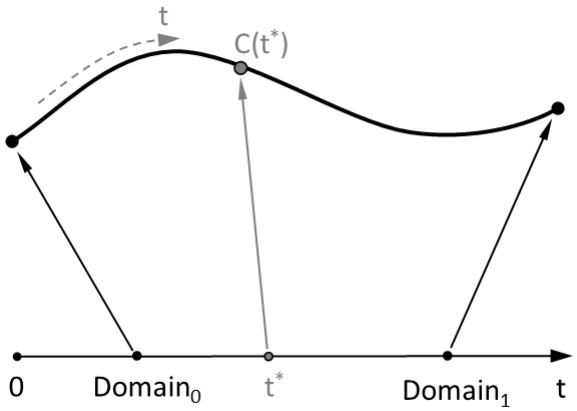


Figure 97 – 3D Parametric Curve

All types of model space curves are derived from Curve13BaseType as shown in Figure 98.

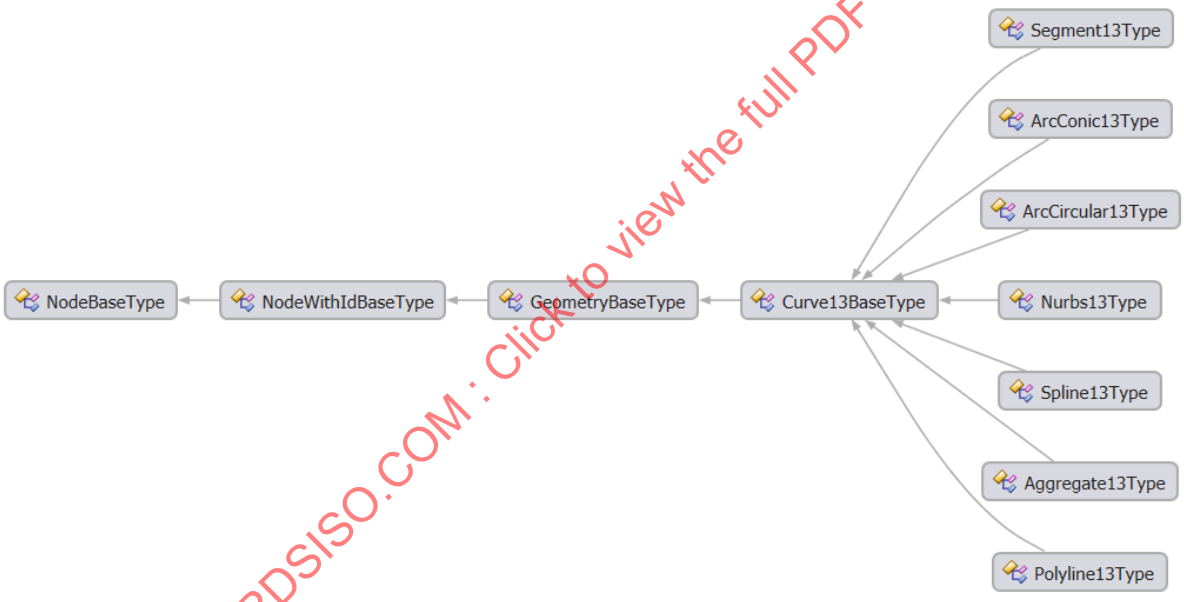


Figure 98 – 3D Curve Types

All 3D geometric curves are represented using a geometric core and a “wrapper" that includes an id for referencing.

The domain of every 3D curve is given in the domain attribute of the core, which is required.

The turned attribute of 3D curves is optional. If it does not appear, its default value is false.

The 3D curves are normally used as underlying geometry for topological edges.

All 3D curves contain the following fields:

Field Name	Data Type	Description
{Curve}/{Curve}Core	{Curve}CoreType	The curve core.
{Curve}/{Curve}Core/@domain	ParameterRangeType	The domain of the parameter space of the curve.
{Curve}/Transform	ElementReferenceType	The identifier of a three-dimensional transformation matrix.

7.5.2.3.1 *Segment*

Segment13 describes a linear segment in 3D space as shown in Figure 99.

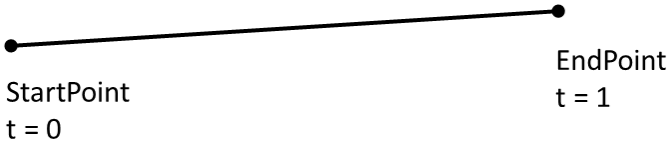


Figure 99 – 3D Segment

A linear segment is defined and positioned in 3D space with start and end points.

Function:

$$Segment13(t)\colon R_1 \rightarrow R_3$$

$$Segment13(t) = StartPoint + t(EndPoint - StartPoint), \qquad t \in [0,1]$$

Fields:

Field Name	Data Type	Description
Segment13Core/StartPoint	PointSimpleType	The beginning point of the linear segment.
Segment13Core/EndPoint	PointSimpleType	The end point of the linear segment.

Example:

```
<Segment13 id="350">
   <Segment13Core domain="0 1">
      <StartPoint>10.1 -2431.7 13.7</StartPoint>
      <EndPoint>34.1 4542.4 34.4</EndPoint>
   </Segment13Core>
 </Segment13>
```

7.5.2.3.2 *Polyline*

Polyline13 describes a polyline in 3D space as shown in Figure 100.

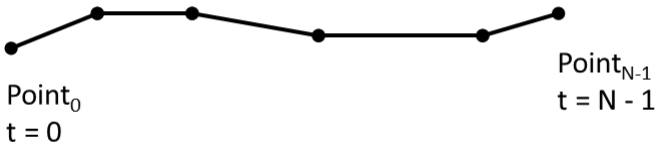


Figure 100 – 3D Polyline

A polyline is defined as a series of connected linear segments.

Function:

$$Polyline13(t)\colon R_1 \rightarrow R_3$$

$$Polyline13(t) = Point_i + (t - i)(Point_{i+1} - Point_i)$$

$$t \in [i, i+1], \qquad i \in [0, N-2], \qquad N - number\ of\ points$$

Fields:

Field Name	Data Type	Description
Polyline13Core/Points or Polyline13Core/PointsBinary	ArrayPointType or ArrayBinaryType	The array of 3D polyline points.

Example:

```
<Polyline13 id="32">
  <Polyline13Core domain="0 4">
    <Points n="5">
      1.2 11.5 10.4
      4.1 13.5 14.4
      7.3 16.2 17.1
      8.9 23.5 65.3
      9.2 29.8 84.4
    </Points>
  </Polyline13Core>
</Polyline13>
```

7.5.2.3.3 *Circular Arc Curve*

ArcCircular13 describes a circular arc in 3D space.

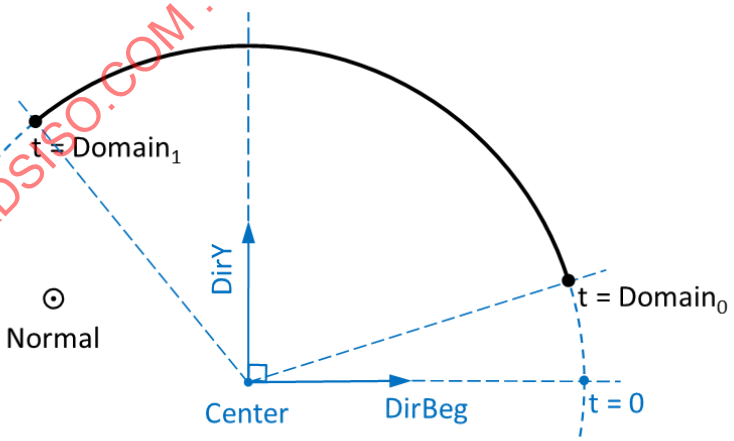


Figure 101 – 3D Circular Arc

A circular arc is defined as a portion of a circle as shown in Figure 101.

Function:

$ArcCircular13(t)\colon R_1 \rightarrow R_3$

$ArcCircular13(t) = Center + Radius(\cos(t)\, DirBeg + \sin(t)\, DirY)$

$DirY = Normal \times DirBeg$

where t is in radians, $t \in (-\infty, +\infty)$

Fields:

Field Name	Data Type	Description
ArcCircular13Core/Radius	xs:double	The arc radius.
ArcCircular13Core/Center	PointSimpleType	The center of arc.
ArcCircular13Core/DirBeg	UnitVectorSimpleType	The unit vector representing the beginning of the circular arc. The DirBeg must be perpendicular to the Normal.
ArcCircular13Core/Normal	UnitVectorSimpleType	The normal of the plane wherein the circular arc is defined.

Example:

```
<ArcCircular13 id="10">
  <ArcCircular13Core domain="0 3.1415926">
    <Radius>10.2</Radius>
    <Center>1.4 6.7 9.0</Center>
    <DirBeg>1.0 0.0 0.0</DirBeg>
    <Normal>0.0 0.0 1.0</Normal>
  </ArcCircular13Core>
</ArcCircular13>
```


7.5.2.3.4 *Conic Arc Curve*

ArcConic13 describes a conic arc in 3D space as shown in Figure 102 through Figure 104.

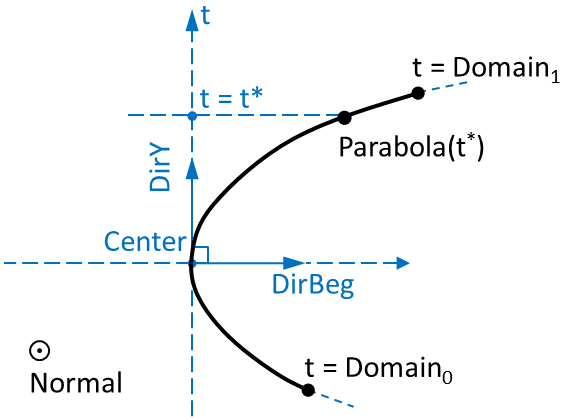


Figure 102 – 3D Conic Arc (form = PARABOLA)

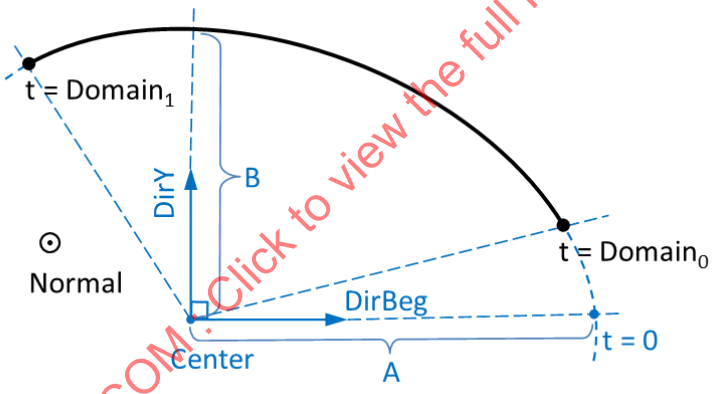


Figure 103 – 3D Conic Arc (form = ELLIPSE)

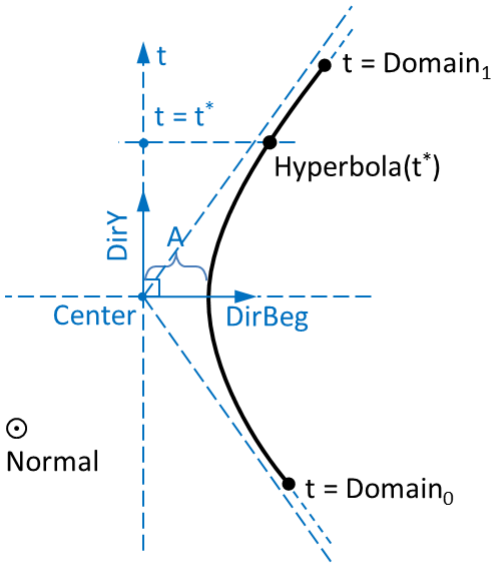


Figure 104 – 3D Conic Arc (form = HYPERBOLA)

A conic arc is defined as a portion of a conic curve. The conic curve can have one the following forms: an ellipse, a parabola, or a hyperbola.

Function:

$$ArcConic13(t)\colon R_1 \rightarrow R_3$$

$$ArcConic13(t) = Center + x(t)DirBeg + y(t)DirY$$

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{cases} \begin{bmatrix} At \\ Bt^2 \end{bmatrix}, & form = PARABOLA \\ \begin{bmatrix} A\cos(t) \\ B\sin(t) \end{bmatrix}, & form = ELLIPSE \\ \begin{bmatrix} A\sqrt{1+t^2}/B^2 \\ t \end{bmatrix}, & form = HYPERBOLA \end{cases}$$

$$DirY = Normal \times DirBeg$$

$$t \in (-\infty, +\infty)$$

Fields:

Field Name	Data Type	Description
ArcConic13Core/@form	ArcConicFormEnumType	The form of the conic arc.
ArcConic13Core/A	xs:double	Ellipse: the major radius of the arc. Parabola: the coefficient “A”. Hyperbola: the semi-axis (the distance from the center to the extreme point).
ArcConic13Core/B	xs:double	Ellipse: the minor radius of the arc. Parabola: the coefficient “B”. Hyperbola: the implicit parameter.

ArcConic13Core/Center	PointSimpleType	The center point.
ArcConic13Core/DirBeg	UnitVectorSimpleType	The unit vector representing the beginning of the conic arc. The DirBeg must be perpendicular to the Normal.
ArcConic13Core/Normal	UnitVectorSimpleType	The normal of the plane wherein the conic arc is defined.

Example:

```
<ArcConic13 id="123">
  <ArcConic13Core form="PARABOLA" domain="-0.2 3.5">
    <A>4.5</A>
    <B>3.1</B>
    <Center>13.3 -43.2 34.1</Center>
    <DirBeg>1.0 0.0 0.0</DirBeg>
    <Normal>0.0 1.0 0.0</Normal>
  </ArcConic13Core>
</ArcConic13>

<ArcConic13 id="123">
  <ArcConic13Core form="ELLIPSE" domain="10.3 18.9">
    <A>14.2</A>
    <B>33.1</B>
    <Center>63.7 -34.0 45.9</Center>
    <DirBeg>1.0 0.0 0.0</DirBeg>
    <Normal>0.0 1.0 0.0</Normal>
  </ArcConic13Core>
</ArcConic13>

<ArcConic13 id="123">
  <ArcConic13Core form="HYPERBOLA" domain="1.2 7.0">
    <A>14.5</A>
    <B>30.1</B>
    <Center>0.3 3.2 5.8</Center>
    <DirBeg>1.0 0.0 0.0</DirBeg>
    <Normal>0.0 1.0 0.0</Normal>
  </ArcConic13Core>
</ArcConic13>
```

7.5.2.3.5 *Spline Curve*

Spline13 describes a spline curve in 3D space as shown in Figure 105.

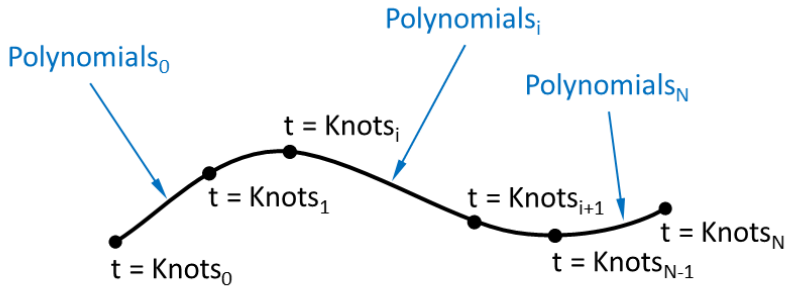


Figure 105 – 3D Spline Curve

A spline curve is defined as a sequence of parametric polynomial segments.

Function:

$$Spline13(t)\colon R_1 \rightarrow R_3$$

$$Spline13(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{matrix} \sum_{i=0}^{Degree_p} (C_i^p)_x (t_p)^i \\ \sum_{i=0}^{Degree_p} (C_i^p)_y (t_p)^i \\ \left\lfloor \sum_{i=0}^{Degree_p} (C_i^p)_z (t_p)^i \right\rfloor \end{matrix}$$

$$N - number\ of\ polynomials$$

$$N = number\ of\ knots - 1$$

$$Coefficients = \{C^0, \ldots, C^{N-1}\}$$

$$p - index\ of\ polynomials$$

$$C^p = \{c_0 \ldots c_{Degree_p}\}, \quad Degree_p = Orders_p - 1$$

$$t_p - parameter\ of\ polynomials\ p$$

$$t_p \in \begin{cases} [Knots_p, Knots_{p+1}], & normalized = false \\ [0,1], & normalized = true \end{cases}$$

$$t_p = \begin{cases} t - Knots_p, & normalized = false \\ (t - Knots_p)/(Knots_{p+1} - Knots_p), & normalized = true \end{cases}$$

$$t \in [Knots_p, Knots_{p+1}], \qquad p \in [0, N-1]$$

Fields:

Field Name	Data Type	Description
Spline13Core/@normalized	xs:boolean	This flag shows if the curve is normalized.
Spline13Core/Knots	ArrayDoubleType	The spline breakpoints.
Spline13Core/Orders	ArrayNaturalType	The orders of the polynomial segments. The order is 'the degree of the polynomial' + 1. The size of this array is 'the number of spline breakpoints' - 1.
Spline13Core/Coefficients	ArrayPointType	The coefficients of the polynomial segments are triplets. For each segment the number of coefficients equal to the order on this segment. The total size of this array is the sum of all orders.

Example:

```
<Spline13 id="302">
  <Spline13Core domain="0 2">
    <Knots n="3">
      0 1 2
    </Knots>
    <Orders n="2">
      4 4
    </Orders>
    <Coefficients n="8">
      87.5 237.5 0
      14.0625 -7.8125 0
      0 0 0
      -1.5625 1.5625 0
      100 231.25 0
      9.375 -3.125 0
      -4.6875 4.6875 0
      1.5625 -1.5625 0
    </Coefficients>
  </Spline13Core>
</Spline13>
```

7.5.2.3.6 *NURBS Curve*

Nurbs13 describes a NURBS curve in 3D space as shown in Figure 106.

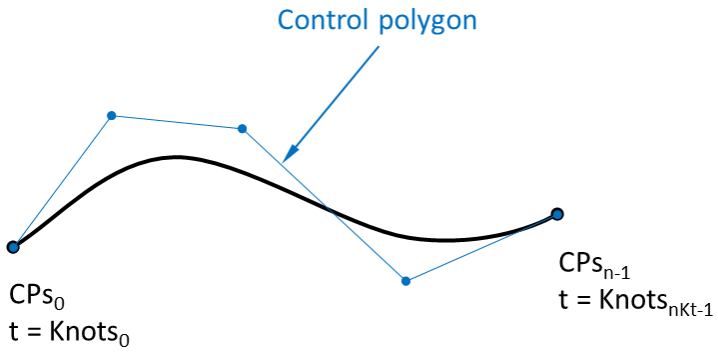


Figure 106 – 3D NURBS Curve

A NURBS curve is a freeform curve built on the B-spline basis functions and defined by its order (= degree + 1), a knot vector (an increasing sequence of real numbers which divides the parametric space in the intervals called knot spans), and an array of control points with an optional set of associated weights (positive real numbers). If the weights are not defined or if the weights are equal, the curve is a polynomial one (otherwise rational).

Function:

$$Nurbs13(t)\colon R_1 \to R_3$$

$$Nurbs13(t) = \frac{\sum_{i=0}^{n-1} N_{i,Degree}(t) Weights_i CPs_i}{\sum_{i=0}^{n-1} N_{i,Degree}(t) Weights_i}$$

$$Degree = Order - 1$$

$$N_{i,j}(t) - the\ bspline\ basis\ functions$$

$$N_{i,0}(t) = \begin{cases} 1, & t \in [Knots_i, Knots_{i+1}) \\ 0, & otherwise \end{cases}$$

$$N_{i,p}(t) = \frac{t - Knots_i}{Knots_{i+p} - Knots_i} N_{i,p-1}(t) + \frac{Knots_{i+p+1} - t}{Knots_{i+p+1} - Knots_{i+1}} N_{i+1,p-1}(t)$$

$$t \in [Knots_0, Knots_{nKt-1}], \qquad nKt - number\ of\ knots$$

$$nKt = n + Order, \qquad n - number\ of\ control\ points$$

Fields:

Field Name	Data Type	Description
Nurbs13Core/Order	NaturalType	The order (= degree + 1).

Nurbs13Core/Knots	ArrayDoubleType	The knot vector is an increasing sequence of real numbers. The size of the knot vector is “number of control points” + “order”.
Nurbs13Core/CPs or Nurbs13Core/CPsBinary	ArrayPointType or ArrayBinaryType	The array of 3D control points.
Nurbs13Core/Weights	ArrayDoubleType	The array of weights associated with control points (positive real numbers).

Example:

```
<Nurbs13 id="607">
  <Nurbs13Core domain="0 1">
    <Order>3</Order>
    <Knots n="10">
       0 0 0 0.33333333 0.33333333 0.66666667 0.66666667 1 1 1
    </Knots>
    <CPs n="7">
      -386.61 -291.50 -44.00
      -386.61 -291.50 -54.39
      -377.61 -291.50 -49.19
      -368.61 -291.50 -44.00
      -377.61 -291.50 -38.80
      -386.61 -291.50 -33.60
      -386.61 -291.50 -44.00
    </CPs>
    <Weights n="7">
       1 0.5 1 0.5 1 0.5 1
    </Weights>
  </Nurbs13Core>
</Nurbs13>
```

7.5.2.3.7 *Aggregate Curve*

Aggregate13 describes an aggregate curve in 3D space as shown in Figure 107.

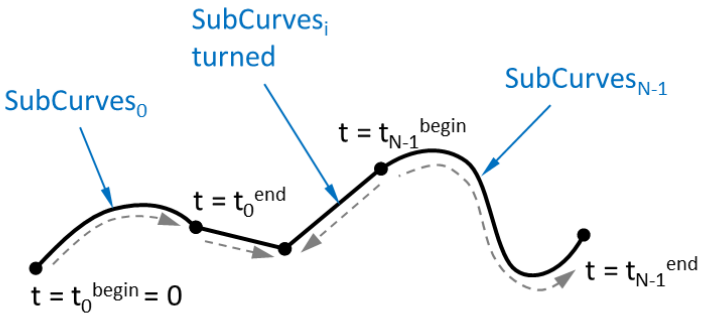


Figure 107 – 3D Aggregate Curve

An aggregate curve is a continuous curve defined as a sequence of oriented parametric sub-curves.

Function:

$$Aggregate13(t)\colon R_1 \to R_3$$

$$Aggregate13(t) = SubCurves_i(t_i)$$

$$t \in [t_i^{begin}, t_i^{end}], \qquad i \in [0, N-1], \qquad N - number\ of\ subcurves$$

$$domain_i^j := domain_j\ of\ SubCurves_i, \qquad j \in \{0,1\}$$

$$t_i^{begin} = \sum_{j<i} \left(domain_j^1 - domain_j^0\right)$$

$$t_i^{end} = t_i^{begin} + (domain_i^1 - domain_i^0)$$

$$t_i = \begin{cases} domain_i^0 + (t - t_i^{begin}), & SubCurves_i\ is\ not\ turned \\ domain_i^1 - (t - t_i^{begin}), & SubCurves_i\ is\ turned \end{cases}$$

Fields:

Field Name	Data Type	Description
Aggregate13Core/SubCurves	ArraySubCurve13Type	The array of oriented sub-curves.
Aggregate13Core/SubCurves/SubCurve/@turned	xs:boolean	The orientation of sub-curve.

Example:

```
<Aggregate13 id="419">
  <Aggregate13Core domain="0 4">
    <SubCurves n="4">
      <SubCurve turned="1">
        <Segment13Core domain="0 1">
          <StartPoint>-367.36 -214.75 -36.50</StartPoint>
```



```
          <EndPoint>-367.36 -207.50 -36.50</EndPoint>
        </Segment13Core>
      </SubCurve>
      <SubCurve>
        <Nurbs13Core domain="0 1">
          <Order>3</Order>
          <Knots n="6">
            0 0 0 1 1 1
          </Knots>
          <CPs n="3">
            -367.36 -214.75 -36.50
            -366.61 -214.75 -36.50
            -366.61 -214.75 -37.25
          </CPs>
          <Weights n="3">
            1 0.70710678 1
          </Weights>
        </Nurbs13Core>
      </SubCurve>
      <SubCurve>
        <Segment13Core domain="0 1">
          <StartPoint>-366.61 -214.75 -37.25</StartPoint>
          <EndPoint>-366.61 -207.50 -37.25</EndPoint>
        </Segment13Core>
      </SubCurve>
      <SubCurve>
        <Nurbs13Core domain="0 1">
          <Order>4</Order>
          <Knots n="10">
            0 0 0 0 0.91406368 0.91406368 1 1 1 1
          </Knots>
          <CPs n="6">
            -366.61 -207.50 -37.25
            -366.61 -207.50 -36.87
            -366.88 -207.50 -36.56
            -367.29 -207.50 -36.50
            -367.33 -207.50 -36.50
            -367.36 -207.50 -36.50
          </CPs>
        </Nurbs13Core>
      </SubCurve>
    </SubCurves>
  </Aggregate13Core>
</Aggregate13>
```


7.5.2.4 Parametric Surfaces

This chapter describes parametric surfaces, S(u, v):R2 → R3, where

- R2 is surface parameter space - the 2D real number space, where each point is defined as a pair (u, v)
- R3 is model space - the three-dimensional Euclidean space, where each point is defined as a coordinate triplet (x, y, z) specified in a right-handed Cartesian coordinate system

The correspondence between 2D parameter space and 3D Euclidean space is illustrated in Figure 108.

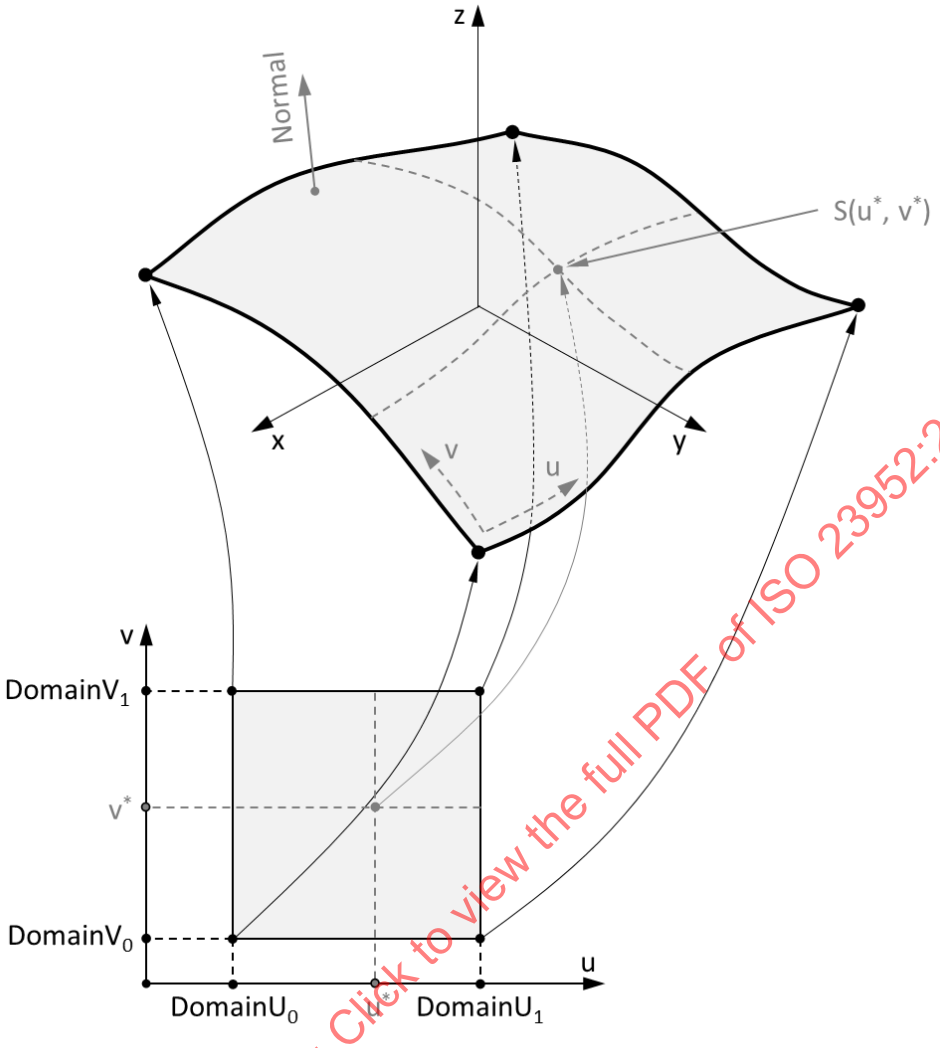


Figure 108 – Parametric Surface

All types of QIF surfaces are derived from **SurfaceBaseType** as shown in Figure 109.

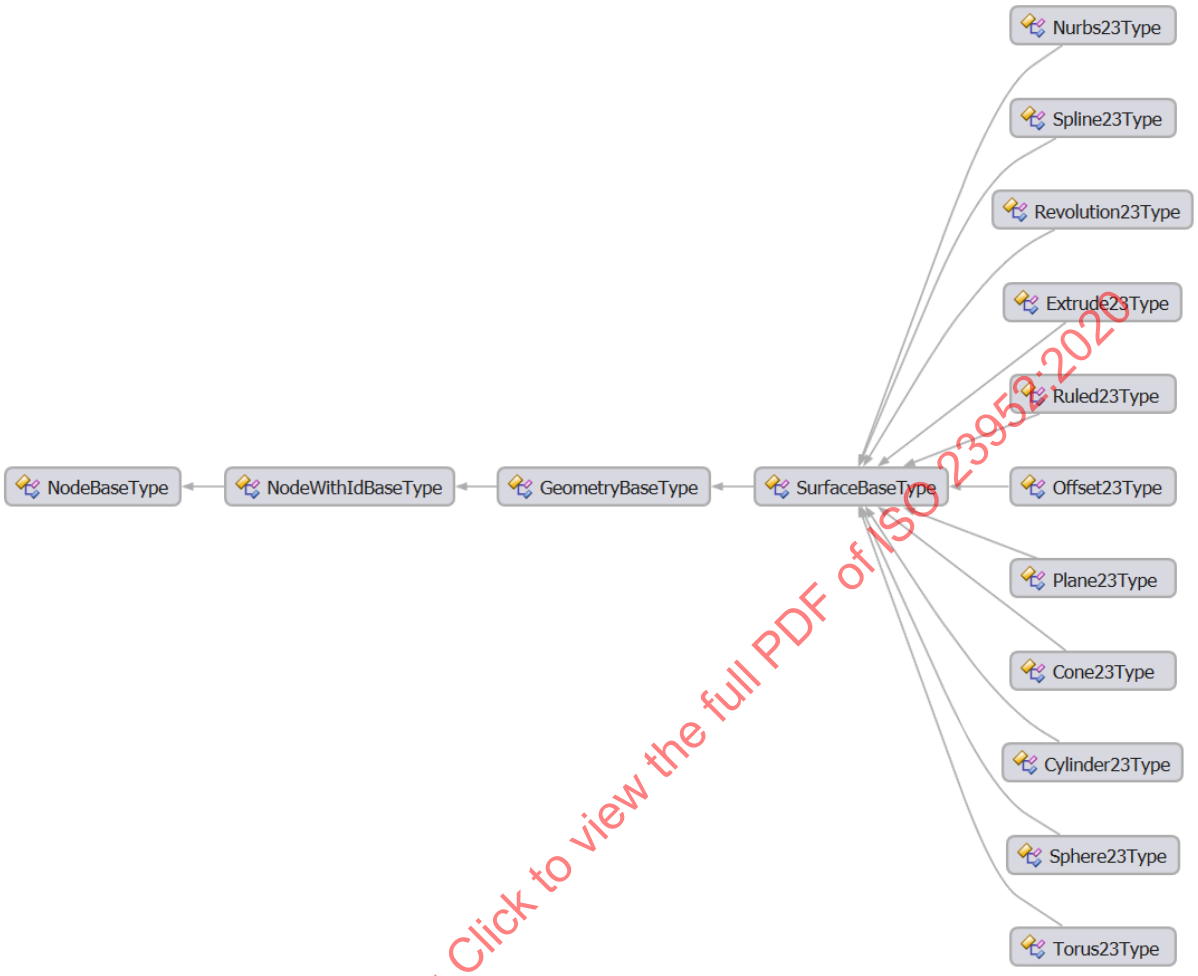


Figure 109 – Parametric Surface Types

The surfaces are normally used as underlying geometry for topological faces.

All parametric surfaces contain the following fields:

Field Name	Data Type	Description
{Curve}/{Curve}Core	{Curve}CoreType	The surface core.
{Curve}/{Curve}Core/@form	Attr23CoreEnumType	This field enumerates values that describe the surface form: 'FREEFORM' - a freeform surface (NURBS, spline etc.); 'CYLINDER' - a cylindrical surface; 'CONE' - a conical surface; 'TORUS' - a toroidal surface; 'SPHERE' - a spherical surface; 'PLANE' - a plane surface;
{Curve}/Transform	ElementReferenceType	The identifier of a three-dimensional transformation matrix.

About the scaling coefficient

The scaling coefficient scaleV normalizes the parameterization in the v-direction, so very large and very small cylinders, cones and tori have reasonable parameterization. Additionally, the scaling coefficient preserves approximate angle correspondence in 2D and 3D space as shown in Figure 110.

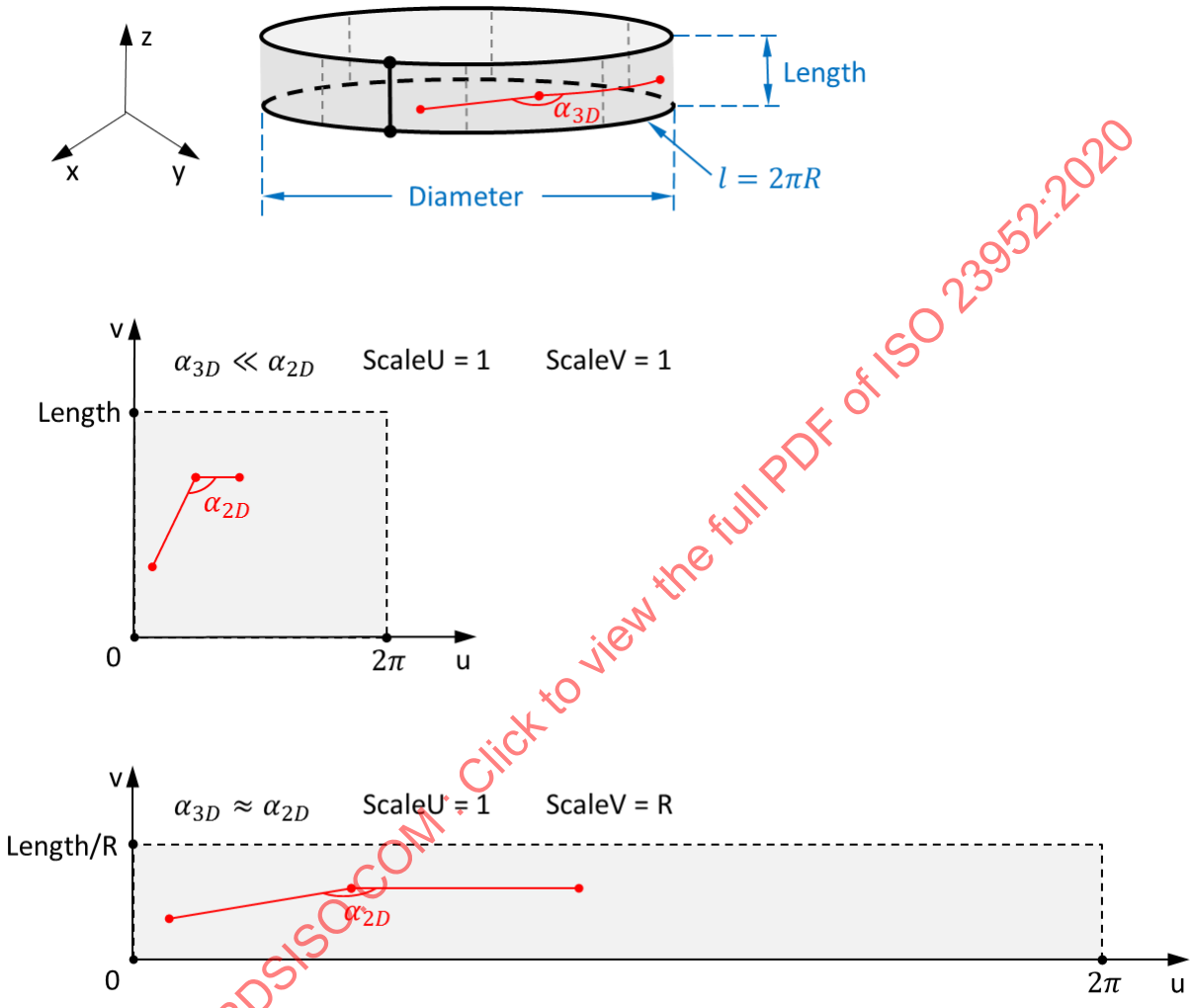


Figure 110 – Scaling coefficient

7.5.2.4.1 *Plane Surface*

Plane23 describes a plane parametric surface as shown in Figure 111.

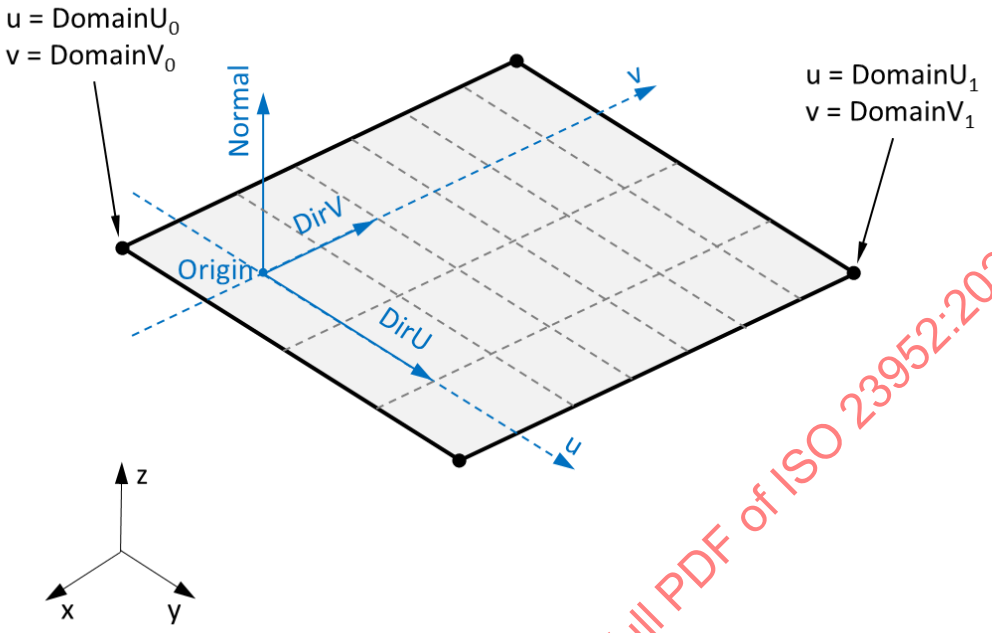


Figure 111 – Plane

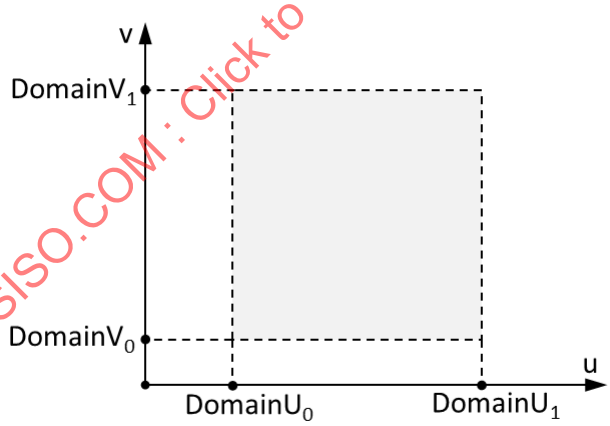


Figure 112 – Plane (Parameter Space)

Plane is an infinite flat surface. The parameterization of the plane is defined by two vectors: DirU and DirV. As seen in Figure 112, domain values (domainU and domainV) 'limit' the plane infinity.

Function:

$$Plane23(u, v)\colon R_2 \rightarrow R_3$$

$$Plane23(u, v) = Origin + uDirU + vDirV$$

$$u \in [domainU_0, domainU_1], \qquad v \in [domainV_0, domainV_1]$$

$$Normal = \frac{DirU \times DirV}{\|DirU \times DirV\|}$$

Fields:

Field Name	Data Type	Description
Plane23Core/@domainU	ParameterRangeType	The parameter domain in the U-direction.
Plane23Core/@domainV	ParameterRangeType	The parameter domain in the V-direction.
Plane23Core/Origin	PointSimpleType	The origin point on the plane.
Plane23Core/DirU	VectorSimpleType	The direction and scaling of U-parameter lines. The DirU vector must not be parallel or anti-parallel to the DirV vector.
Plane23Core/DirV	VectorSimpleType	The direction and scaling of V-parameter lines.

Example:

```
<Plane23 id="233">
  <Plane23Core domainU="-1.0 10.0" domainV="3.0 4.0">
    <Origin>3.0 0.0 5.5</Origin>
    <DirU>2.0 0.0 0.0</DirU>
    <DirV>0.0 3.0 0.0</DirV>
  </Plane23Core>
</Plane23>
```


7.5.2.4.2 *Cylinder Surface*

Cylinder23 describes a cylinder parametric surface as shown in Figure 113, Figure 114, and Figure 115.

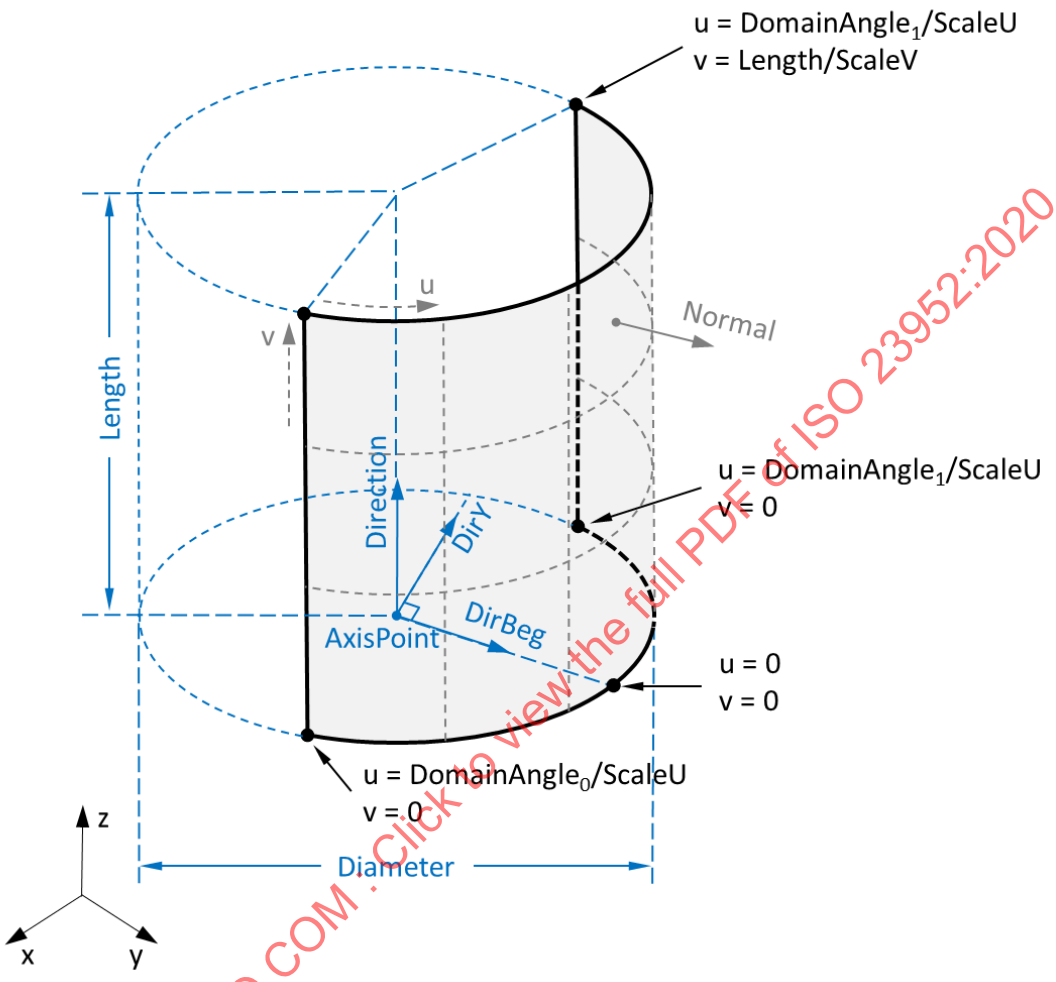


Figure 113 – Cylinder

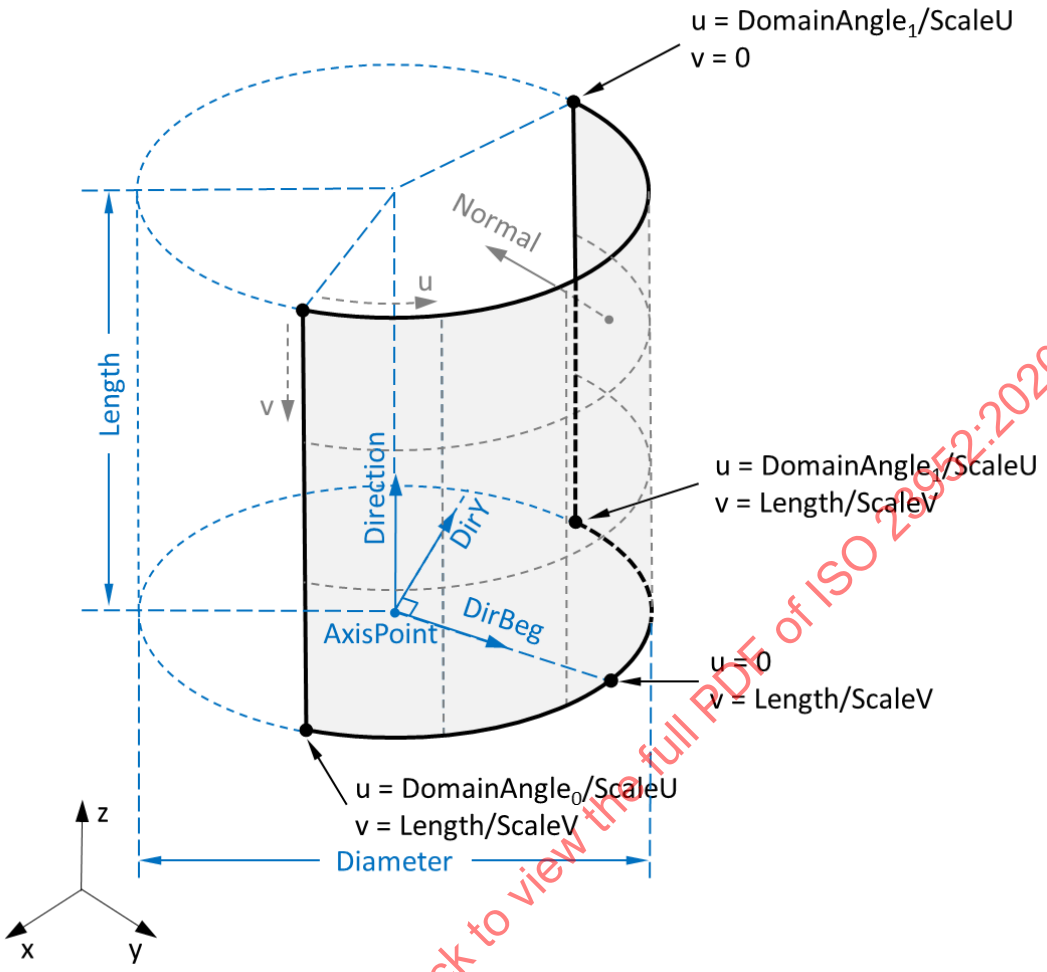


Figure 114 – Cylinder (turnedV = true)

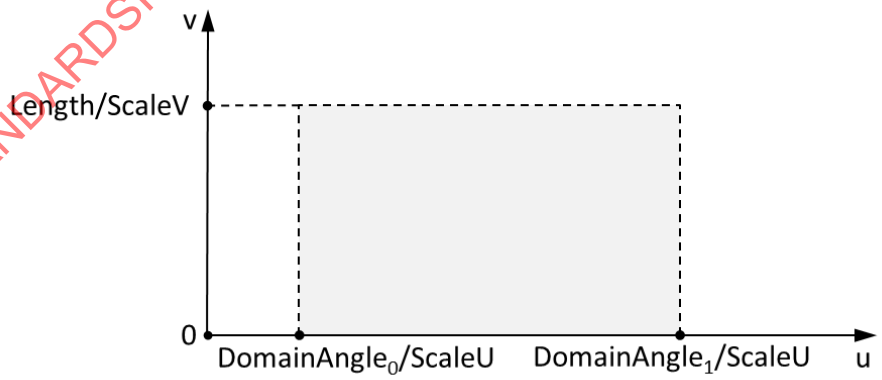


Figure 115 – Cylinder (Parametric Space)

The cylinder is a surface generated by rotating a 3D linear segment around a specified axis on a specified angle. The rotating segment must be parallel to the specified axis.

Function:

$$Cylinder23(u,v)\colon R_2 \to R_3$$

$$Cylinder23(u,v) = AxisPoint + R(\cos(u')DirBeg + \sin(u')DirY) + v'Direction$$

$$R = \frac{Diameter}{2}, \qquad DirY = Direction \times DirBeg$$

$$u' = u * scaleU$$

$$where\ u'\ is\ measured\ in\ radians$$

$$v' = \begin{cases} v * scaleV, & turnedV = false \\ Length - v * scaleV, & turnedV = true \end{cases}$$

$$u \in \left[\frac{domainAngle_0}{scaleU}, \frac{domainAngle_1}{scaleU}\right], \qquad v \in [0, Length/scaleV]$$

$$where\ DomainAngle\ in\ formula\ is\ measured\ in\ radians,$$
$$and\ DomainAngle\ in\ file\ is\ measured\ in\ the\ units\ specified\ in\ the\ element,$$
$$or\ in\ radians\ if\ no\ units\ are\ specified$$

Fields:

Field Name	Data Type	Description
Cylinder23Core/@turnedV	xs:boolean	This flag shows if the v direction of the cylinder must be inverted.
Cylinder23Core/@scaleU	DoublePositiveType	The scaling coefficient of the u direction of the parametric space.
Cylinder23Core/@scaleV	DoublePositiveType	The scaling coefficient of the v direction of the parametric space.
Cylinder23Core/Diameter	xs:double	The cylinder diameter.
Cylinder23Core/Length	xs:double	The height of the cylinder – a distance between the cylinder top and bottom.
Cylinder23Core/Axis/AxisPoint	PointType	The axis origin (the bottom center).
Cylinder23Core/Axis/Direction	UnitVectorType	The axis vector.
Cylinder23Core/Sweep/DirBeg	UnitVectorType	The start direction specifies the position of the cylinder seam (u=0). This vector must lie in a plane normal to the axis of the cylinder.
Cylinder23Core/Sweep/DomainAngle	AngleRangeType	The sweep angle from the start direction (the surface domain in the u direction).

Example:

```
<Cylinder23 id="132">
  <Cylinder23Core>
    <Diameter>4.4</Diameter>
    <Length>7.1</Length>
    <Axis>
      <AxisPoint>10.0 0.0 0.0</AxisPoint>
      <Direction>0.0 0.0 1.0</Direction>
```

```
    </Axis>
    <Sweep>
      <DirBeg>1.0 0.0 0.0</DirBeg>
      <DomainAngle>1.3 1.9</DomainAngle>
    </Sweep>
  </Cylinder23Core>
</Cylinder23>
```

7.5.2.4.3 *Cone Surface*

Cone23 describes a cone parametric surface as shown in Figure 116, Figure 117, and Figure 118.

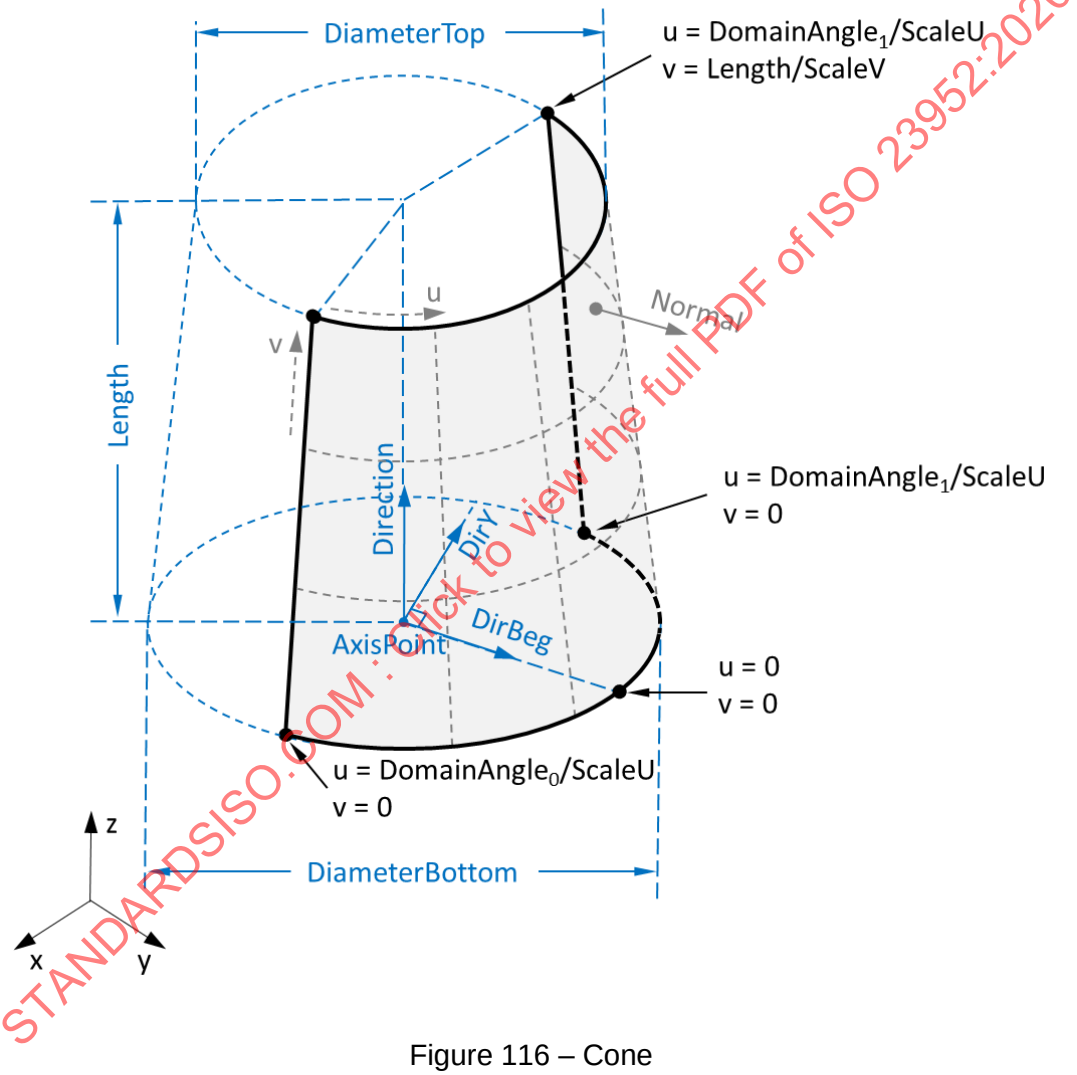


Figure 116 – Cone

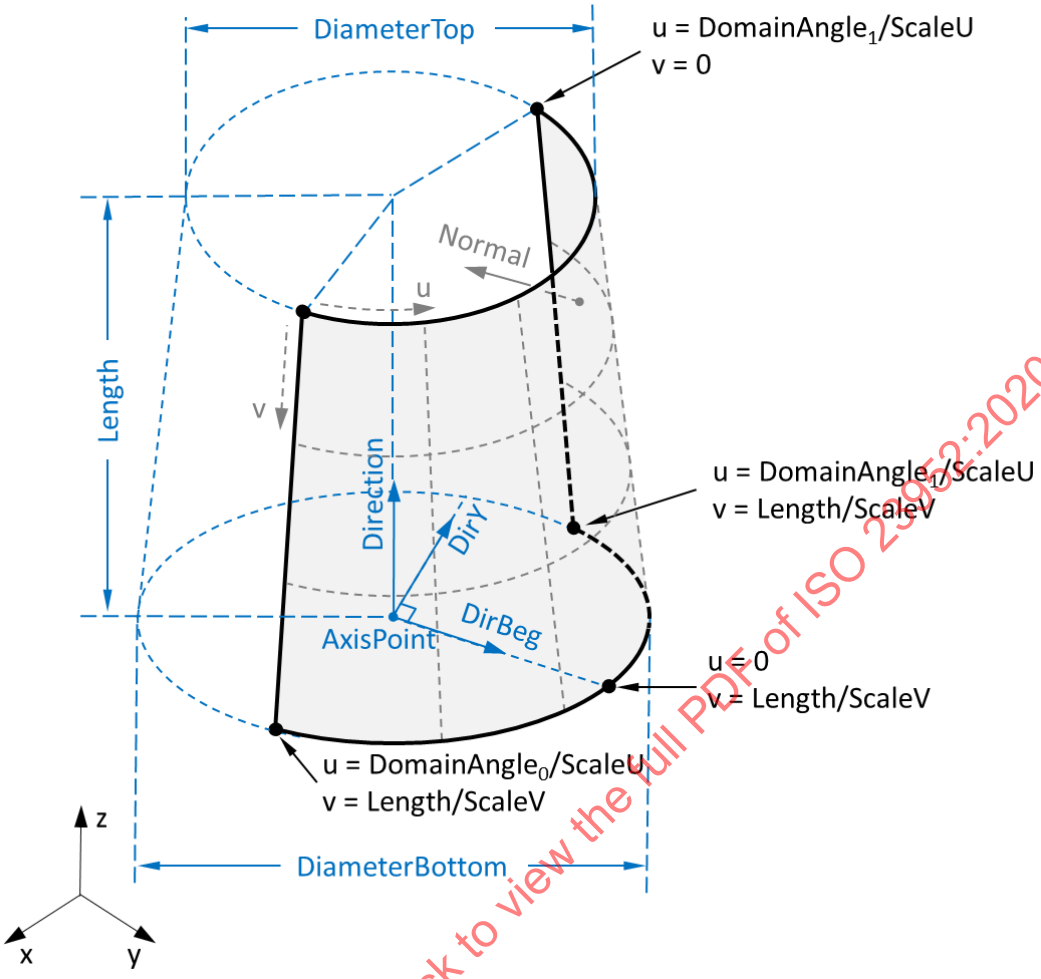


Figure 117 – Cone (turnedV = true)

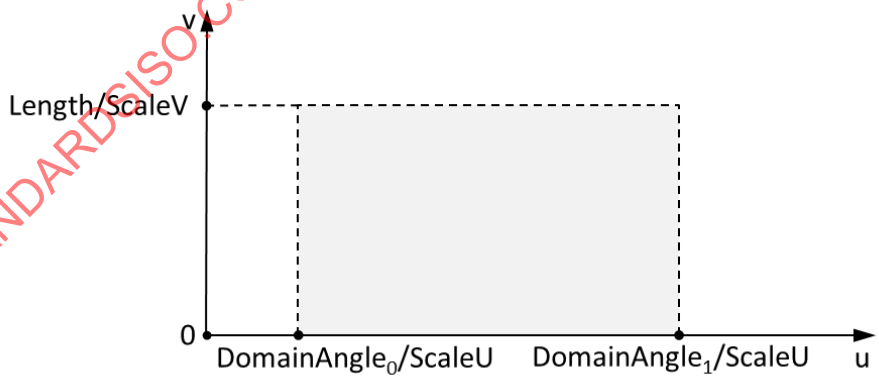


Figure 118 – Cone (Parametric Space)

The cone is a surface generated by rotating a 3D linear segment around a specified axis on a specified angle. The axis and the rotating segment must be coplanar.

Function:

$$Cone23(u,v)\colon R_2 \to R_3$$

$$Cone23(u,v) = AxisPoint + R(v')(\cos(u')DirBeg + \sin(u')DirY) + v'Direction$$

$$RadiusTop = \frac{DiameterTop}{2}, \qquad RadiusBottom = \frac{DiameterBottom}{2}$$

$$R(v) = RadiusBottom + v\frac{RadiusTop - RadiusBottom}{Length}$$

$$DirY = Direction \times DirBeg$$

$$u' = u * scaleU$$

$$where\ u'\ is\ measured\ in\ radians$$

$$v' = \begin{cases} v * scaleV, & turnedV = false \\ Length - v * scaleV, & turnedV = true \end{cases}$$

$$u \in [DomainAngle_0/scaleU, DomainAngle_1/scaleU], \qquad v \in [0, Length/scaleV]$$

$$where\ DomainAngle\ in\ formula\ is\ measured\ in\ radians,$$
$$and\ DomainAngle\ in\ file\ is\ measured\ in\ the\ units\ specified\ in\ the\ element,$$
$$or\ in\ radians\ if\ no\ units\ are\ specified$$

Fields:

Field Name	Data Type	Description
Cone23Core/@turnedV	xs:boolean	This flag shows if the v direction of the cone must be inverted.
Cone23Core/@scaleU	DoublePositiveType	The scaling coefficient of the u direction of the parametric space.
Cone23Core/@scaleV	DoublePositiveType	The scaling coefficient of the v direction of the parametric space.
Cone23Core/DiameterBottom	xs:double	The diameter at the bottom.
Cone23Core/DiameterTop	xs:double	The diameter at the top.
Cone23Core/Length	xs:double	The cone height – a distance between the cone top and bottom.
Cone23Core/Axis/AxisPoint	PointType	The axis origin (the bottom center).
Cone23Core/Axis/Direction	UnitVectorType	The axis vector.
Cone23Core/Sweep/DirBeg	UnitVectorType	The start direction specifies the position of the cone seam (u=0). This vector must lie in a plane normal to the axis of the cone.
Cone23Core/Sweep/DomainAngle	AngleRangeType	The sweep angle from the start direction (the surface domain in the u direction).

Example:

```
<Cone23 id="132">
  <Cone23Core>
    <DiameterBottom>4.4</DiameterBottom>
    <DiameterTop>2.1</DiameterTop>
```



```
    <Length>7.1</Length>
    <Axis>
      <AxisPoint>10.0 0.0 0.0</AxisPoint>
      <Direction>0.0 0.0 1.0</Direction>
    </Axis>
    <Sweep>
      <DirBeg>1.0 0.0 0.0</DirBeg>
      <DomainAngle>1.3 1.9</DomainAngle>
    </Sweep>
  </Cone23Core>
</Cone23>
```

7.5.2.4.4 *Sphere Surface*

Sphere23 describes a sphere parametric surface as shown in Figure 119, Figure 120, and Figure 121.

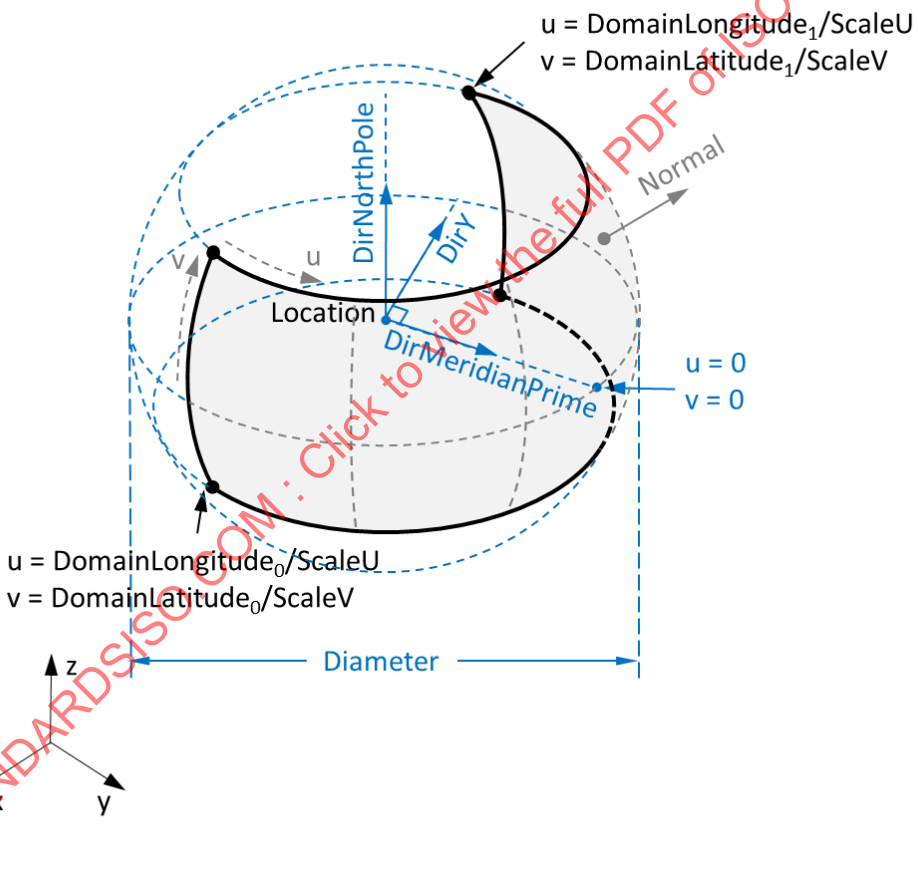


Figure 119 – Sphere

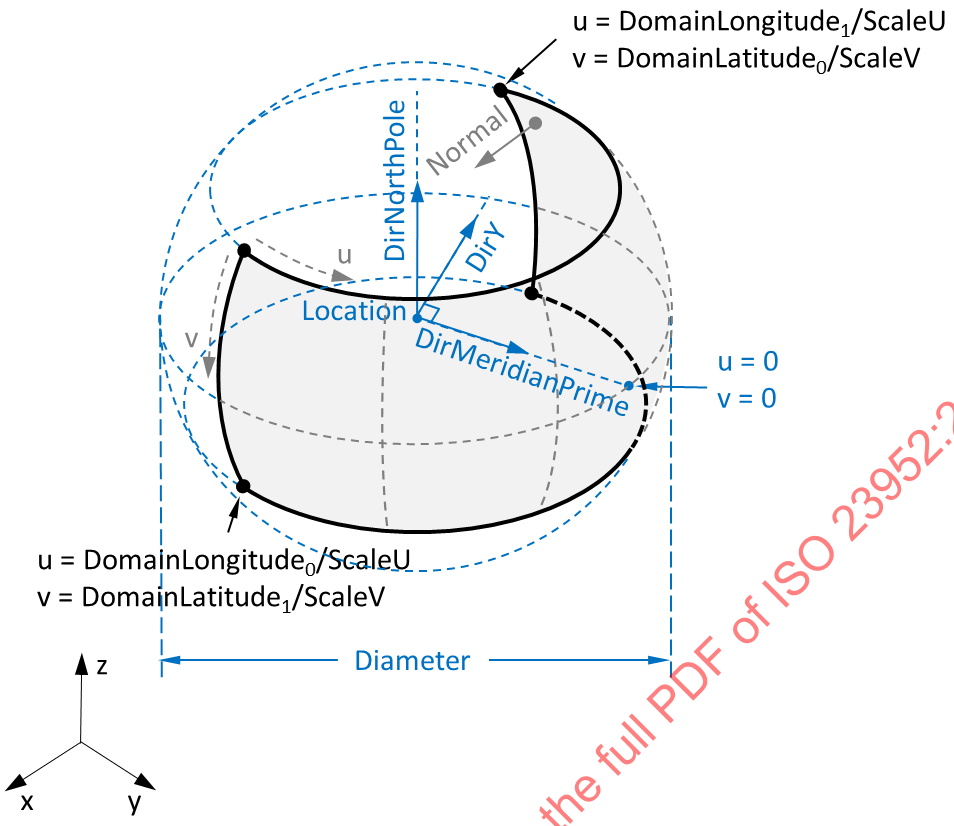


Figure 120 – Sphere (turnedV = true)

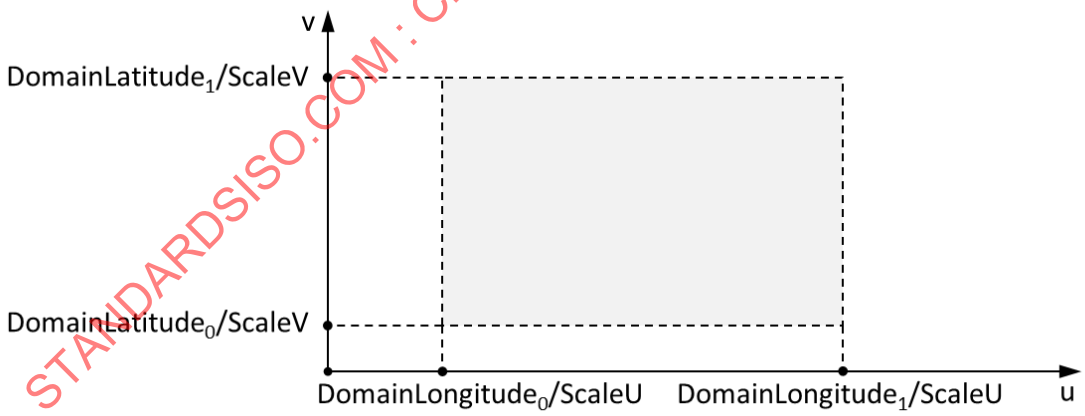


Figure 121 – Sphere (Parametric Space)

The sphere is a surface defined by its center point and radius.

Function:

$$Sphere23(u, v)\colon\ R_2 \rightarrow R_3$$

$$Sphere23(u,v) = Location + R\sin(v')\,DirNorthPole + R\cos(v')(\cos(u')DirMeridianPrime + \sin(u')DirY)$$

$$R = \frac{Diameter}{2}$$

$$DirY = DirNorthPole \times DirMeridianPrime$$

$$u' = u * scaleU$$

$$where\ u'\ is\ measured\ in\ radians$$

$$v' = \begin{cases} v * scaleV, & turnedV = false \\ -v * scaleV, & turnedV = true \end{cases}$$

$$where\ v'\ is\ measured\ in\ radians$$

$$u \in [DomainLongitude_0/scaleU, DomainLongitude_1/scaleU]$$

$$where\ DomainLongitude\ in\ formula\ is\ measured\ in\ radians,\ and\ DomainLongitude\ in\ file\ is\ measured\ in\ the\ units\ specified\ in\ the\ element,\ or\ in\ radians\ if\ no\ units\ are\ specified$$

$$v \in [DomainLatitude_0/scaleV, DomainLatitude_1/scaleV]$$

$$where\ DomainLatitude\ in\ formula\ is\ measured\ in\ radians,\ and\ DomainLatitude\ in\ file\ is\ measured\ in\ the\ units\ specified\ in\ the\ element,\ or\ in\ radians\ if\ no\ units\ are\ specified$$

Fields:

Field Name	Data Type	Description
Sphere23Core/@turnedV	xs:boolean	This flag shows if the v direction of the sphere must be inverted.
Sphere23Core/@scaleU	DoublePositiveType	The scaling coefficient of the u direction of the parametric space.
Sphere23Core/@scaleV	DoublePositiveType	The scaling coefficient of the v direction of the parametric space.
Sphere23Core/Diameter	xs:double	The sphere diameter.
Sphere23Core/Location	PointSimpleType	The sphere center.
Sphere23Core/LatitudeLongitudeSweep/DirMeridianPrime	UnitVectorType	The direction of the prime meridian vector. The longitude is 0 on

		the PrimeMeridianVector. This vector must be perpendicular to the north pole vector.
Sphere23Core/LatitudeLongitudeSweep/DomainLatitude	AngleRangeType	The latitude domain (the u direction). The latitude end angle must be greater than the latitude start angle.
Sphere23Core/LatitudeLongitudeSweep/DomainLongitude	AngleRangeType	The longitude domain (the u direction).
Sphere23Core/LatitudeLongitudeSweep/DirNorthPole	UnitVectorType	The direction of the north pole vector.

Example:

```
<Sphere23 id="132">
  <Sphere23Core>
    <Diameter>7.4</Diameter>
    <Location>2.0 3.0 2.3</Location>
    <LatitudeLongitudeSweep>
      <DirMeridianPrime>1.0 0.0 0.0</AxisPoint>
      <DomainLatitude>-0.1 0.4</DomainLatitude>
      <DomainLongitude>0.2 1.4</ DomainLongitude>
      <DirNorthPole>0.0 0.0 1.0</DirNorthPole>
    </LatitudeLongitudeSweep >
  </Sphere23Core>
</Sphere23>
```


7.5.2.4.5 *Torus Surface*

Torus23 describes a torus parametric surface as shown in Figure 122, Figure 123, and Figure 124.

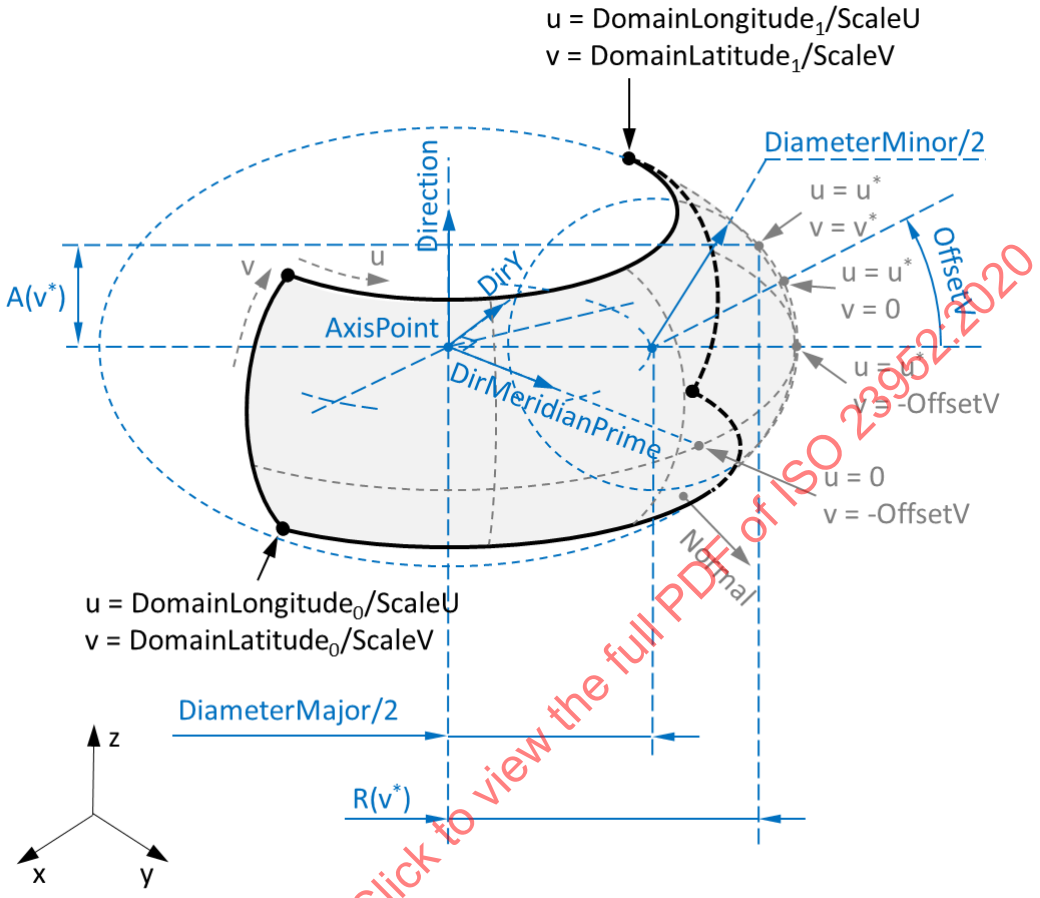


Figure 122 – Torus

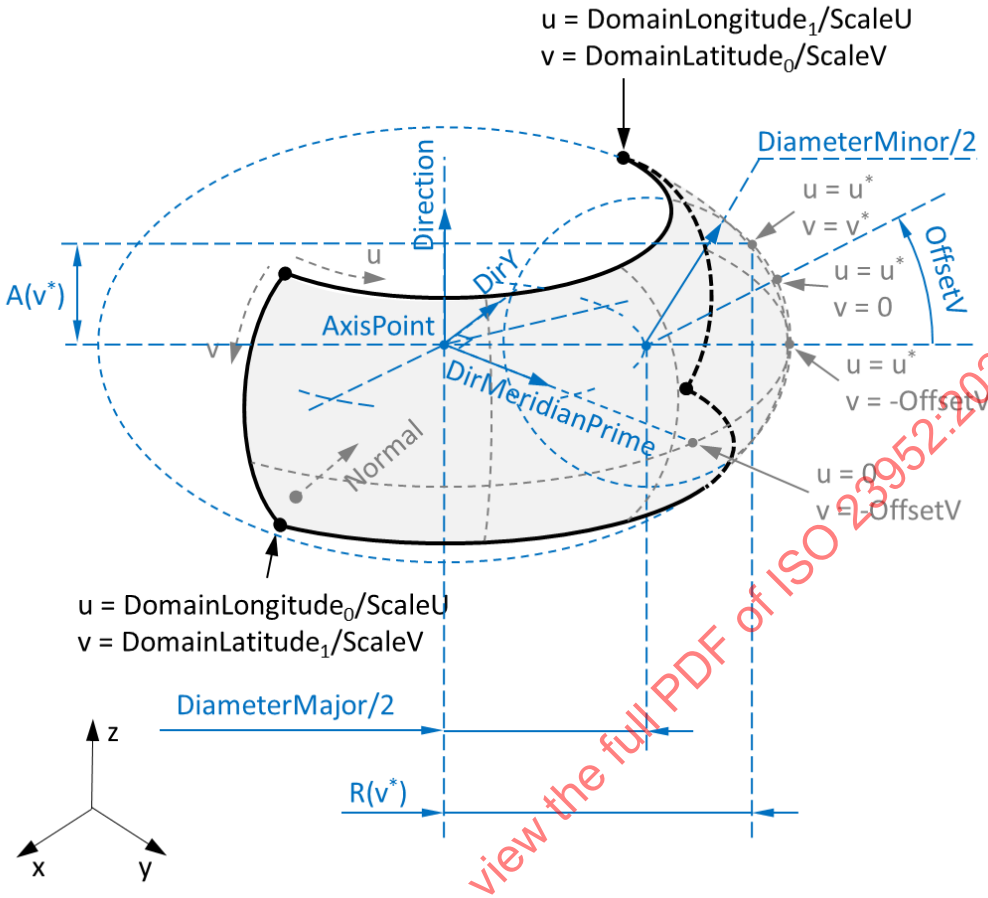


Figure 123 – Torus (turnedV = true)

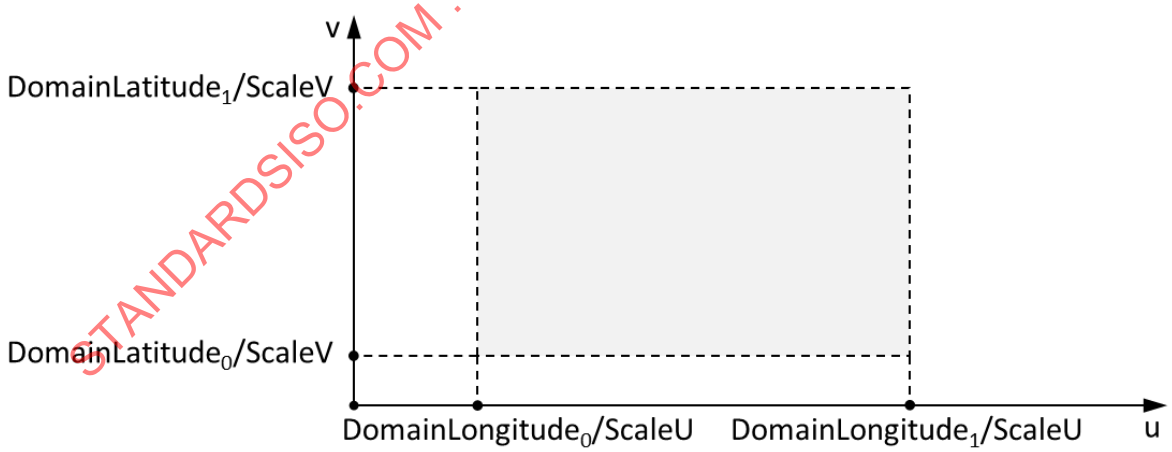


Figure 124 – Torus (Parametric Space)

The torus is a surface generated by rotating a 3D circular arc around an axis coplanar with the arc on a specified angle.

Function:

$$Torus23(u, v)\colon R_2 \rightarrow R_3$$

$$Torus23(u,v) = AxisPoint + A(v) + R(v)(\cos(u')DirMeridianPrime + \sin(u')\,DirY)$$

$$A(v) = RadiusMinor\,\sin(v')Direction$$

$$RadiusMinor = \frac{DiameterMinor}{2}, \qquad RadiusMajor = \frac{DiameterMajor}{2}$$

$$R(v) = RadiusMajor + RadiusMinor\,\cos(v')$$

$$DirY = Direction \times DirMeridianPrime$$

$$u' = u * scaleU$$

$$where\ u'\ is\ measured\ in\ radians$$

$$v' = \begin{cases} OffsetV + v * scaleV, & turnedV = false \\ OffsetV - v * scaleV, & turnedV = true \end{cases}$$

$$where\ v'\ is\ measured\ in\ radians$$

$$u \in [DomainLongitude_0/scaleU, DomainLongitude_1/scaleU]$$

$$where\ DomainLongitude\ in\ formula\ is\ measured\ in\ radians,$$
$$and\ DomainLongitude\ in\ file\ is\ measured\ in\ the\ units\ specified\ in\ the\ element,$$
$$or\ in\ radians\ if\ no\ units\ are\ specified$$

$$v \in [DomainLatitude_0/scaleV, DomainLatitude_1/scaleV]$$

$$where\ DomainLatitude\ in\ formula\ is\ measured\ in\ radians,$$
$$and\ DomainLatitude\ in\ file\ is\ measured\ in\ the\ units\ specified\ in\ the\ element,$$
$$or\ in\ radians\ if\ no\ units\ are\ specified$$

Fields:

Field Name	Data Type	Description
Torus23Core/@turnedV	xs:boolean	This flag shows if the v direction of the torus must be inverted.
Torus23Core/@offsetV	xs:double	The offsetting distance of the v direction of the parametric space.
Torus23Core/@scaleU	DoublePositiveType	The scaling coefficient of the u direction of the parametric space.
Torus23Core/@scaleV	DoublePositiveType	The scaling coefficient of the v direction of the parametric space.
Torus23Core/DiameterMinor	xs:double	The torus minor diameter.

Torus23Core/DiameterMajor	xs:double	The torus major diameter.
Torus23Core/Axis/AxisPoint	PointType	The axis origin (the bottom center).
Torus23Core/Axis/Direction	UnitVectorType	The axis vector.
Torus23Core/LatitudeLongitudeSweep/DirMeridianPrime	UnitVectorType	The direction of the prime meridian vector. The longitude is 0 on the PrimeMeridianVector. This vector must be perpendicular to the north pole vector.
Torus23Core/LatitudeLongitudeSweep/DomainLatitude	AngleRangeType	The latitude domain (the v direction).
Torus23Core/LatitudeLongitudeSweep/DomainLongitude	AngleRangeType	The longitude domain (the u direction). Regardless of the values of the longitude domain, the longitude sweep is in the positive direction.

Example:

```
<Torus23 id="344">
  <Torus23Core>
    <DiameterMinor>8.0</DiameterMinor>
    <DiameterMajor>24.0</DiameterMajor>
    <Axis>
      <AxisPoint>10.0 3.0 0.0</AxisPoint>
      <Direction>0.0 0.0 1.0</Direction>
    </Axis>
    <LatitudeLongitudeSweep>
      <DirMeridianPrime>1.0 0.0 0.0</DirMeridianPrime>
      <DomainLatitude>0.0 1.57</DomainLatitude>
      <DomainLongitude>0.0 3.14</DomainLongitude>
    </LatitudeLongitudeSweep>
  </Torus23Core>
</Torus23>
```


7.5.2.4.6 *Extrude Surface*

Extrude23 describes an extruded parametric surface as shown in Figure 125 and Figure 126.

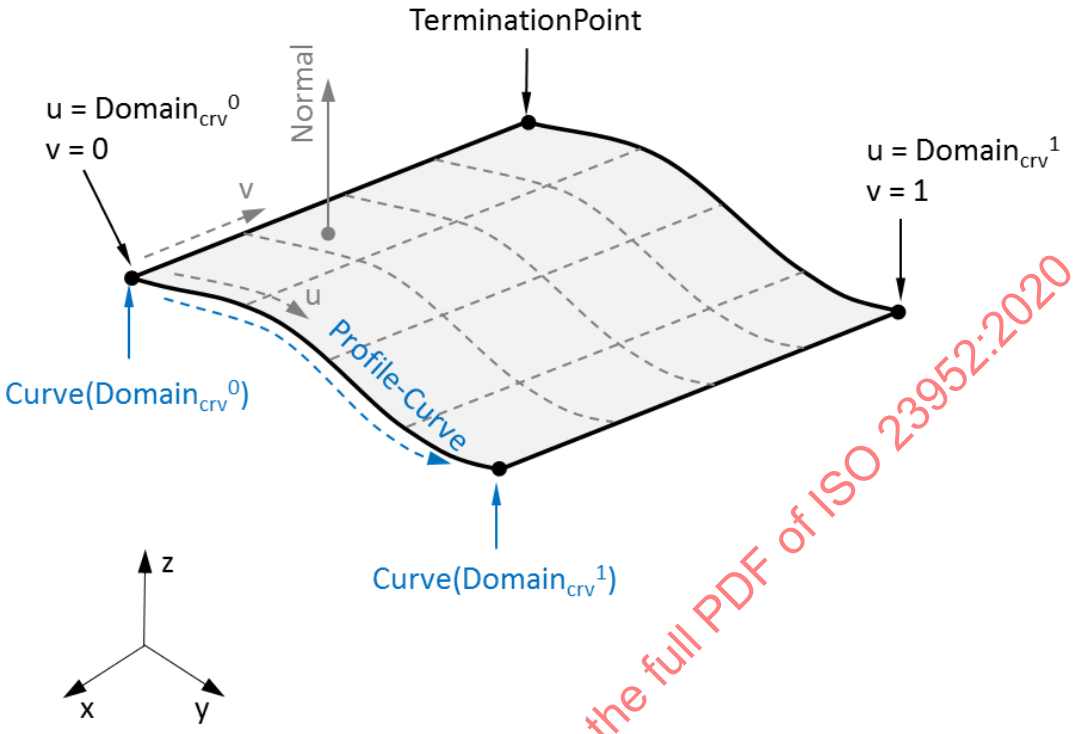


Figure 125 – Extrude Surface

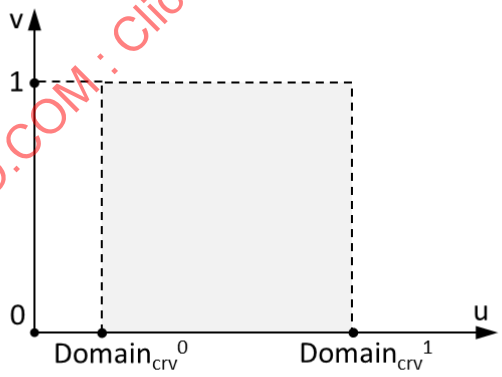


Figure 126 – Extrude Surface (Parametric Space)

The extrude surface is an extrusion of the profile-curve along the extrusion direction for a distance specified by the termination point.

Function:

$$Extrude23(u, v)\colon R_2 \rightarrow R_3$$

$$Extrude23(u, v) = Curve(u) + (TerminationPoint - Curve(domain_{crv}^{0}))v$$

$u \in [domain^0_{crv}, domain^1_{crv}], \quad v \in [0,1]$

Fields:

Field Name	Data Type	Description
Extrude23Core/TerminationPoint	PointSimpleType	The termination point. Together with the curve start point it specifies the extrusion direction and distance.
Extrude23Core/Curve	Curve13CoreType	The curve to be used as the profile of extrusion.

Example:

```
<Extrude23 id="357">
  <Extrude23Core >
    <TerminationPoint>20.4 11.2 -0.9</TerminationPoint>
    <Curve>
      <ArcCircular13Core domain="0 3.14159265358979">
        <Radius>0.25</Radius>
        <Center>20.0 11.0 0.0</Center>
        <DirBeg>1.0 0.0 0.0</DirBeg>
        <Normal>0.0 0.0 1.0</Normal>
      </ArcCircular13Core>
    </Curve>
  </Extrude23Core>
</Extrude23>
```


7.5.2.4.7 *Ruled Surface*

Ruled23 describes a ruled parametric surface as shown in Figure 127, Figure 128, and Figure 129.

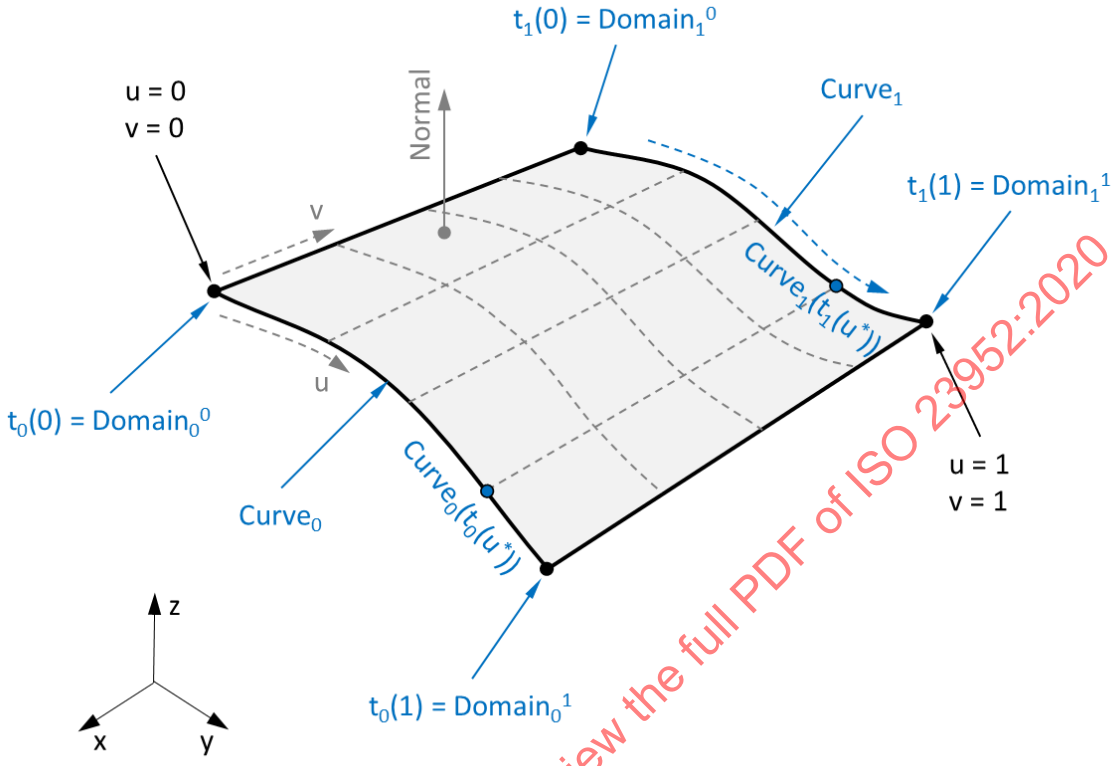


Figure 127 – Ruled Surface

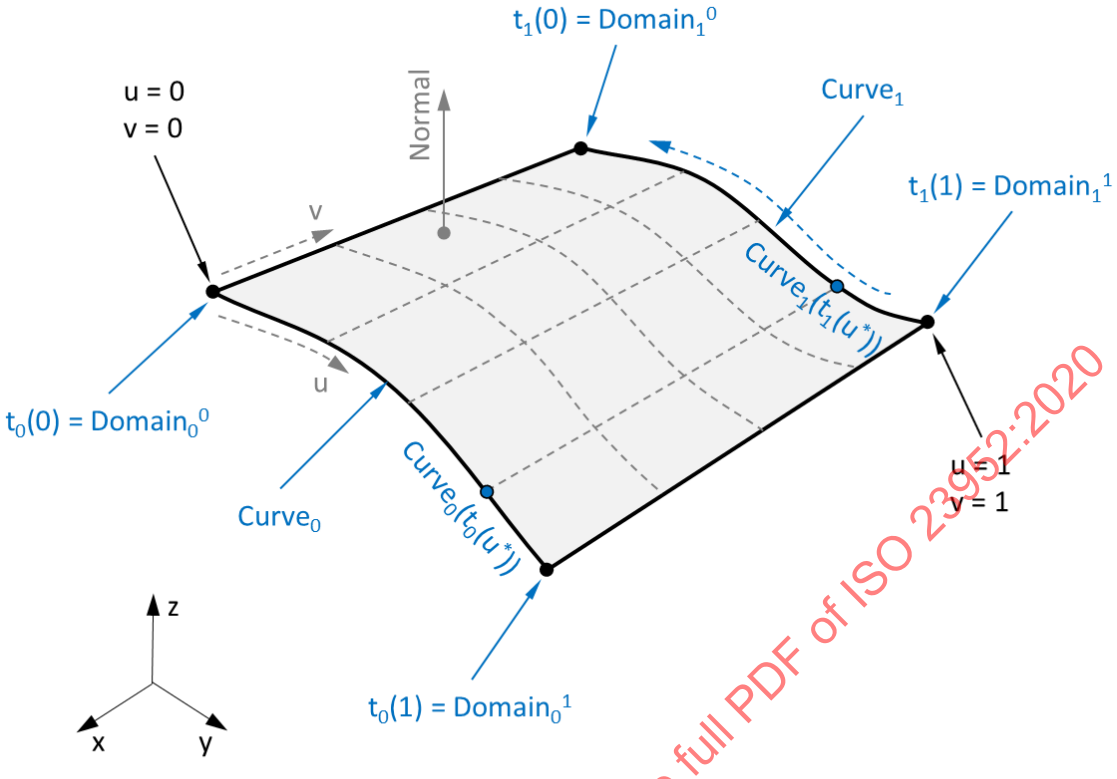


Figure 128 – Ruled Surface (turnedSecondCurve = true)

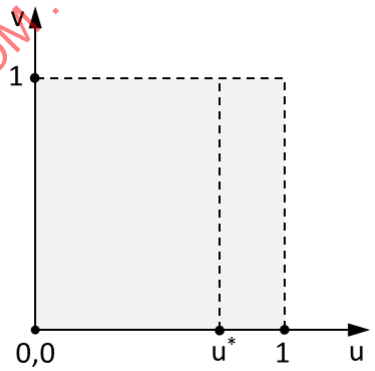


Figure 129 – Ruled Surface (Parametric Space)

The ruled surface is generated by connecting corresponding points on two 3D curves by a set of linear segments.

Function:

$$Ruled23(u,v)\colon R_2 \rightarrow R_3$$

$$Ruled23(u,v) = Curve_0(t_0(u))(1-v) + Curve_1(t_1(u))v$$

$domain_c - domain\ of\ curve\ c, \qquad c \in \{0,1\}$

$u \in [0,1], \qquad v \in [0,1]$

$t_0(u) = domain_0^0 + u(domain_0^1 - domain_0^0)$

$$t_1(u) = \begin{cases} domain_1^0 + u(domain_1^1 - domain_1^0), & turnedSecondCurve = false \\ domain_1^1 + u(domain_1^0 - domain_1^1), & turnedSecondCurve = true \end{cases}$$

Fields:

Field Name	Data Type	Description
Ruled23Core/@turnedSecondCurve	xs:boolean	This flag shows if the second curve is turned.
Ruled23Core/Curve[0]	Curve13CoreType	The first curve.
Ruled23Core/Curve[1]	Curve13CoreType	The second curve.

Example:

```
<Ruled23 id="361">
  <Ruled23Core>
    <Curve>
      <ArcCircular13Core domain="2.214 4.068">
        <Radius>0.625</Radius>
        <Center>14.8 11.0 0</Center>
        <DirBeg>1.0 0.0 0.0</DirBeg>
        <Normal>0.0 0.0 1.0</Normal>
      </ArcCircular13Core>
    </Curve>
    <Curve>
      <Segment13Core domain="0 1">
        <StartPoint>13.5 11.5 0.1</StartPoint>
        <EndPoint>13.5 10.5 0.3</EndPoint>
      </Segment13Core>
    </Curve>
  </Ruled23Core>
</Ruled23>
```

7.5.2.4.8 *Surface of Revolution*

Revolution23 describes a parametric surface of revolution as shown in Figure 130, Figure 131, and Figure 132.

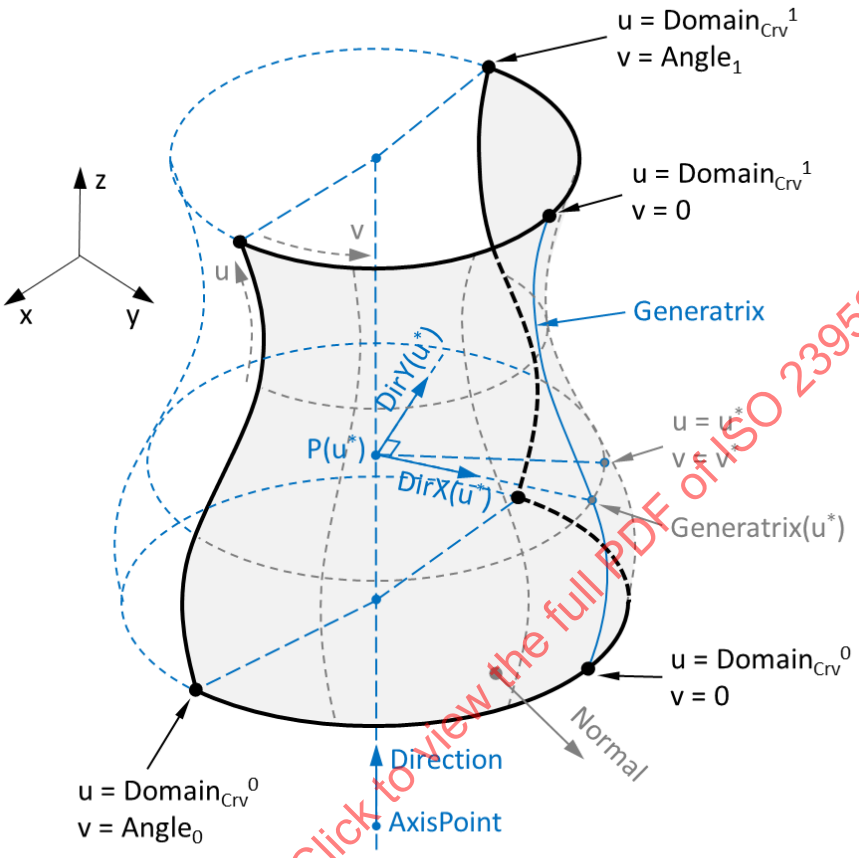


Figure 130 – Surface Of Revolution

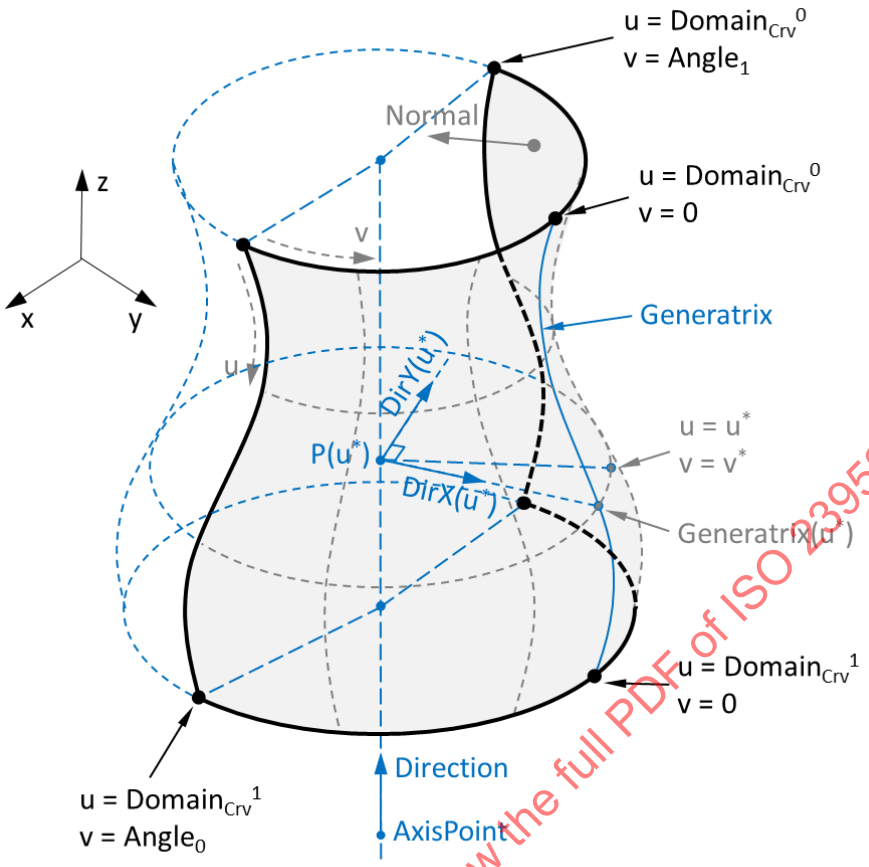


Figure 131 – Surface Of Revolution (turned Generatrix)

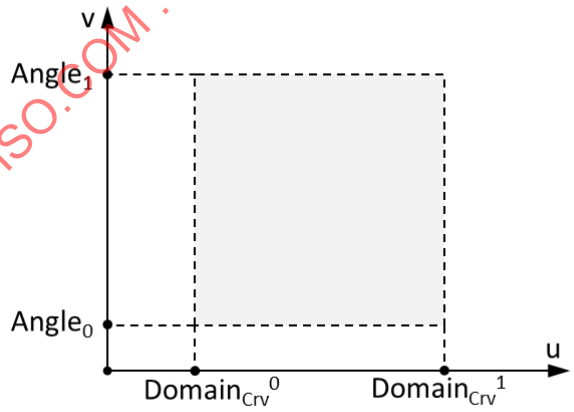


Figure 132 – Surface Of Revolution (Parametric Space)

The surface of revolution is generated by rotating a 3D generatrix curve around a specified axis on a specified angular range.

Function:

$$Revolution23(u, v)\colon\ R_2 \rightarrow R_3$$

$$Revolution23(u,v) = P(u) + R(u)\,(\cos(v)\,DirX(u) + \sin(v)\,DirY(u))$$

$$P(u) = AxisPoint + \big((Generatrix(u) - AxisPoint)\cdot Direction\big)Direction$$

$$R(u) = \|Generatrix(u) - P(u)\|$$

$$DirX(u) = \frac{Generatrix(u) - P(u)}{\|Generatrix(u) - P(u)\|}$$

$$DirY(u) = Direction \times DirX(u)$$

$$u \in [domain^0_{crv}, domain^1_{crv}], \qquad v \in [angle_0, angle_1]$$

Fields:

Field Name	Data Type	Description
Revolution23Core/@angle	ParameterRangeType	This field specifies start and end rotation angles.
Revolution23Core/Axis/AxisPoint	PointType	The axis origin.
Revolution23Core/Axis/Direction	UnitVectorType	The axis direction.
Revolution23Core/Generatrix	Curve13CoreType	The 3D curve to be rotated around the axis.

Example:

```
<Revolution23 id="354">
  <Revolution23Core angle="0 6.28318">
    <Axis>
      <AxisPoint>18.0 10.5 0</AxisPoint>
      <Direction>0.0 1.0 0.0</Direction>
    </Axis>
    <Generatrix>
      <ArcCircular13Core domain="1.965 4.317">
        <Radius>0.4</Radius>
        <Center>18.6 11.1 0.0</Center>
        <DirBeg>1.0 0.0 0.0</DirBeg>
        <Normal>0.0 0.0 1.0</Normal>
      </ArcCircular13Core>
    </Generatrix>
  </Revolution23Core>
</Revolution23>
```


7.5.2.4.9 *Spline Surface*

Spline23 describes a parametric spline surface as shown in Figure 133 and Figure 134.

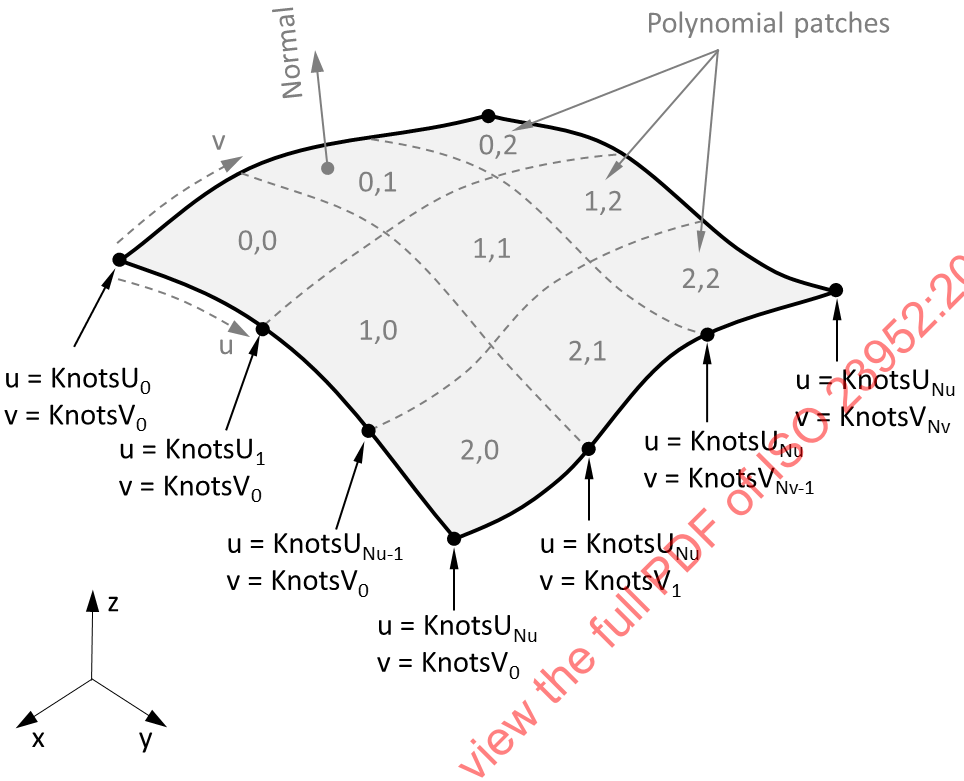


Figure 133 – Spline Surface

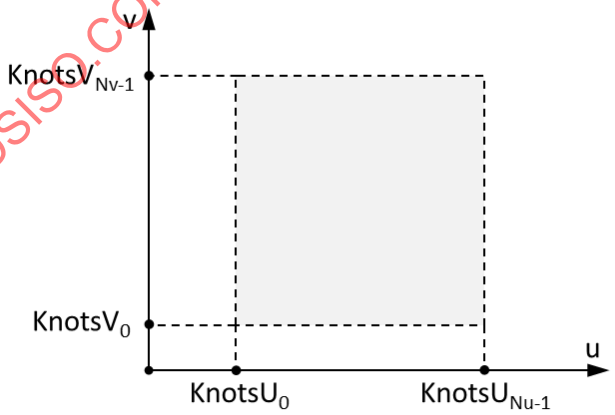


Figure 134 – Spline Surface (Parameter Space)

A spline surface is a grid of parametric polynomial patches.

Function:

$$Spline23(u, v)\colon R_2 \rightarrow R_3$$

$$Spline23(u,v) = \begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix} = \begin{matrix} \sum_{i=0}^{DegreeU_p} \sum_{j=0}^{DegreeV_q} (C_{i,j}^{p,q})_x (u_p)^i (v_q)^j \\ \sum_{i=0}^{DegreeU_p} \sum_{j=0}^{DegreeV_q} (C_{i,j}^{p,q})_y (u_p)^i (v_q)^j \\ \sum_{i=0}^{DegreeU_p} \sum_{j=0}^{DegreeV_q} (C_{i,j}^{p,q})_z (u_p)^i (v_q)^j \end{matrix}$$

$$Coefficients = \{\underbrace{C^{0,0} \dots C^{N_u-1,0}}_{N_u}, \dots, \underbrace{C^{0,N_v-1} \dots C^{N_u-1,N_v-1}}_{N_u}\}$$

$$DegreeU_p = OrdersU_p - 1$$

$$DegreeV_q = OrdersV_q - 1$$

$$NC - number\ of\ coefficients$$

$$NC^{p,q} - number\ of\ coefficients\ of\ polynomial\ patch\ (p,q)$$

$$NC^{p,q} = OrdersU_p OrdersV_q$$

$$NC = \sum_{p=0}^{N_u-1} \sum_{q=0}^{N_v-1} NC^{p,q}$$

$$C^{p,q} - coefficients\ of\ polynomial\ patch\ (p,q)$$

$$C^{p,q} = \{\underbrace{c_{0,0} \dots c_{DegreeU_p,0}}_{OrdersU_p}, \dots, \underbrace{c_{0,DegreeV_q} \dots c_{DegreeU_p,DegreeV_q}}_{OrdersU_p}\}$$

$$N_u, N_v - number\ of\ polynomials\ in\ the\ u\ and\ v\ directions$$

$$p \in [0, N_u], \qquad N_u = number\ of\ knots\ in\ the\ u\ direction - 1$$

$$q \in [0, N_v], \qquad N_v = number\ of\ knots\ in\ the\ v\ direction - 1$$

$$u_p - parameter\ of\ polynomials\ p\ in\ the\ u\ direction$$

$$u_p \in \begin{cases} [KnotsU_p, KnotsU_{p+1}], & normalized = false \\ [0,1], & normalized = true \end{cases}$$

$$u_p = \begin{cases} u - KnotsU_p, & normalized = false \\ (u - KnotsU_p)/(KnotsU_{p+1} - KnotsU_p), & normalized = true \end{cases}$$

$$v_q - parameter\ of\ polynomials\ p\ in\ the\ v\ direction$$

$$v_q \in \begin{cases} [KnotsV_q, KnotsV_{q+1}], & normalized = false \\ [0,1], & normalized = true \end{cases}$$

$$v_q = \begin{cases} v - KnotsV_q, & normalized = false \\ (v - KnotsV_q)/(KnotsV_{q+1} - KnotsV_q), & normalized = true \end{cases}$$

$$u \in [KnotsU_i, KnotsU_{i+1}], \qquad i \in [0, N_u - 1]$$

$$v \in [KnotsV_j, KnotsV_{j+1}], \qquad j \in [0, N_v - 1]$$

Fields:

Field Name	Data Type	Description
Spline13Core/@normalized	xs:boolean	This flag shows if the spline surface is normalized. A value of 1 (or true) means the surface is normalized. A value of 0 (or false) means the surface is not normalized.
Spline23Core/KnotsU	ArrayDoubleType	The knot vector in the u direction (the u spline breakpoints).
Spline23Core/KnotsV	ArrayDoubleType	The knot vector in the v direction (the v spline breakpoints).
Spline23Core/OrdersV	ArrayNaturalType	The orders of the polynomial patches in the u direction. The order is 'the degree of the polynomial' + 1. The size of this array is 'the number of the u spline breakpoints' - 1.
Spline23Core/OrdersV	ArrayNaturalType	The orders of the polynomial patches in the v direction. The order is 'the degree of the polynomial' + 1. The size of this array is 'the number of the v spline breakpoints' - 1.
Spline13Core/Coefficients	ArrayPointType	The coefficients of the polynomial patches. For each patch the number of coefficients equals 'the u polynomial order of the patch' * 'the v polynomial order of the patch'. The total size of this array is the sum of all patch coefficients.

Example:

```
<Spline23 id="101">
  <Spline23Core>
    <KnotsU n="3">
      0 1 2
    </KnotsU>
    <KnotsV n="2">
      0 1
    </KnotsV>
    <OrdersU n="2">4 4</OrdersU>
    <OrdersV n="2">4 4</OrdersV>
    <Coefficients n="32">
      -29.7 -20 13
      47.1 18 33
      -21.6 -18 -90
      4.45 6.75 49
      -15.9 60 -48
```

```
      2.7 -54 0
      -2.7 54 132.3
      7.65 -20.25 -83.7
      30.9 -60 12
      6.3 54 117
      -13.05 -54 -314.1
      0.6 20.25 147.9
      -20.3 34 -5
      -14.1 -60 -129
      19.35 60 241.8
      -6.2 -20.75 -100.45
      0.25 -13.25 5
      17.25 2.25 0
      -8.25 2.25 57
      25.75 -6.25 -49
      -8.25 39.75 0.6
      20.25 -6.75 13.5
      20.25 -6.75 -118.8
      -53.25 18.75 77.7
      24.75 -39.75 -37.2
      -18 6.75 -67.5
      -11.25 6.75 129.6
      45.3 -18.75 -54.9
      -21.25 13.25 7.35
      6 -2.25 53.25
      0.75 -2.25 -59.55
      -8.3 16.25 14.95
    </Coefficients>
  </Spline23Core>
</Spline23>
```


7.5.2.4.10 *NURBS Surface*

Nurbs23 describes a NURBS parametric surface as shown in Figure 135 and Figure 136.

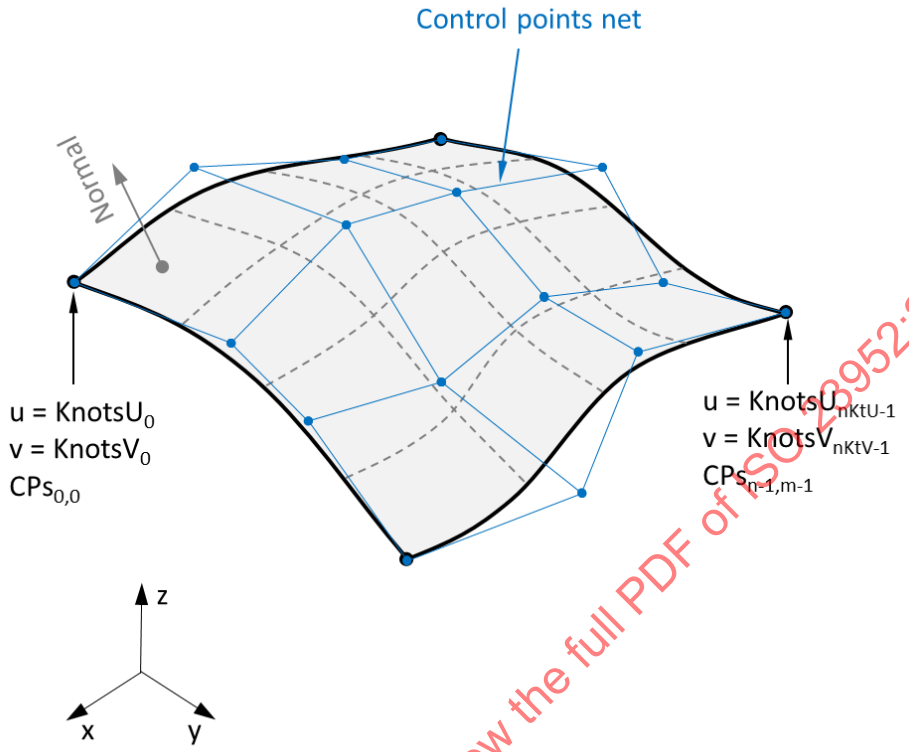


Figure 135 – NURBS Surface

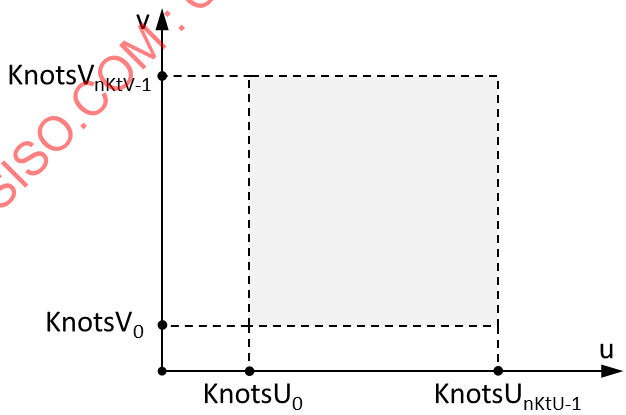


Figure 136 – NURBS Surface (Parameter Space)

A NURBS surface is a bivariate vector-valued piecewise rational function built on the B-spline basis functions and defined by its orders in the u and v directions, two knot vectors (an increasing sequence of real numbers which divides the parametric space in the intervals called knot spans), and a bidirectional control net with an optional set of associated weights (positive real numbers). If the weights are not defined or if the weights are equal, the surface is polynomial (otherwise rational).

Function:

$$Nurbs23(u,v)\colon R_2 \rightarrow R_3$$

$$Nurbs23(u,v) = \frac{\sum_{i=0}^{n-1}\sum_{j=0}^{m-1} N_{i,DegreeU}(u)N_{j,DegreeV}(v)Weights_{i,j}CPs_{ij}}{\sum_{i=0}^{n-1}\sum_{j=0}^{m-1} N_{i,DegreeU}(u)N_{j,DegreeV}(v)Weights_{i,j}}$$

$$DegreeU = OrderU - 1, \qquad DegreeV = OrderV - 1$$

$$N_{i,j}(t) - the\ bspline\ basis\ functions$$

$$N_{i,0}(t) = \begin{cases} 1, & t \in [Knots_i, Knots_{i+1}) \\ 0, & otherwise \end{cases}$$

$$N_{i,p}(t) = \frac{t - Knots_i}{Knots_{i+p} - Knots_i} N_{i,p-1}(t) + \frac{Knots_{i+p+1} - t}{Knots_{i+p+1} - Knots_{i+1}} N_{i+1,p-1}(t)$$

$$u \in [KnotsU_0, KnotsU_{nKtU-1}], \qquad n - number\ of\ control\ points\ in\ the\ u\ direction$$

$$v \in [KnotsV_0, KnotsV_{nKtV-1}], \qquad m - number\ of\ control\ points\ in\ the\ v\ direction$$

$$nKtU = n + OrderU, \qquad nKtV = m + OrderV$$

$$CPs = \{\underbrace{CPs_{0,0}, \dots, CPs_{n-1,0}}_{n}, \dots, \underbrace{CPs_{0,m-1}, \dots, CPs_{n-1,m-1}}_{n}\}$$

$$Weights = \{\underbrace{Weights_{0,0}, \dots, Weights_{n-1,0}}_{n}, \dots, \underbrace{Weights_{0,m-1}, \dots, Weights_{n-1,m-1}}_{n}\}$$

Fields:

Field Name	Data Type	Description
Nurbs23Core/OrderU	NaturalType	The order in the u direction (= degree + 1).
Nurbs23Core/OrderV	NaturalType	The order in the v direction (= degree + 1).
Nurbs23Core/KnotsU	ArrayDoubleType	The knot vector in the u direction. The size of the knot vector is 'number of control points in the u direction' + 'order in the u direction'.
Nurbs23Core/KnotsV	ArrayDoubleType	The knot vector in the v direction. The size of the knot vector is 'number of control points in the v direction' + 'order in the v direction'.
Nurbs23Core/CPs or Nurbs23Core/CPsBinary	ArrayPointType or ArrayBinaryType	The array of control points. The size of this array is ('number of the u-knots' - 'the u-order') * ('number of the v-knots' - 'the v-order').
Nurbs23Core/Weights	ArrayDoubleType	The array of weights associated with control points (positive real numbers).

Example:

```
<Nurbs23 id="102">
  <Nurbs23Core>
    <OrderU>4</OrderU>
    <OrderV>4</OrderV>
    <KnotsU n="8">
```



```
      0 0 0 0 1 1 1 1
    </KnotsU>
    <KnotsV n="8">
      0 0 0 0 1 1 1 1
    </KnotsV>
    <CPs n="16">
      -1.165 -0.787 0.511
      -0.551 -0.551 0.944
      -0.216 -0.551 0.196
      0.009 -0.521 0.196
      -1.377 0 -0.118
      -0.748 0 0.314
      -0.413 0 0.145
      -0.098 0 0.204
      -1.181 0 -0.590
      -0.511 0 0.354
      -0.206 0 -0.098
      0.118 0 -0.275
      -1.377 0.551 -1.102
      -0.826 0 -0.826
      -0.511 0 -0.944
      -0.177 0 -0.954
    </CPs>
  </Nurbs23Core>
</Nurbs23>
```

7.5.2.4.11 *Offset Surface*

Offset23 describes an offset parametric surface as shown in Figure 137 and Figure 138.

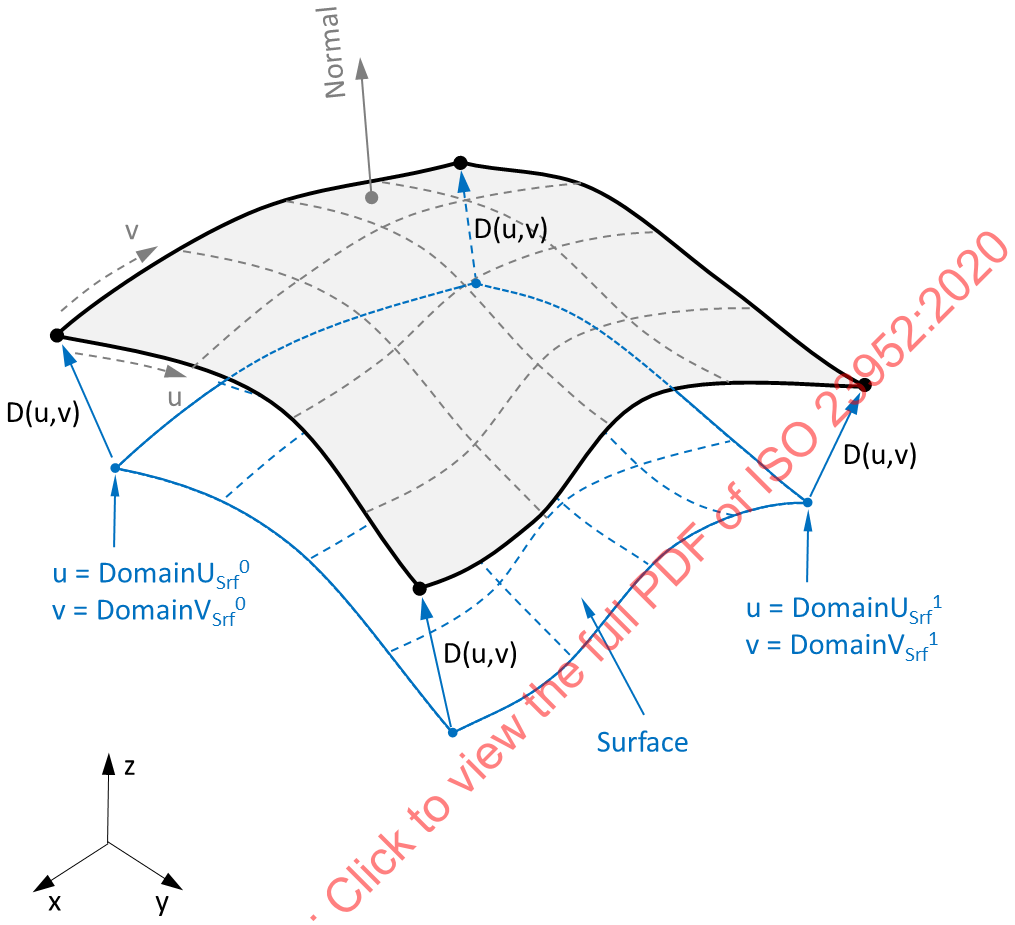


Figure 137 – Offset Surface

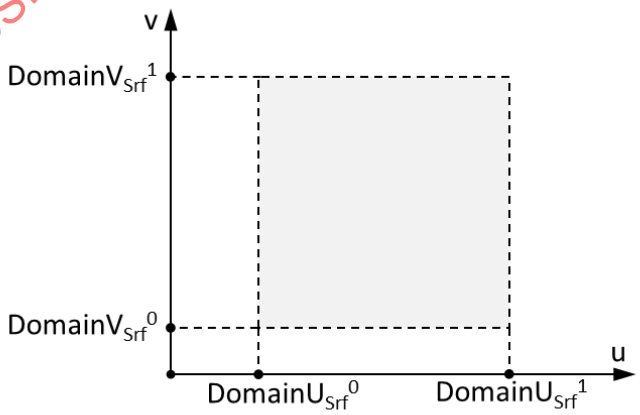


Figure 138 – Offset Surface (Parametric Space)

The offset surface is constructed from an existing surface by moving every surface point in the normal direction by an offset distance.

Function:

$$Offset23(u,v)\colon R_2 \rightarrow R_3$$

$$Offset23(u,v) = Surface(u,v) + D(u,v)$$

$$D(u,v) = N_{srf}(u,v)Distance$$

$$N_{srf}(u,v) = \frac{Surface'_u \times Surface'_v}{\|Surface'_u \times Surface'_v\|}$$

$$u \in \left[domainU^0_{srf}, domainU^1_{srf}\right], \qquad v \in \left[domainV^0_{srf}, domainV^1_{srf}\right]$$

Fields:

Field Name	Data Type	Description
Offset23Core/Distance	xs:double	The offset distance.
Offset23Core/Surface	SurfaceCoreType	The base surface for offsetting.

Example:

```
<Offset23 id="97">
  <Offset23Core>
    <Distance>0.2</Distance>
    <Surface>
      <Nurbs23Core>
        <OrderU>4</OrderU>
        <OrderV>4</OrderV>
        <KnotsU n="8">
          0 0 0 0 1 1 1 1
        </KnotsU>
        <KnotsV n="8">
          0 0 0 0 1 1 1 1
        </KnotsV>
        <CPs n="16">
          -1.165 -0.787 0.511
          -0.551 -0.551 0.944
          -0.216 -0.551 0.196
          0.009 -0.521 0.196
          -1.377 0 -0.118
          -0.748 0 0.314
          -0.413 0 0.145
          -0.098 0 0.204
          -1.181 0 -0.590
          -0.511 0 0.354
          -0.206 0 -0.098
          0.118 0 -0.275
          -1.377 0.551 -1.102
          -0.826 0 -0.826
          -0.511 0 -0.944
          -0.177 0 -0.954
```

```
        </CPs>
      </Nurbs23Core>
    </Surface>
  </Offset23Core>
</Offset23>
```


7.5.2.5 Mesh Curves

7.5.2.5.1 *Triangulation Path*

PathTriangulation describes a path of triangulation edges. The way the identification of triangle edges relate to the identification of triangle vertices is illustrated in Figure 139.

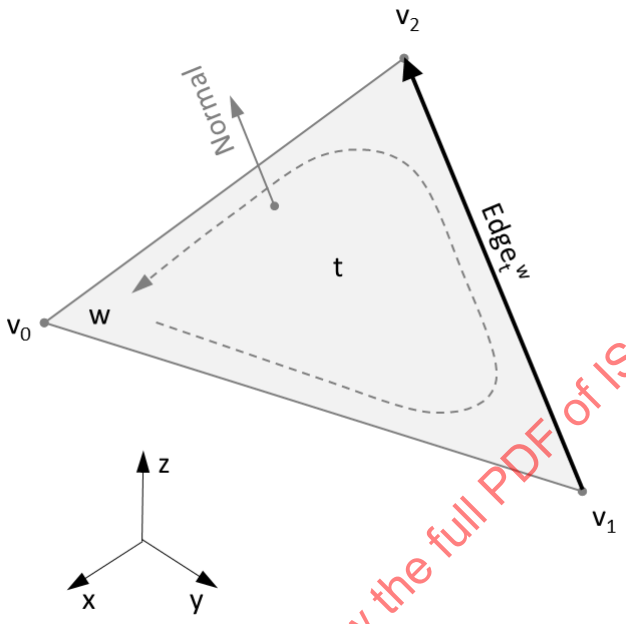


Figure 139 – The edge 'w' of triangle 't'

$$E_t^w - the\ edge\ 'w'\ of\ the\ triangle\ 't'\ is\ opposite\ to\ the\ vertex\ 'w'$$

$$E_t^w = \begin{cases} Start\ point = v_1 = Vertices_a, & a = Triangles_t^{(w+1)\ mod\ 3} \\ End\ point = v_2 = Vertices_b, & b = Triangles_t^{(w+2)\ mod\ 3} \end{cases}$$

$$Triangles_t^w = vertex\ w\ in\ triangle\ t$$

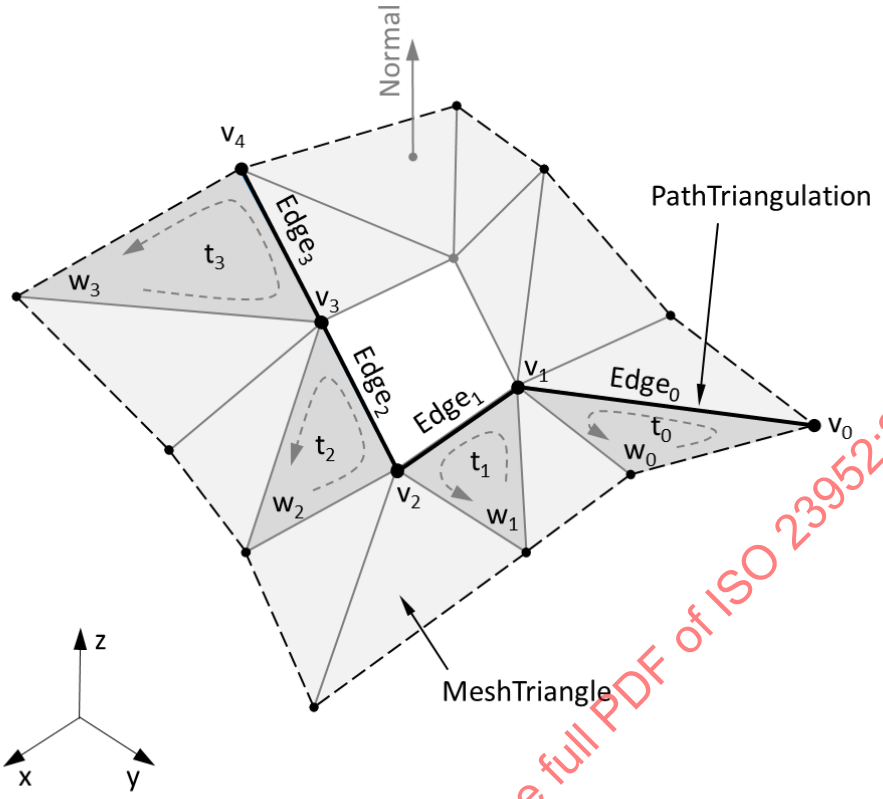


Figure 140 – Triangulation Path

$$Edges = \{\underbrace{t_0, w_0}_{Edge_0}, \underbrace{t_1, w_1}_{Edge_1}, \underbrace{t_2, w_2}_{Edge_2}, \underbrace{t_3, w_3}_{Edge_3}\}$$

$$V = \{v_0, v_1, v_2, v_3, v_4\}$$

The triangulation path is a polyline formed from the mesh edges as shown in Figure 140.

Function:

$$Edges = \{\underbrace{t_0, w_0}_{Edge_0}, \dots, \underbrace{t_{n-1}, w_{n-1}}_{Edge_{n-1}}\}, \qquad n - number\ of\ edges$$

$$Pairs\ (t_i, w_i)\ specifies\ edges\ E_{t_i}^{w_i}$$

$$V - sequence\ of\ vertices\ in\ the\ path$$

$$V = \{v_0, \dots, v_n\}$$

$$v_i = Vertices_j, \qquad j = \begin{cases} Triangles_0^{(w_0+1)\ mod\ 3} & i = 0 \\ Triangles_i^{(w_i+1)\ mod\ 3} = Triangles_{i-1}^{(w_{i-1}+2)\ mod\ 3}, & 0 < i < n \\ Triangles_{n-1}^{(w_{n-1}+2)\ mod\ 3}, & i = n \end{cases}$$

Fields:

Field Name	Data Type	Description
PathTriangulationCore/Edges or PathTriangulationCore/EdgesBinary	ArrayI2Type or ArrayBinaryType	The array of triangle edges that forms a triangulation path. This is an array of pairs of positive integers, where the first value is a triangle index and the second value is a vertex index opposite to the edge.
MeshTriangle	ElementReferenceType	The identifier of a triangle mesh.

Example:

```
<PathTriangulation id="32">
  <PathTriangulationCore>
    <Edges n="2">
      1 0
      0 2
    </Edges>
  </PathTriangulationCore>
  <MeshTriangle>
    <Id>98</Id>
  </MeshTriangle>
</PathTriangulation>

<MeshTriangle id="98">
  <MeshTriangleCore>
    <Triangles n="2">
      0 1 2
      2 3 0
    </Triangles>
    <Neighbours n="2">
      -1 1 -1
      -1 0 -1
    </Neighbours>
    <Vertices n="4">
      0.0 0.0 0.0
      10.0 0.0 0.0
      10.0 5.0 0.0
      0.0 5.0 0.0
    </Vertices>
  </MeshTriangleCore>
</MeshTriangle>
```


7.5.2.6 Mesh Surfaces

7.5.2.6.1 *Triangulation Mesh Surface*

MeshTriangle23 describes a triangulation mesh surface. Figure 141 illustrates how two triangles may be sewn together.

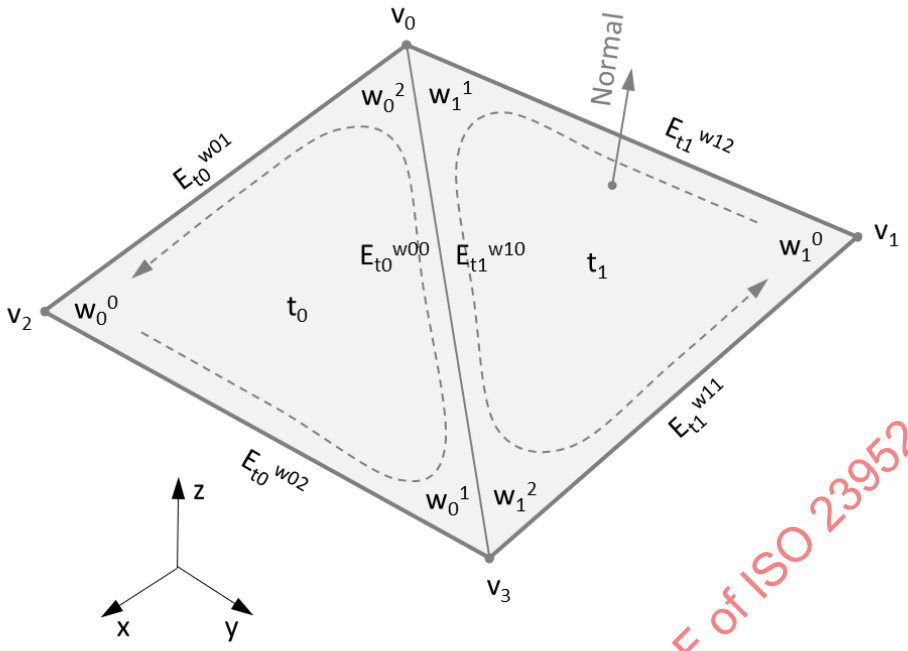


Figure 141 – Two sewn triangles

$$t_0 = Neighbours_{t_1}^{w_1^0}, \quad t_1 = Neighbours_{t_0}^{w_0^0}$$

$$w_t^i - index\ of\ vertex\ of\ triangle\ t$$

$$w_t^i \in \{0,1,2\}, \quad i \in \{0,1,2\}, \quad t - index\ of\ triangle$$

$$w_t^1 = (w_t^0 + 1)\ mod\ 3, \quad w_t^2 = (w_t^0 + 2)\ mod\ 3$$

$$\begin{cases} v_0 = Vertices_i = Vertices_j, & i = Triangles_{t_0}^{w_0^2}, j = Triangles_{t_1}^{w_1^1} \\ v_1 = Vertices_i, & i = Triangles_{t_1}^{w_1^0} \\ v_2 = Vertices_i, & i = Triangles_{t_0}^{w_0^0} \\ v_3 = Vertices_i = Vertices_j, & i = Triangles_{t_0}^{w_0^1}, j = Triangles_{t_1}^{w_1^2} \end{cases}$$

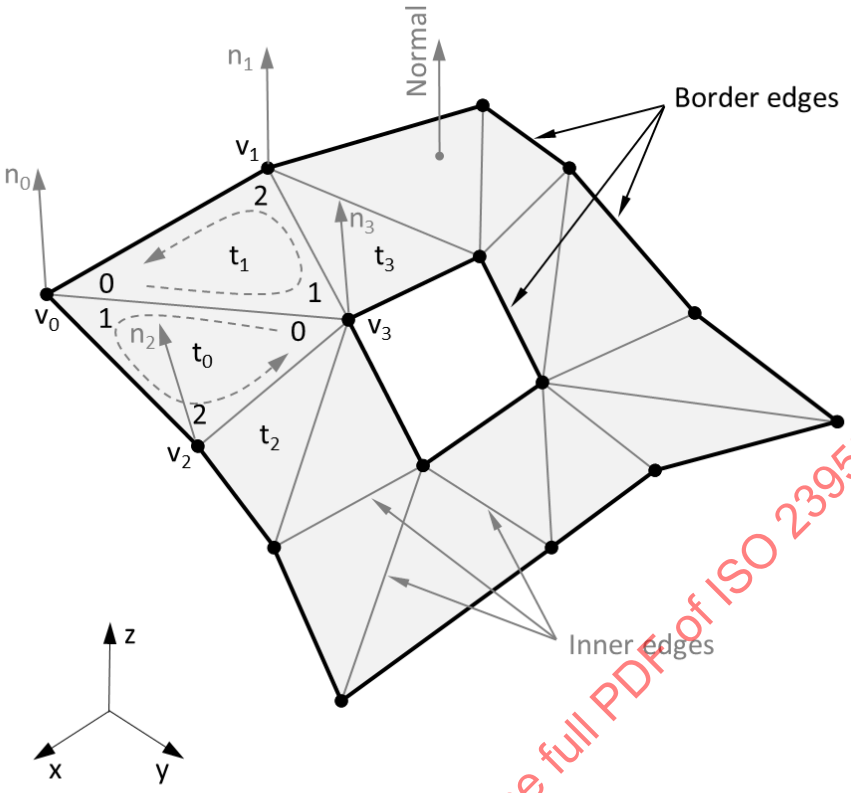


Figure 142 – Triangle Mesh

$$Triangles_{t_0} = \begin{bmatrix} v_3 \\ v_0 \\ v_2 \end{bmatrix}, \qquad Triangles_{t_1} = \begin{bmatrix} v_0 \\ v_3 \\ v_1 \end{bmatrix}$$

$$Neighbours_{t_0} = \begin{bmatrix} -1 \\ t_2 \\ t_1 \end{bmatrix}, \qquad Neighbours_{t_1} = \begin{bmatrix} t_3 \\ -1 \\ t_0 \end{bmatrix}$$

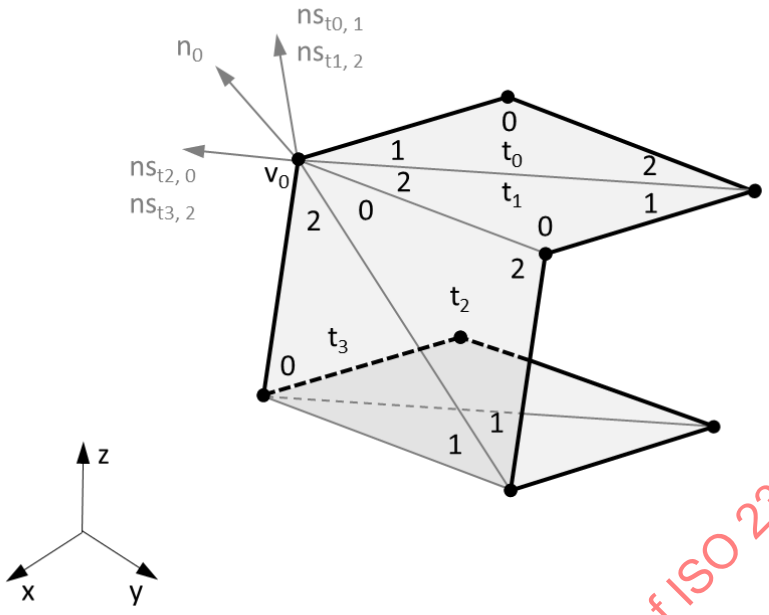


Figure 143 – Triangle Mesh with special normals

$$ns_{t,w} = special\ normal\ of\ vertex\ w\ in\ triangle\ t$$

$$t - index\ of\ triangle\ (first\ integer\ in\ NormalSpecial/@vertex)$$

$$w - vertex\ index\ in\ a\ triangle\ (second\ integer\ in\ NormalSpecial/@vertex)$$

A mesh surface is a set of triangles connected by their common edges as shown in Figure 142. QIF allows defining more than one normal vector per vertex; in this case, the special normals array shall be used as shown in Figure 143 – Triangle Mesh with special normals.

Function:

$$Triangles_t^w = the\ vertex_w\ in\ the\ triangle_t$$

$$E_t^w - the\ edge_w\ in\ the\ triangle_t\ is\ opposite\ to\ the\ vertex_w$$

$$E_t^w = \begin{cases} Start\ point = Vertices_a, & a = Triangles_t^{(w+1)\ mod\ 3} \\ End\ point = Vertices_b, & b = Triangles_t^{(w+2)\ mod\ 3} \end{cases}$$

$$Neighbours_t^w = \begin{cases} neighbour\ triangle\ index, & E_t^w\ is\ a\ border\ edge \\ -1, & otherwise \end{cases}$$

$$w - vertex\ index\ in\ a\ triangle, \qquad w \in \{0,1,2\}$$

$$N_t - unit\ normal\ of\ the\ triangle\ 't'$$

$$N_t = \frac{(Vertices_{v_1} - Vertices_{v_0}) \times (Vertices_{v_2} - Vertices_{v_0})}{\left\|(Vertices_{v_1} - Vertices_{v_0}) \times (Vertices_{v_2} - Vertices_{v_0})\right\|}$$

$$\begin{cases} v_i = Triangles_t^i \\ i \in \{0,1,2\} \end{cases}$$

Fields:

Field Name	Data Type	Description
MeshTriangleCore/Triangles or MeshTriangleCore/TrianglesBinary	ArrayI3Type or ArrayBinaryType	The array of indices of the triangle vertices. The number of array elements corresponds to the number of triangles in the mesh. Each element of this array is a triplet of integer numbers: index of the first vertex, index of the second vertex and index of the third vertex. All three vertex indices of a triangle must be different and must lie in the range [0, ..., number of vertices - 1].
MeshTriangleCore/Neighbours or MeshTriangleCore/NeighboursBinary	ArrayI3Type or ArrayBinaryType	The array of indices of the triangle neighbors. The number of array elements corresponds to the number of triangles in the mesh. Each element of this array is a triplet of integer numbers: index of a neighbor triangle opposite to the first triangle vertex, index of a neighbor triangle opposite to the second triangle vertex, index of a neighbor triangle opposite to the third triangle vertex. There is a special index value "-1" which shows that there is no neighbor. The neighbor indices must lie in the range [-1, ..., number of triangles - 1].
MeshTriangleCore/Vertices or MeshTriangleCore/VerticesBinary	ArrayPointType or ArrayBinaryType	The array of 3D points. The number of array elements corresponds to the number of vertices in the mesh. Each element of this array is a triplet of real numbers: the X-coordinate, the Y-coordinate and the Z-coordinate.
MeshTriangleCore/Normals or MeshTriangleCore/NormalsBinary	ArrayUnitVectorType or ArrayBinaryType	The mesh normals - an array of unit vectors. The number of array elements

		corresponds to the number of vertices in the mesh. Each element of this array is a triplet of real numbers: the X-component, the Y-component and the Z-component.
NormalsSpecial or NormalsSpecialBinary	ArrayTriangleVertexNormalType or ArrayBinaryType	The special mesh normal - an array of normal vectors specified on the triangle level. This array is an addition to the core mesh normal vectors and it allows for the definition of different normal vectors in a common vertex for different triangles. This situation is typical for vertices of mesh feature lines.
NormalsSpecial/NormalSpecial/@vertex	I2Type	The pair of non-negative integers, where the first value is a triangle index and the second value is a vertex index.

Example:

```
<MeshTriangle id="98">
  <MeshTriangleCore>
    <Triangles n="2">
      0 1 2
      2 3 0
    </Triangles>
    <Neighbours n="2">
      -1 1 -1
      -1 0 -1
    </Neighbours>
    <Vertices n="4">
      0.0 0.0 0.0
      10.0 0.0 0.0
      10.0 5.0 0.0
      0.0 5.0 0.0
    </Vertices>
    <Normals n="4">
      0 0 1
      0 0 1
      0 0 1
      0 0 1
    </Normals>
  </MeshTriangleCore>
  <NormalsSpecial n="2">
    <NormalSpecial vertex="0 0">
      0.5 0 0.8660254037844
    </NormalSpecial>
```

```
    <NormalSpecial vertex="1 2">
      0 0.5 0.8660254037844
    </NormalSpecial>
  </NormalsSpecial>
</MeshTriangle>
```

7.5.3 Topology

QIF topology elements define boundaries of geometry objects and specify connectivity relations. In QIF they are separated by type and organized in the following sets:

- VertexSet
- EdgeSet
- LoopSet
- FaceSet
- ShellSet
- BodySet
- PointCloudSet

As seen in Figure 144, all topology types are derived from TopologyBaseType.

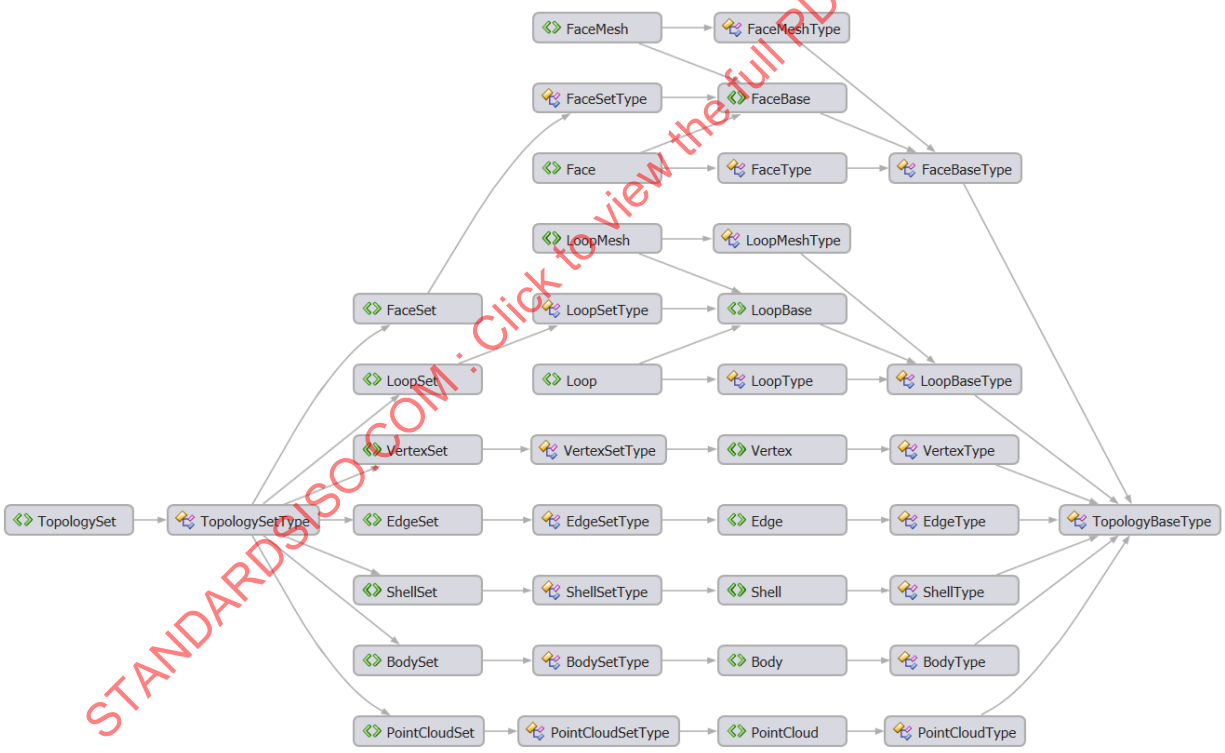


Figure 144 – Topology Types

The main topological items are:

- Body - a set of oriented shells built on collections of connected face elements
- Shell - a set of connected faces
- Face - a bounded portion of a surface
- Loop - a circuit of edges bounding a face
- Edge - a bounded piece of a 3D curve
- Vertex - lies at a point and can be used to bound an edge

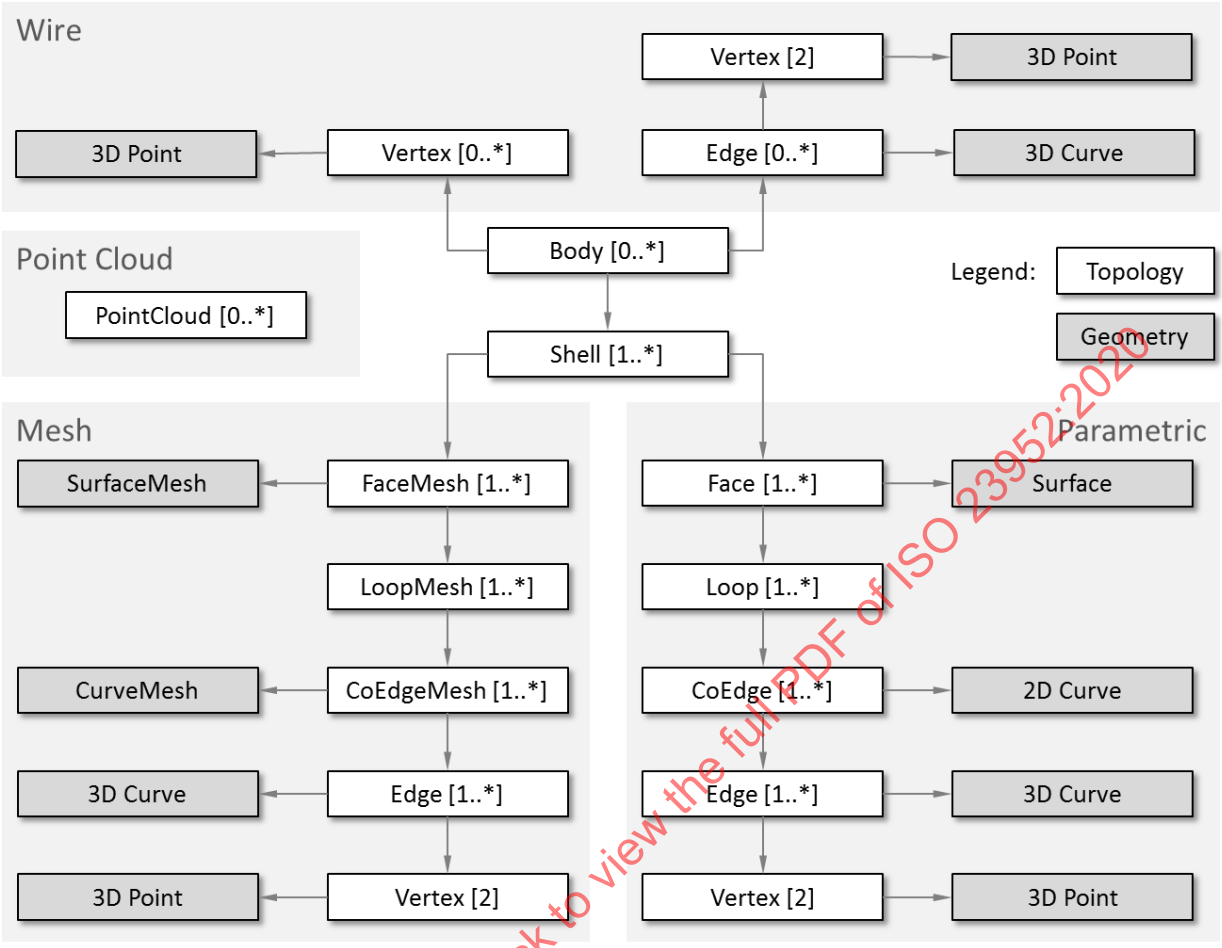


Figure 145 – Boundary Representation

All topology objects contain the following fields:

Field Name	Data Type	Description
@id	QIFIdType	The unique model entity identifier.
@label	xs:string	The model entity "nameplate". Normally it can be seen at the entity item in the model tree.
@color	ColorType	The RGB color property of a model entity.
@transparency	TransparencyType	The transparency property of a model entity.
@hidden	xs:boolean	The visibility property of a model entity in the graphical window.
@size	xs:double	A recommended size for visualization of an infinite drawable element.
Attributes	AttributesType	User-defined attributes (typified, binary array, or XML structured).

7.5.3.1 Vertex

Vertex describes a topological vertex as illustrated in Figure 146.

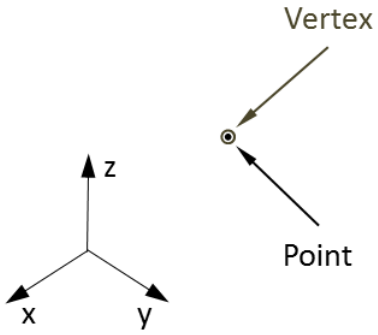


Figure 146 – Vertex

A vertex lies at a 3D point and is normally used to bound an edge.

Fields:

Field Name	Data Type	Description
@tolerance	xs:double	The tolerance value, which is calculated as the maximum distance from the vertex underlying 3D point to the ends of all neighboring edges that are terminated in the neighborhood of this vertex. This value can be defined only for the case of the tolerant body.
Point	ElementReferenceType	The identifier of a 3D point – the underlying geometry of the vertex.

Example:

```
<Vertex id="401">
  <Point>
    <Id>398</Id>
  </Point>
</Vertex>
```

7.5.3.2 Edge

Edge describes a topological edge as shown in Figure 147.

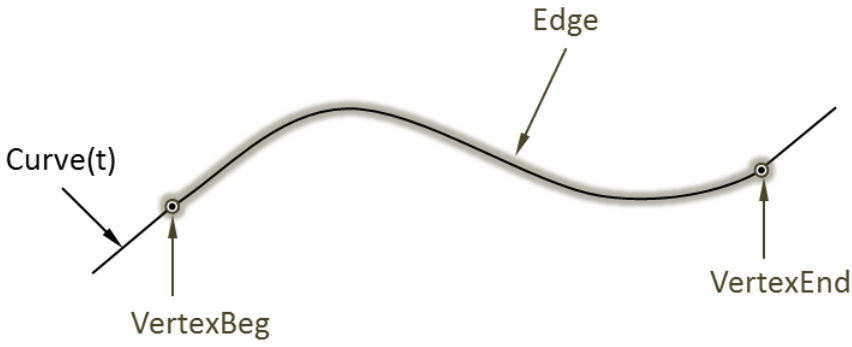


Figure 147 – Edge

An edge is a bounded piece of a 3D curve.

Fields:

Field Name	Data Type	Description
@tolerance	xs:double	The tolerance value calculated as the maximum distance between the 2D parametric co-edges of the neighboring faces. The edge tolerance for the case of tolerant body.
Validation	ValidationEdgeType	The set of edge validation properties.
Curve	ElementReferenceType	The identifier of a 3D curve that is the underlying geometry of this edge.
VertexBeg	ElementReferenceType	The identifier of the vertex, which bounds this edge at the beginning of the edge. The 'underlying' parameter of VertexBeg must be less than the 'underlying' parameter of VertexEnd. Or, in other words, the edge always follows the natural parameterization of the underlying 3D curve. If there is a need to pass an edge in the opposite (to the natural parameterization of the underlying curve) direction then the corresponding flag shall be defined at the loop level.
VertexEnd	ElementReferenceType	The identifier of the vertex, which bounds this edge at the end of the edge. The 'underlying' parameter of VertexEnd must be bigger than the 'underlying' parameter of VertexBeg. In other words, the edge always follows the natural parameterization of the underlying 3D curve. If there is a need to pass an edge in the opposite (to the natural parameterization of the underlying curve) direction then the corresponding flag shall be defined at the loop level.

Example:

```
<Edge id="405">
  <Curve>
    <Id>390</Id>
```



```
    </Curve>
    <VertexBeg>
      <Id>566</Id>
    </VertexBeg>
    <VertexEnd>
      <Id>401</Id>
    </VertexEnd>
  </Edge>
```

7.5.3.3 Loop

Loop describes a topological loop as shown in Figure 148.

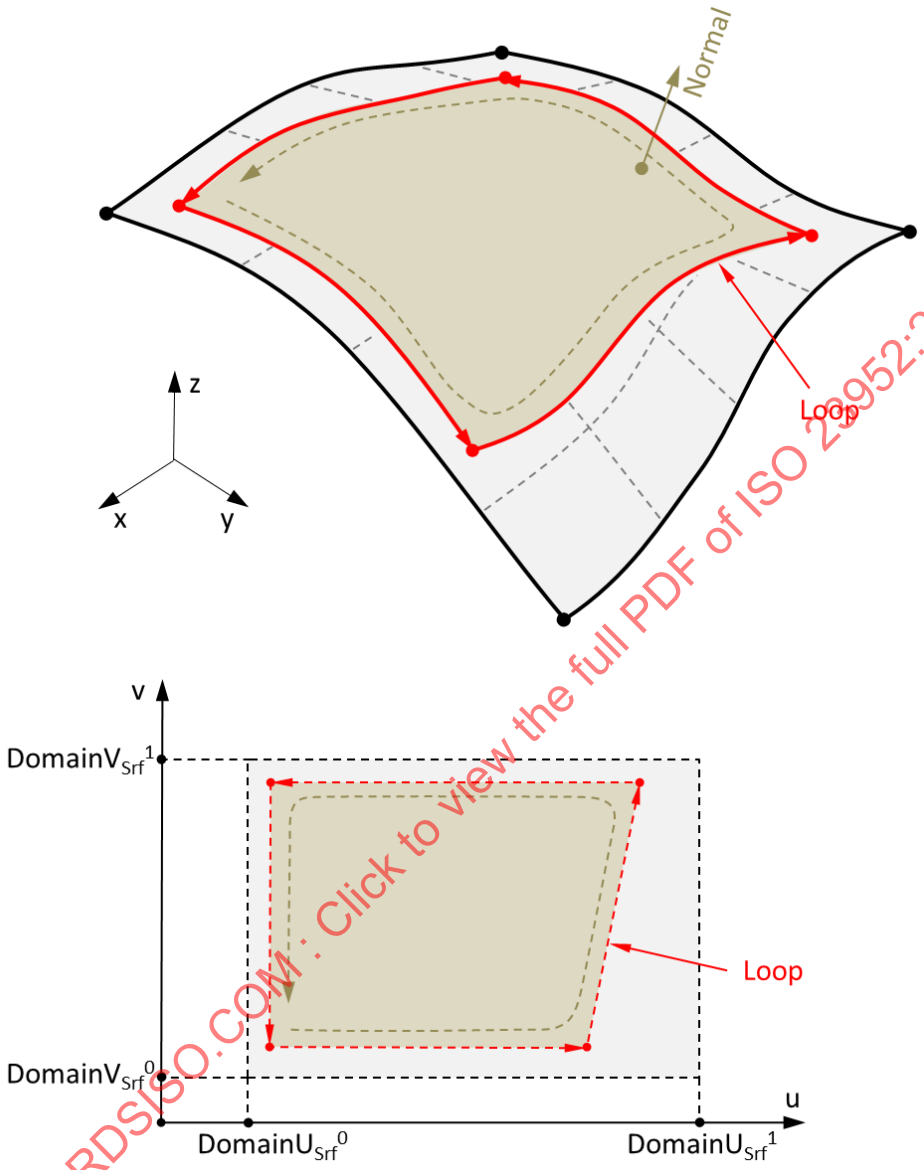


Figure 148 – Loop

A loop is a circuit of edges (in 3D space) with corresponding CoEdges (in surface parameter space) bounding a face. CoEdges are shown in Figure 149.

Fields:

Field Name	Data Type	Description
@form	LoopFormEnumType	The loop type that can take one of the following values: 'UNKNOWN', 'OUTER', 'INNER', 'SLIT' or 'VERTEX'.
CoEdges	CoEdgesType	An array of co-edges that forms the loop.

CoEdges/CoEdge/EdgeOriented/@turned	xs:boolean	This flag shows if the referenced edge shall be inverted.
CoEdges/CoEdge/EdgeOriented/Id	QIFReferenceType	A reference to an edge.
CoEdges/CoEdge/Curve12	ElementReferenceType	A reference to a 2D curve which is a projection of the edge underlying 3D Curve onto the face underlying surface.

Example:

```
<Loop id="113" form="OUTER">
  <CoEdges n="4">
    <CoEdge>
      <EdgeOriented>
        <Id>405</Id>
      </EdgeOriented>
      <Curve12>
        <Id>520</Id>
      </Curve12>
    </CoEdge>
    <CoEdge>
      <EdgeOriented>
        <Id>469</Id>
      </EdgeOriented>
      <Curve12>
        <Id>532</Id>
      </Curve12>
    </CoEdge>
    <CoEdge>
      <EdgeOriented turned="1">
        <Id>511</Id>
      </EdgeOriented>
      <Curve12>
        <Id>547</Id>
      </Curve12>
    </CoEdge>
    <CoEdge>
      <EdgeOriented>
        <Id>567</Id>
      </EdgeOriented>
      <Curve12>
        <Id>563</Id>
      </Curve12>
    </CoEdge>
  </CoEdges>
</Loop>
```


7.5.3.3.1 *Co-Edges*

A CoEdge defines a parameter space curve (i.e. the projection of an underlying 3D Curve of an edge onto an underlying surface of a face) that represents a part of the face trimming loop and additionally includes the corresponding edge orientation flag.

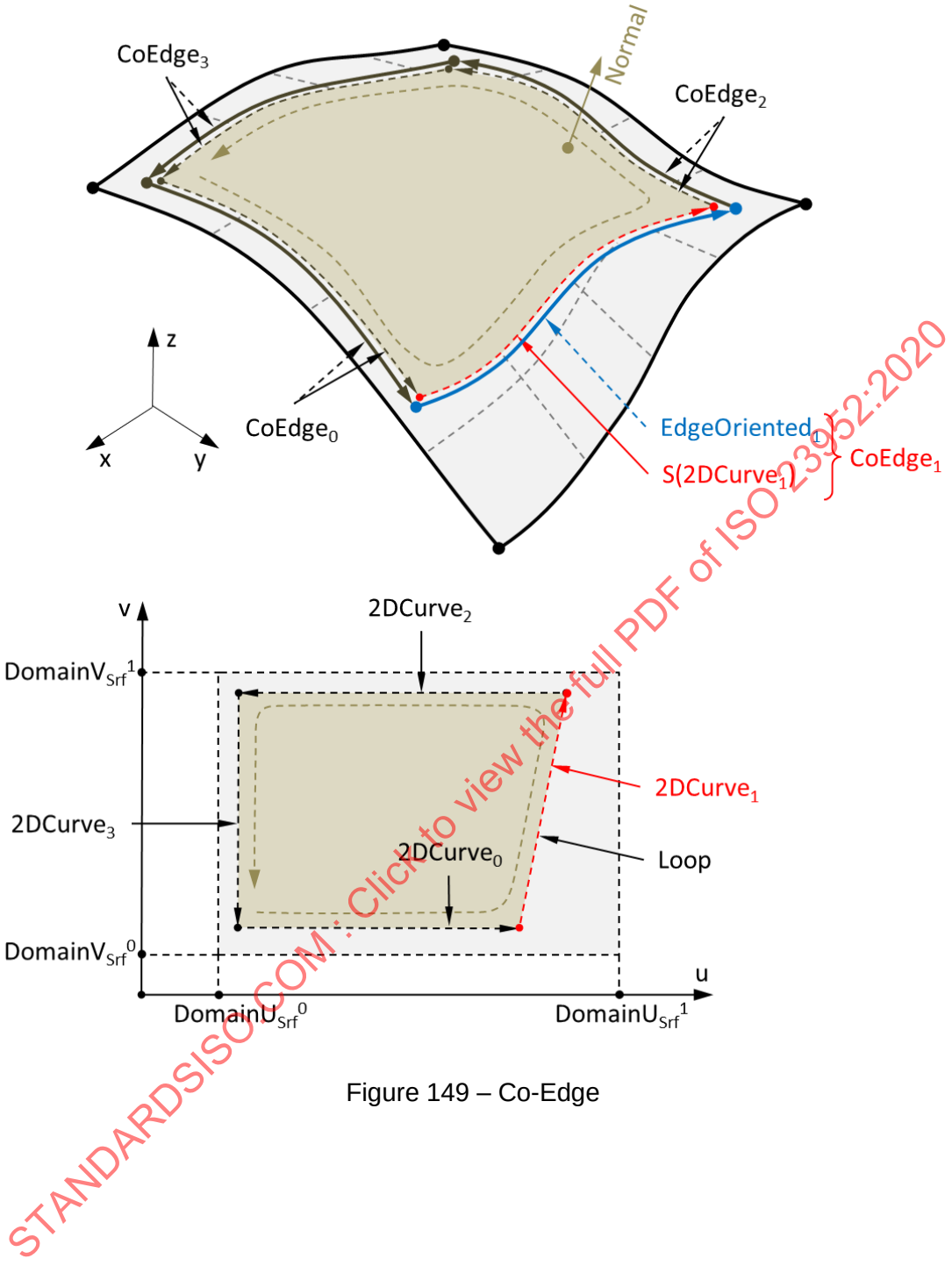


Figure 149 – Co-Edge

7.5.3.3.2 *Loop forms*

The QIF data model allows defining of the four loop forms: 'OUTER' (shown in Figure 150), 'INNER' (shown in Figure 151), ‘SLIT’ (shown in Figure 152) and ‘VERTEX’ (shown in Figure 153). In addition, the ‘UNKNOWN’ loop form may be used.

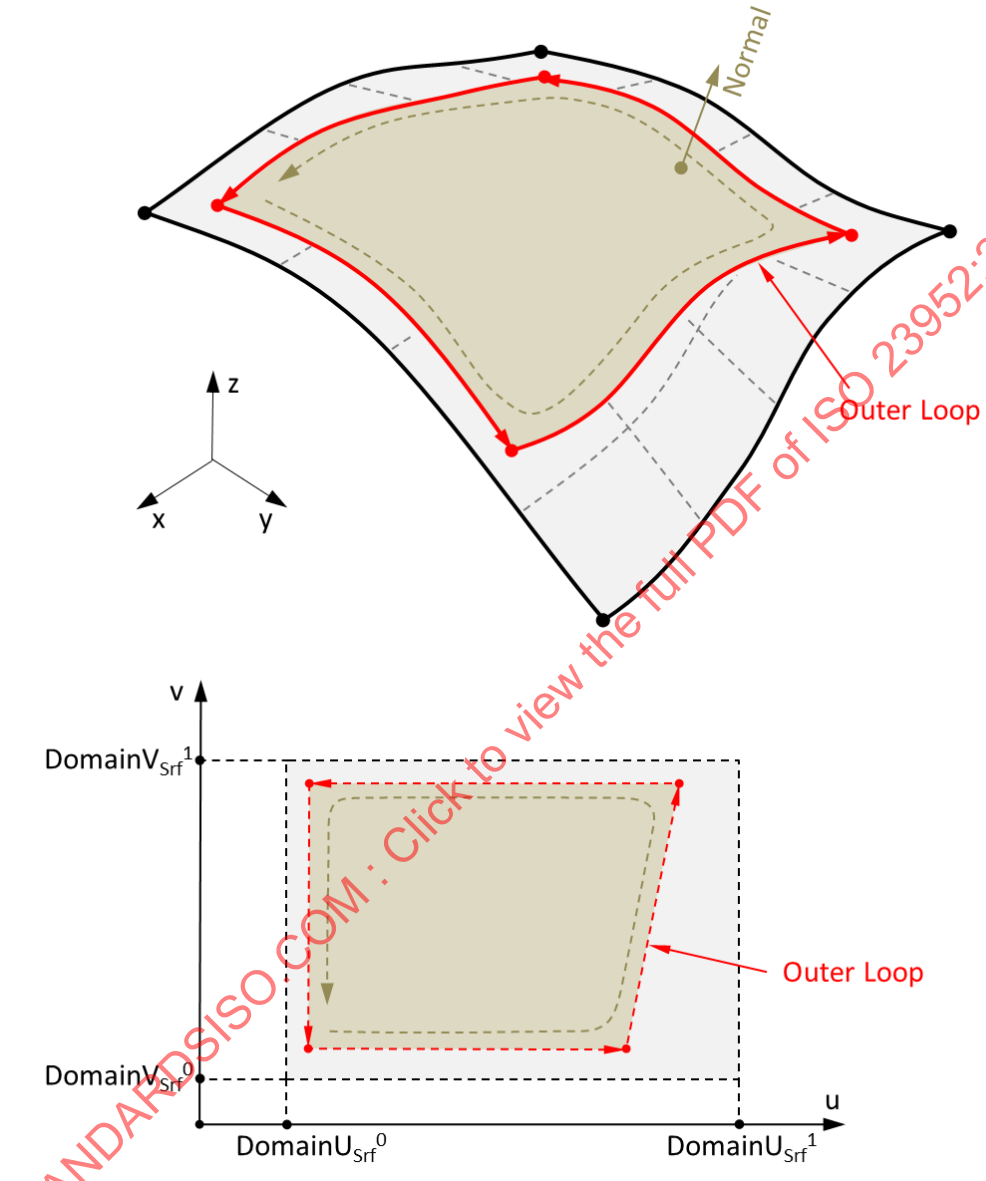


Figure 150 – Outer Loop

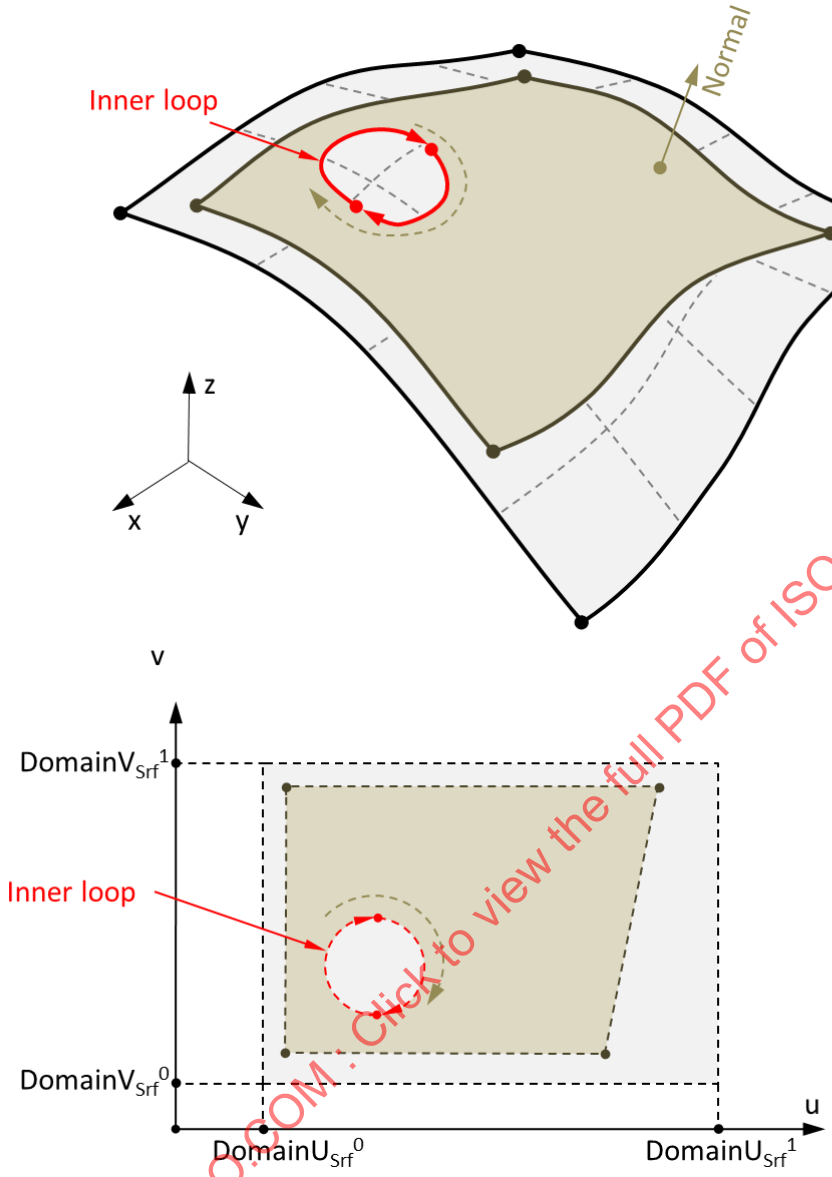


Figure 151 – Inner Loop

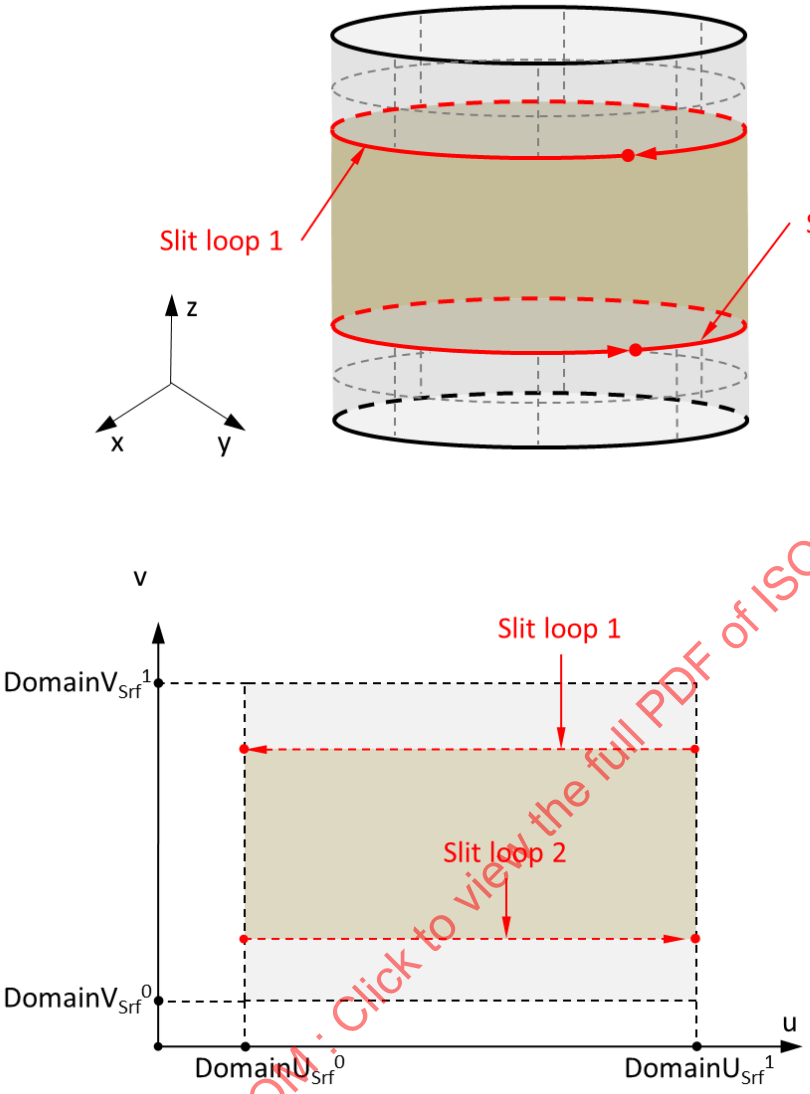


Figure 152 – Slit Loop

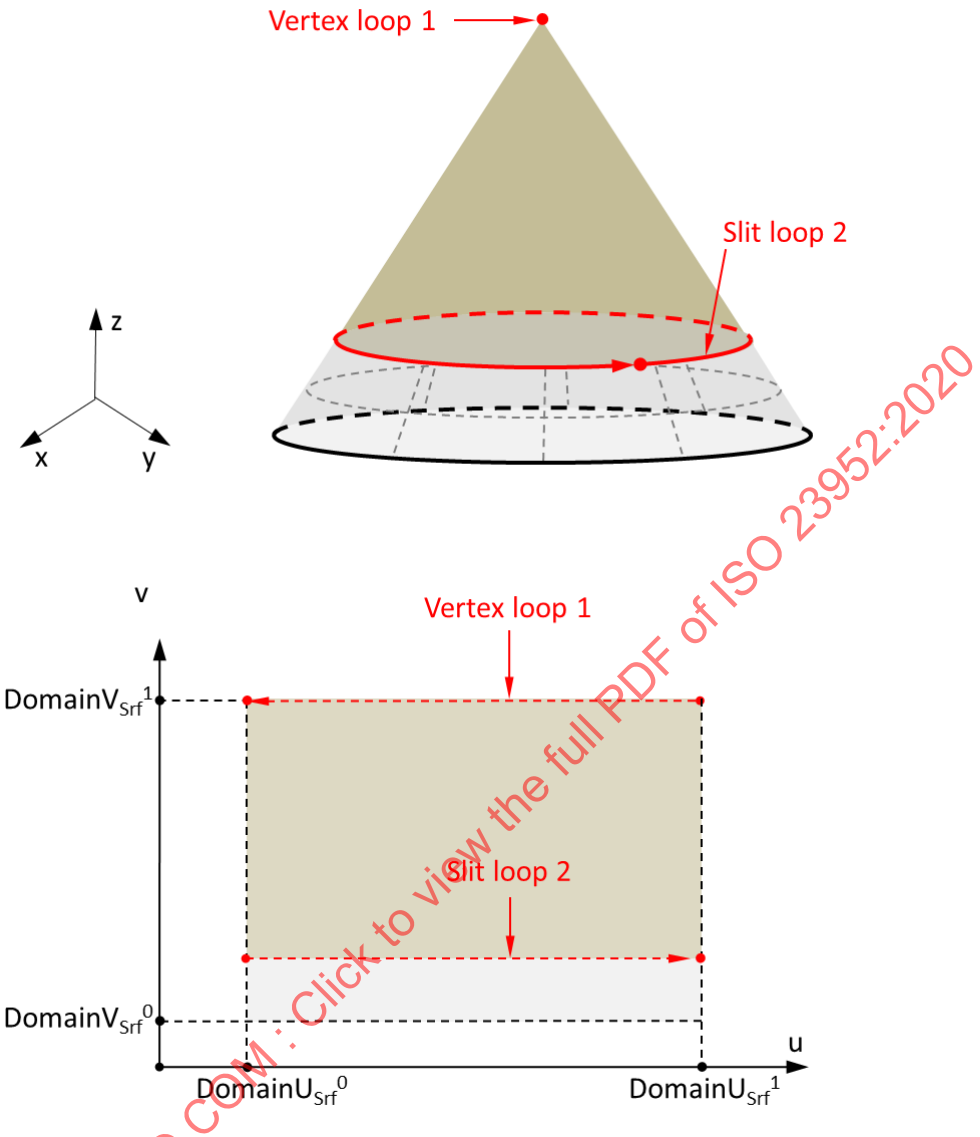


Figure 153 – Vertex Loop

7.5.3.4 Mesh Loop

LoopMesh describes a topological mesh loop as shown in Figure 154.

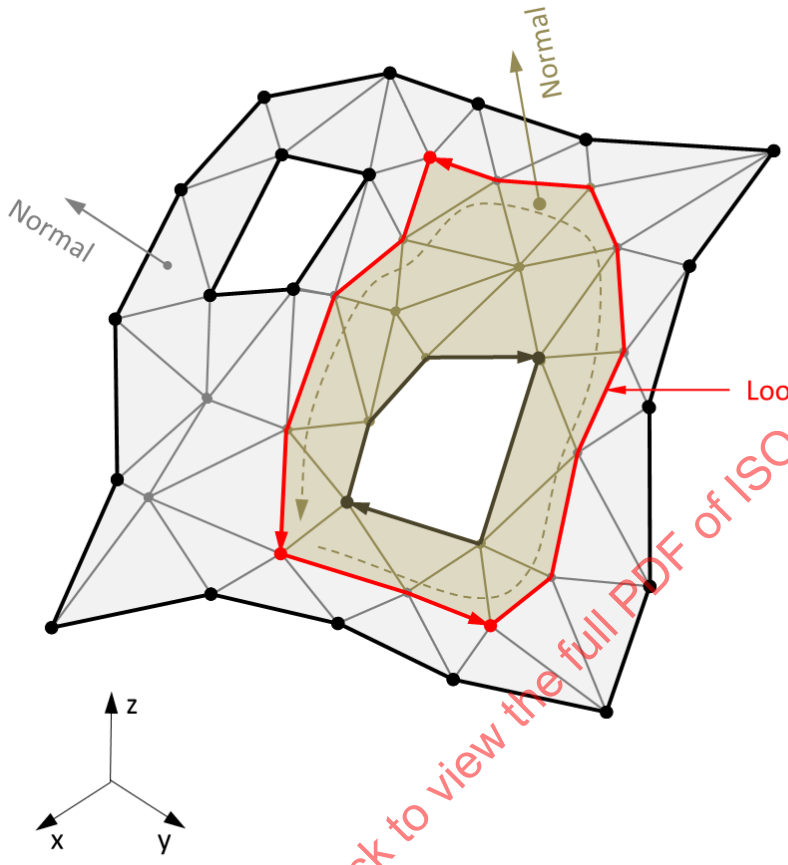


Figure 154 – Mesh Loop

A mesh loop is a circuit of CoEdges bounding a mesh face as shown in Figure 155.

Fields:

Field Name	Data Type	Description
CoEdgesMesh	CoEdgesMeshType	An array of mesh co-edges that forms a mesh loop.
CoEdgesMesh/CoEdgeMesh/EdgeOriented/@turned	xs:boolean	This flag shows if the referenced edge must be reversed from the original edge orientation.
CoEdgesMesh/CoEdgeMesh/EdgeOriented/Id	QIFReferenceType	A reference to an edge with a given orientation.
CoEdgesMesh/CoEdge/CurveMesh	ElementReferenceType	A reference to a mesh curve. This is the projection of the

		underlying 3D Curve of an oriented Edge onto an underlying mesh surface of a mesh face.

Example:

```
<LoopMesh id="113">
  <CoEdgesMesh n="4">
    <CoEdgeMesh>
      <EdgeOriented>
        <Id>405</Id>
      </EdgeOriented>
      <CurveMesh>
        <Id>520</Id>
      </CurveMesh>
    </CoEdgeMesh>
    <CoEdgeMesh>
      <EdgeOriented>
        <Id>469</Id>
      </EdgeOriented>
      <CurveMesh>
        <Id>532</Id>
      </CurveMesh>
    </CoEdgeMesh>
    <CoEdgeMesh>
      <EdgeOriented turned="1">
        <Id>511</Id>
      </EdgeOriented>
      <CurveMesh>
        <Id>547</Id>
      </CurveMesh>
    </CoEdgeMesh>
    <CoEdgeMesh>
      <EdgeOriented>
        <Id>567</Id>
      </EdgeOriented>
      <CurveMesh>
        <Id>563</Id>
      </CurveMesh>
    </CoEdgeMesh>
  </CoEdgesMesh>
</LoopMesh>
```


7.5.3.4.1 *Mesh Co-Edges*

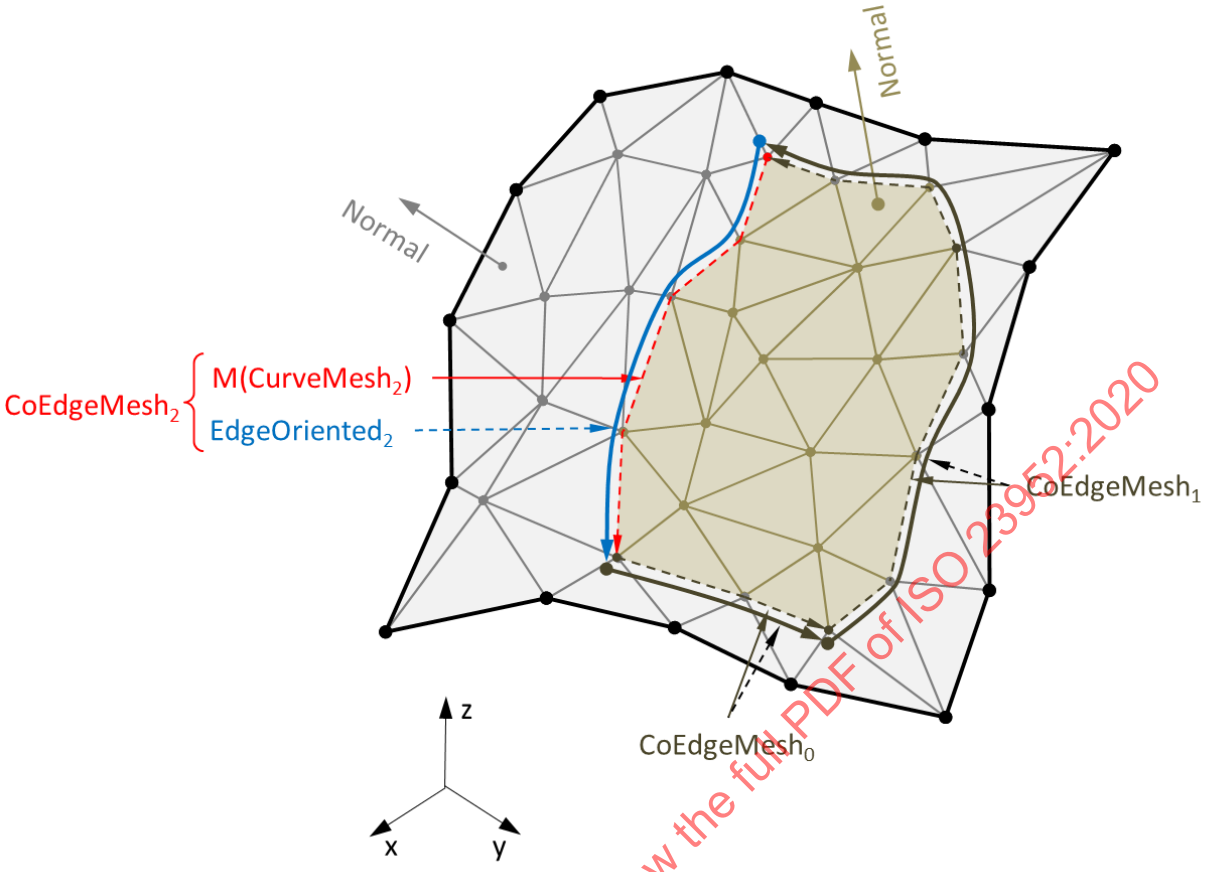


Figure 155 – Mesh Co-Edge

7.5.3.5 Face

Face describes a topological face as shown in Figure 156.

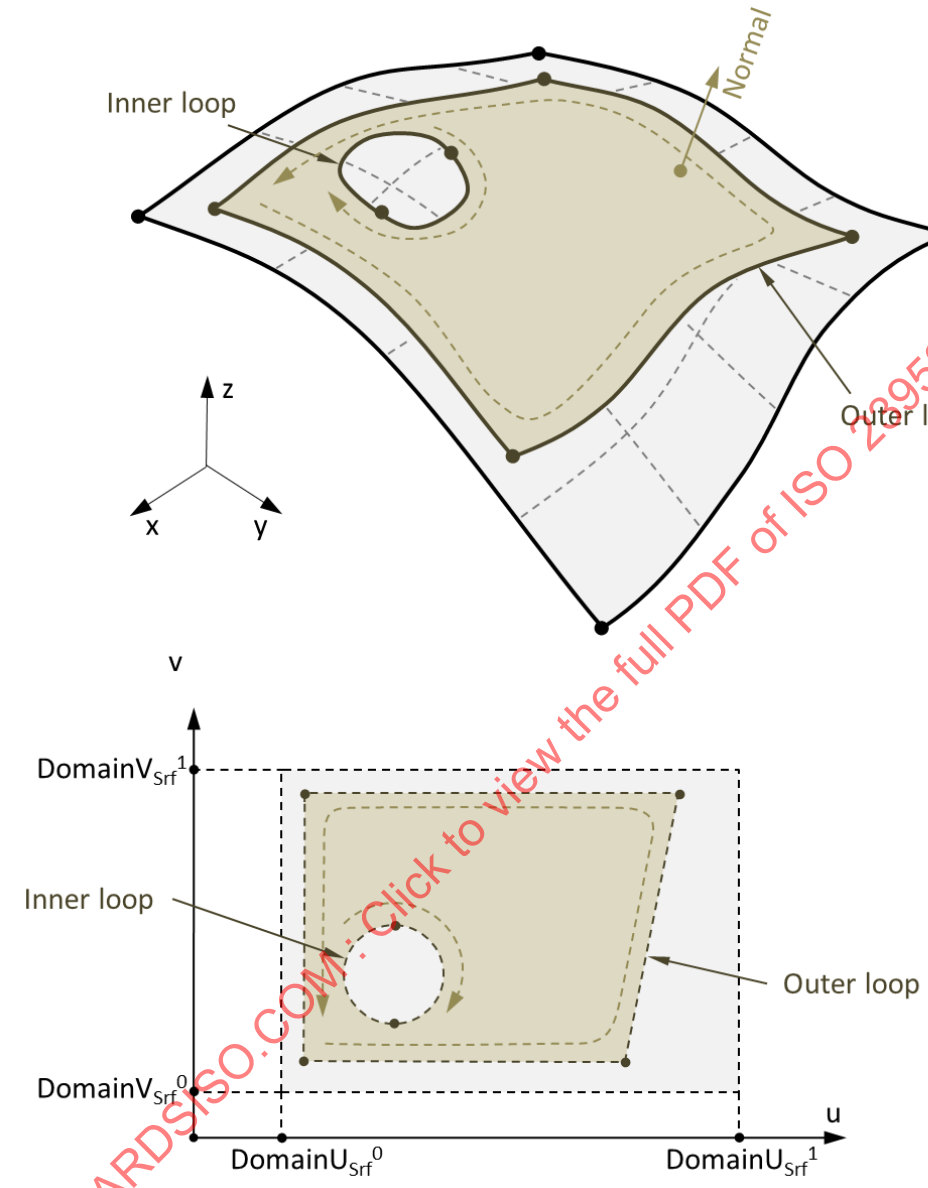


Figure 156 – Face

A face is a bounded portion of a parametric surface.

Fields:

Field Name	Data Type	Description
@turned	xs:boolean	This flag shows if the face normal goes in the opposite direction of the surface normal.
@hasOuter	xs:boolean	This flag shows if the face has its outer loop explicitly represented in the loop array. If this attribute is false, the natural border of the underlying surface shall be taken as the face outer loop.

Validation	ValidationFaceType	The set of face validation properties.
Surface	ElementReferenceType	The identifier of the underlying surface.
LoopIds	ArrayReferenceType	The array of identifiers of the face trimming contours.

Example:

```
<Face id="174" color="191 191 191" turned="1">
  <Surface>
    <Id>102</Id>
  </Surface>
  <LoopIds n="1">
    <Id>113</Id>
  </LoopIds>
</Face>
```

7.5.3.6 Mesh Face

FaceMesh describes a topological mesh face as shown in Figure 157.

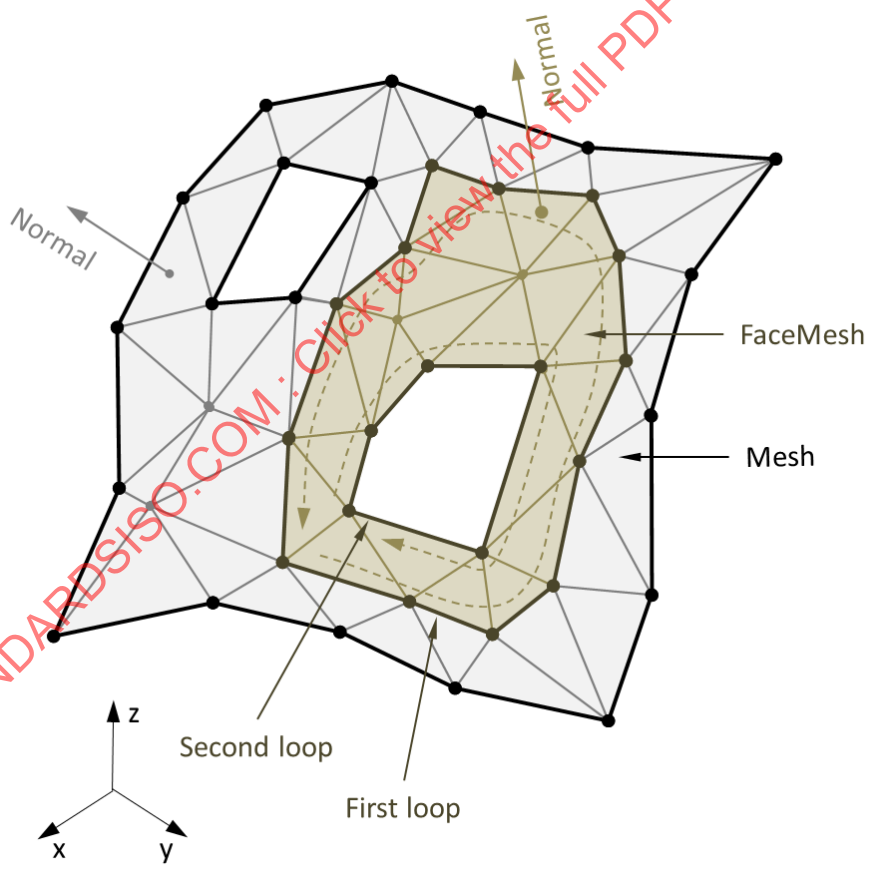


Figure 157 – Mesh Face

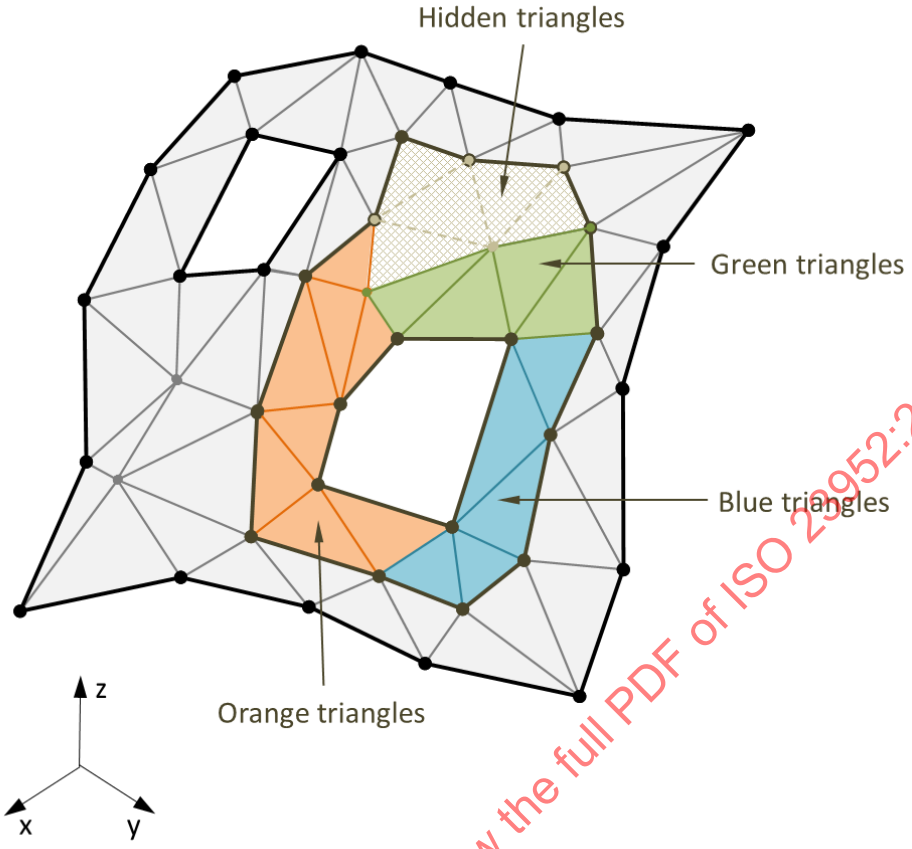


Figure 158 – Mesh Face (Triangle Visibility and Color)

As shown in Figure 158, a mesh face is built on a mesh surface bounded by a set of closed triangulation paths (polylines formed from the triangle edges). The triangles of a mesh face may be colored and they may be hidden. By default, all triangles in a mesh face are visible, and their appearance is the same on both sides.

Fields:

Field Name	Data Type	Description
@turned	xs:boolean	This attribute shows if the face orientation shall be reversed from the orientation of the underlying surface. If the value is true, the face orientation shall be opposite the surface orientation. If the value is false, the two orientations shall be the same.
Validation	ValidationFaceType	The set of face validation properties.
Mesh	ElementReferenceType	The identifier of the underlying mesh surface.
LoopIds	ArrayReferenceType	An array of identifiers of the face trimming contours.
Triangles or	ArrayIntType or	The array of triangle indices of the underlying mesh surface. All elements of this array must be unique

TrianglesBinary	ArrayBinaryType	and must lie in the range [0, number of triangles in the underlying mesh surface - 1].
TrianglesVisible or TrianglesVisibleBinary or TrianglesHidden or TrianglesHiddenBinary	ArrayIntType or ArrayBinaryType or ArrayIntType or ArrayBinaryType	The indexes from the Triangles array of the triangles that should be visible or hidden. If this element is not used, all triangles are visible. If TrianglesVisible[Binary] is used, all other triangles in the mesh face are hidden. If TrianglesHidden[Binary] is used, all other triangles in the mesh face are visible.
TrianglesColor or TrianglesColorBinary	ArrayIntType or ArrayBinaryType	An array of unsigned byte values that defines colors of the face interior triangles. Each element of this array is a triplet of unsigned byte numbers representing the RGB color: the red-component, the green-component and the blue-component. The number of array elements corresponds to the number of triangles in the face interior.

Example:

```
<FaceMesh id="234">
  <Mesh>
    <Id>34</Id>
  </Mesh>
  <LoopIds n="1">
    <Id>55</Id>
  </LoopIds>
  <Triangles n="8">
    19 20 21 22 23 40 41 42
  </Triangles>
  <TrianglesHidden n="3">
    0 6 7
  </TrianglesHidden>
  <TrianglesColor n="8">
    255 255 255
    255 255 255
    255 255 255
    128 0 0
    255 10 255
    100 255 255
    255 10 255
    128 0 0
  </TrianglesColor>
</FaceMesh>
```

7.5.3.7 Shell

Shell describes a topological shell as shown in Figure 159.

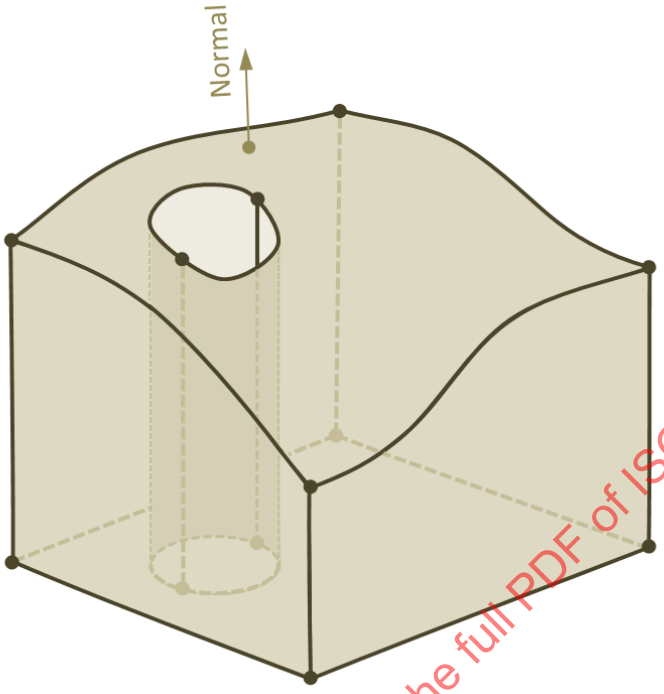


Figure 159 – Shell

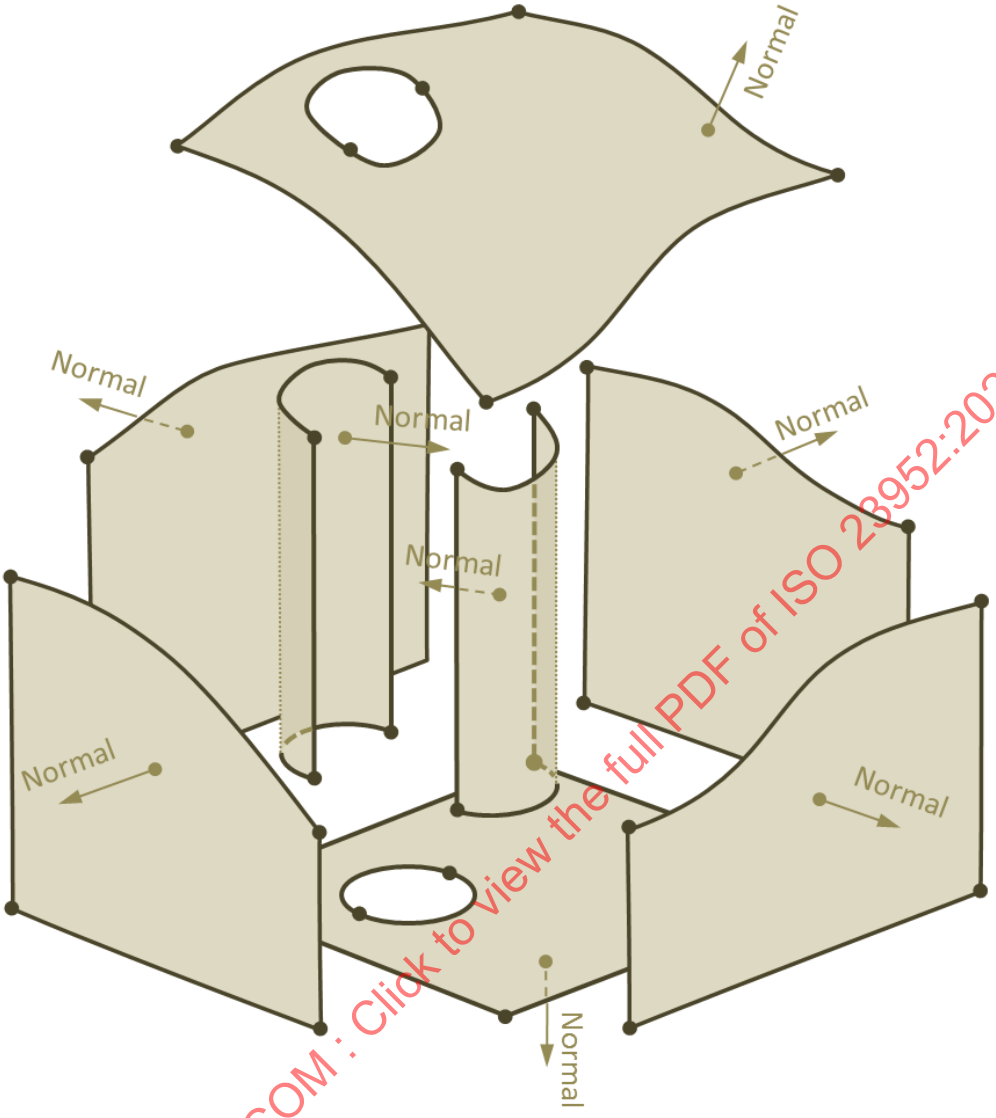


Figure 160 – Shell Faces

A topological shell is a set of connected faces. Figure 160 shows an exploded view of the faces of a shell along with their normal vectors.

Fields:

Field Name	Data Type	Description
@turned	xs:boolean	This flag shows if the shell orientation shall be reversed from the orientation of the component faces. If the value is true, the shell orientation shall be opposite the faces orientation. If the value is false, the two orientations shall be the same.
@closed	xs:boolean	This flag shows if the shell is closed (there are no gaps or open contours).
@form	ShellFormEnumType	The shell type that can take one of the following values: 'UNKNOWN', 'OUTER' or 'INNER'.
FaceIds	ArrayReferenceType	An array of face IDs.

Example:

```
<Shell id="578" closed="1" form="OUTER">
  <FaceIds n="6">
    <Id>176</Id>
    <Id>180</Id>
    <Id>178</Id>
    <Id>174</Id>
    <Id>179</Id>
    <Id>177</Id>
  </FaceIds>
</Shell>
```

7.5.3.8 Body

Body describes a topological body as shown in Figure 161.

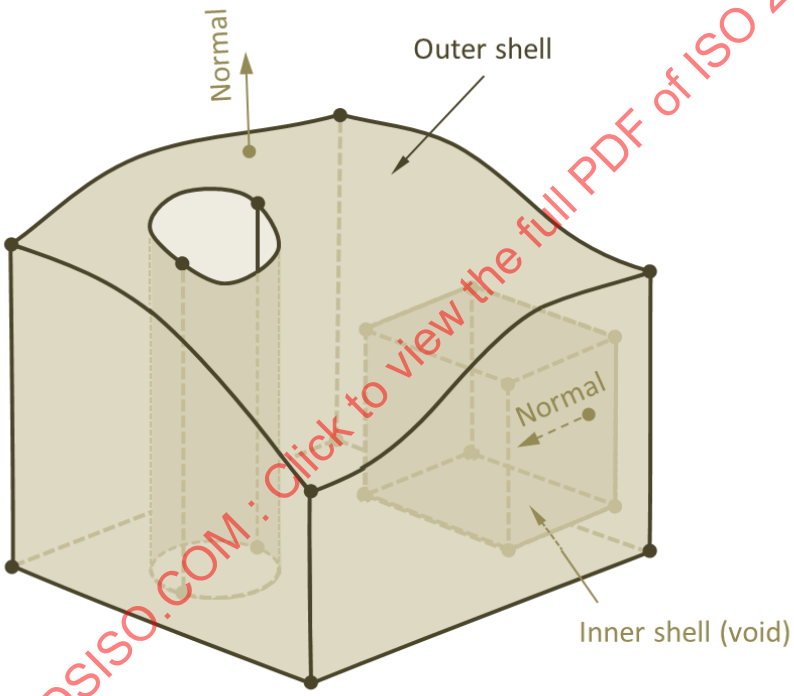


Figure 161 – Body

A body is a solid represented as a set of one outer and a number of inner shells.

Fields:

Field Name	Data Type	Description
@form	BodyFormEnumType	The body type that can take one of the following values: 'UNDEFINED', 'TRIMMED_SURFACE', 'WIRE', 'SOLID', 'SHEET' or 'MIXED'.
Validation	ValidationBodyType	The set of body validation properties.
Transform	ElementReferenceType	The identifier of the transformation matrix.

ShellIds	ArrayReferenceType	The array of shell identifiers
FaceIds	ArrayReferenceType	The array of face identifiers.
LoopIds	ArrayReferenceType	The array of loop identifiers.
EdgeIds	ArrayReferenceType	The array of edge identifiers.
VertexIds	ArrayReferenceType	The array of vertex identifiers.

Example:

```
<Body id="127" color="0 255 0" form="SOLID">
  <ShellIds n="1">
    <Id>578</Id>
  </ShellIds>
  <FaceIds n="6">
    <Id>176</Id>
    <Id>179</Id>
    <Id>180</Id>
    <Id>178</Id>
    <Id>174</Id>
    <Id>177</Id>
  </FaceIds>
  <LoopIds n="6">
    <Id>126</Id>
    <Id>161</Id>
    <Id>171</Id>
    <Id>151</Id>
    <Id>113</Id>
    <Id>139</Id>
  </LoopIds>
  <EdgeIds n="12">
    <Id>539</Id>
    <Id>555</Id>
    <Id>567</Id>
    <Id>575</Id>
    <Id>511</Id>
    <Id>495</Id>
    <Id>487</Id>
    <Id>456</Id>
    <Id>430</Id>
    <Id>405</Id>
    <Id>469</Id>
    <Id>425</Id>
  </EdgeIds>
  <VertexIds n="8">
    <Id>553</Id>
    <Id>566</Id>
    <Id>494</Id>
    <Id>486</Id>
    <Id>451</Id>
    <Id>401</Id>
    <Id>462</Id>
    <Id>421</Id>
  </VertexIds>
</Body>
```

7.5.3.9 Point Cloud

PointCloud describes a cloud of points as shown in Figure 162 and Figure 163.

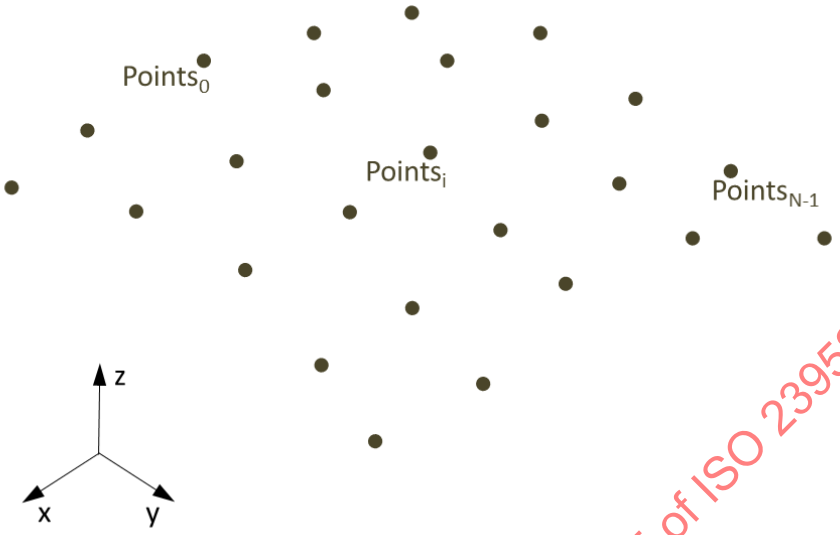


Figure 162 – Cloud of Points

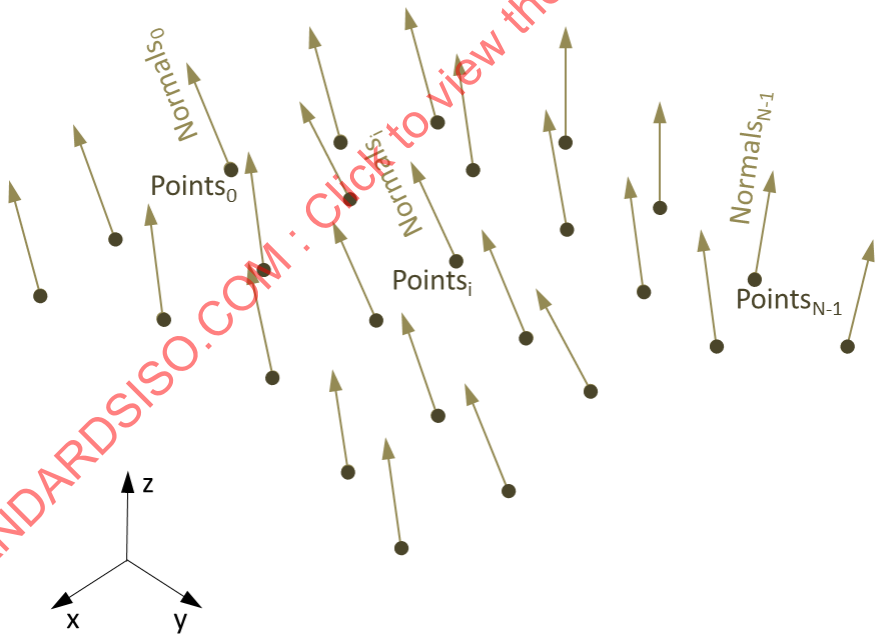


Figure 163 – Cloud of Point with Defined Normals

$N - number\ of\ points$

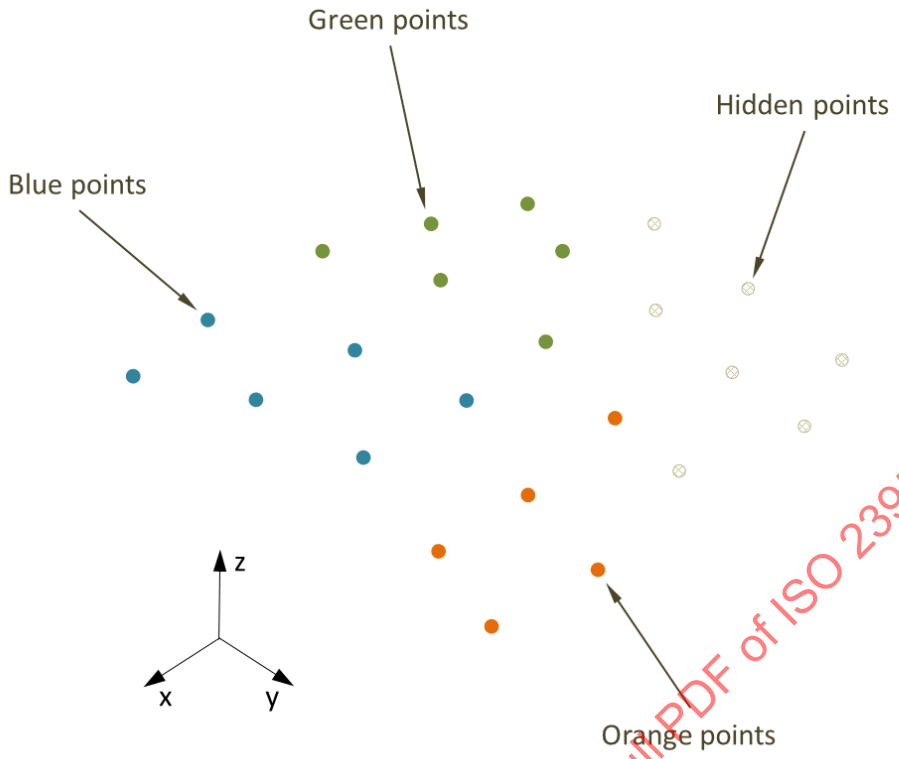


Figure 164 – Point Cloud (Point Visibility and Color)

A point cloud is a collection of 3D points with an optional set of associated normals. The points may be colored and they may be hidden. In order to simplify work with point clouds QIF considers this object as a topological exception. This means that this type of geometric data does not need any extra topological wrapper and it can participate in the model scene as it is.

Fields:

Field Name	Data Type	Description
Points or PointsBinary	ArrayPointType or ArrayBinaryType	The array of 3D points.
Normals or NormalsBinary	ArrayUnitVectorType or ArrayBinaryType	The array of normals. The number of elements in this array must equal the number of 3D points in this point cloud.
PointsVisible or PointsVisibleBinary or PointsHidden or PointsHiddenBinary	ArrayIntType or ArrayBinaryType or ArrayIntType or ArrayBinaryType	The indexes from the Points array of the points that should be visible or hidden. If this element is not used, all points are visible. If PointsVisible[Binary] is used, all other points in the point cloud are hidden. If PointsHidden[Binary] is used, all other points in the point cloud are visible.
PointsColor or PointsColorBinary	ArrayIntType or ArrayBinaryType	An array of unsigned byte values that defines colors of the points. Each element of this array is a triplet of unsigned byte numbers representing the RGB color: the red-component, the green-component and the

		blue-component. The number of array elements corresponds to the number of points.

Example:

```
<PointCloud id="3">
  <Points n="20">
    -1.16929133858268 -0.78740157480315 0.511811023622047
    -0.883047171186937 -0.687746062992126 0.638852435112278
    -0.63918051910178 -0.620078740157481 0.622630504520268
    -0.259113444152814 -0.551181102362205 0.374744823563721
    0.00984251968503937 -0.521653543307087 0.196850393700787
    -1.27238261883931 -0.212890055409741 -0.0729075532225137
    -0.980417571260382 -0.19196288735513 0.125172480368966
    -0.731792013652614 -0.177678407483015 0.203090740200685
    -0.334651069850837 -0.162556840888716 0.146355162394824
    -0.0211431904345291 -0.15456401283173 0.0527121609708776
    -1.28273549139691 0.134149897929426 -0.596383785360163
    -0.995118906818745 0.0690377148072541 -0.360411738656125
    -0.750102610630461 0.0254231338366655 -0.245109793374594
    -0.352964367108432 -0.0143654882645842 -0.24891178726116
    -0.0215077282006419 -0.0193205016039662 -0.352580927384077
    -1.37795275590551 0.551181102362205 -1.10236220472441
    -1.12086249635462 0.318970545348498 -0.995051399825022
    -0.896033829104695 0.163312919218431 -0.939413823272091
    -0.514727325750948 0.020414114902303 -0.927384076990376
    -0.177165354330709 0 -0.954724409448819
  </Points>
  <Normals n="20">
    -0.623765689571477 0.380773422526389 0.682589162828535
    -0.310237069934693 0.562301118777707 0.766531416355309
    0.090750754593406 0.625699548958669 0.774767303758777
    0.577779206607798 0.413801444322832 0.70351940491219
    0.00191672240401954 -0.0146948717641499 0.99989018743013
    -0.509232347350378 0.675520319522933 0.533249204709822
    -0.326094688542491 0.757232035434036 0.56591686546418
    -0.118001946734718 0.795610989745604 0.594204252394364
    0.205316690584309 0.75203536789529 0.626328876870692
    0.00588691478872499 0.776036953180493 0.630659965062479
    -0.325919814041952 0.783893402891083 0.528476496846074
    -0.0597624697937558 0.921696358846995 0.383280927378786
    0.0741285257671752 0.965747087229765 0.248671520634772
    0.0810453652229759 0.988778769243019 0.125491809573915
    0.0200710836160788 0.993714825343097 0.110127187813973
    0.398014876083999 0.696526033146993 0.597022314125991
    0.588385732990174 0.795976710575262 0.142208668636623
    0.501600088180723 0.865095712095178 0.00260008685018021
    0.16965400244185 0.984848612494367 -0.0359267299288623
    0 1 0
  </Normals>
</PointCloud>
```


7.5.3.10 Sewn Faces

Two faces sewn together have at least one common edge between them that corresponds to co-edges defined on each of the two faces. The bypass of the co-edges shall be oriented so that they are opposite from each other in order to conform to the normals of each face. If the orientations of the underlying surfaces are not conformal – the face optional attribute “@turned” shall be used to specify that the face normal is opposite to its underlying surface normal. Sewn faces are illustrated in Figure 165 and Figure 166.

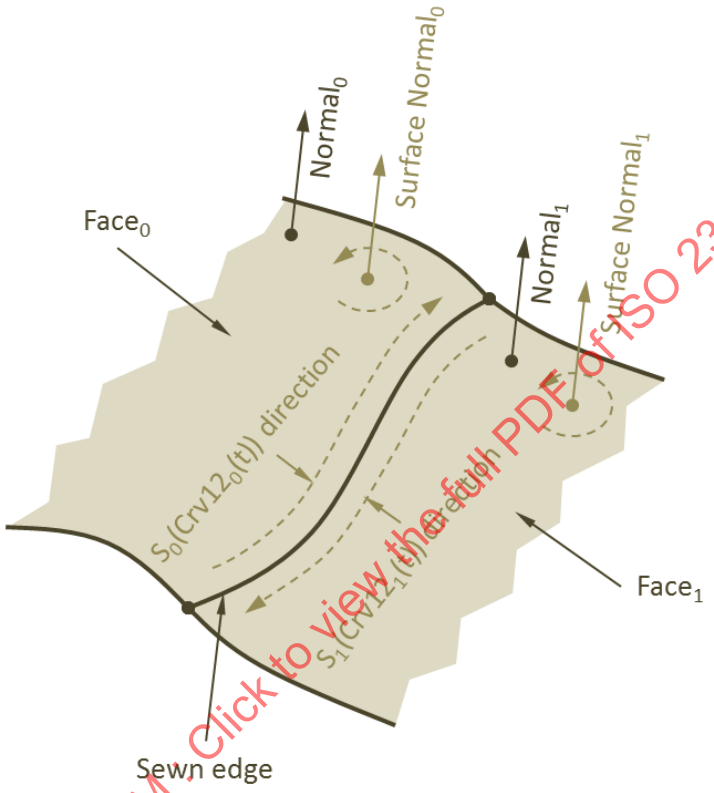


Figure 165 – Sewn Faces (normals of the underlying surfaces are conformed: $Turned_0 = Turned_1 =$ FALSE)

Figure 166 – Sewn Faces (normals of the underlying surfaces are not conformed: Turned$_0$ (FALSE) ≠ Turned$_1$ (TRUE))

One of implications of the face normals conformance requirement is impossibility of a Mobius strip definition.

7.5.3.11 Tolerant Edges and Vertices

It is highly recommended to maintain one global model tolerance for a whole body. It makes the body definition more robust for geometric algorithms and less problematic to transfer to other environments, systems and formats. There are a number of geometric kernels, which allow the loosening of tolerances on some model entities while maintaining topological connectivity. This concept is referred to as tolerant modeling.

QIF can store data from both types of modelers (exact and tolerant) by use of the optional attribute “@tolerance” that can be specified for edges and vertices.

Tolerant edges shall contain the actual tolerance value, which is calculated as the maximum distance between the 2D parametric co-edges of the neighbor faces.

Tolerant vertices shall contain the actual tolerance value, which is calculated as the maximum distance from the vertex underlying 3D point to the ends of all neighboring edges that are terminated in the neighborhood of this vertex.

Figure 167 illustrates tolerant edges and vertices.

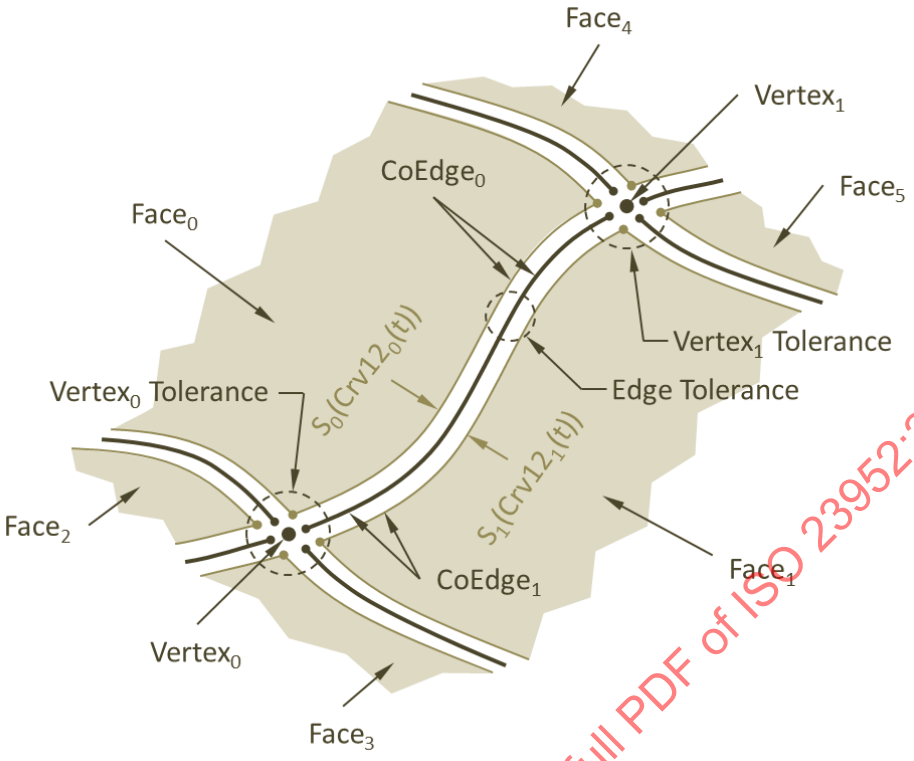


Figure 167 – Tolerant Edges and Vertices

7.5.4 Product structure

7.5.4.1 Assembly Graph: Root, Parts, Assemblies and Components

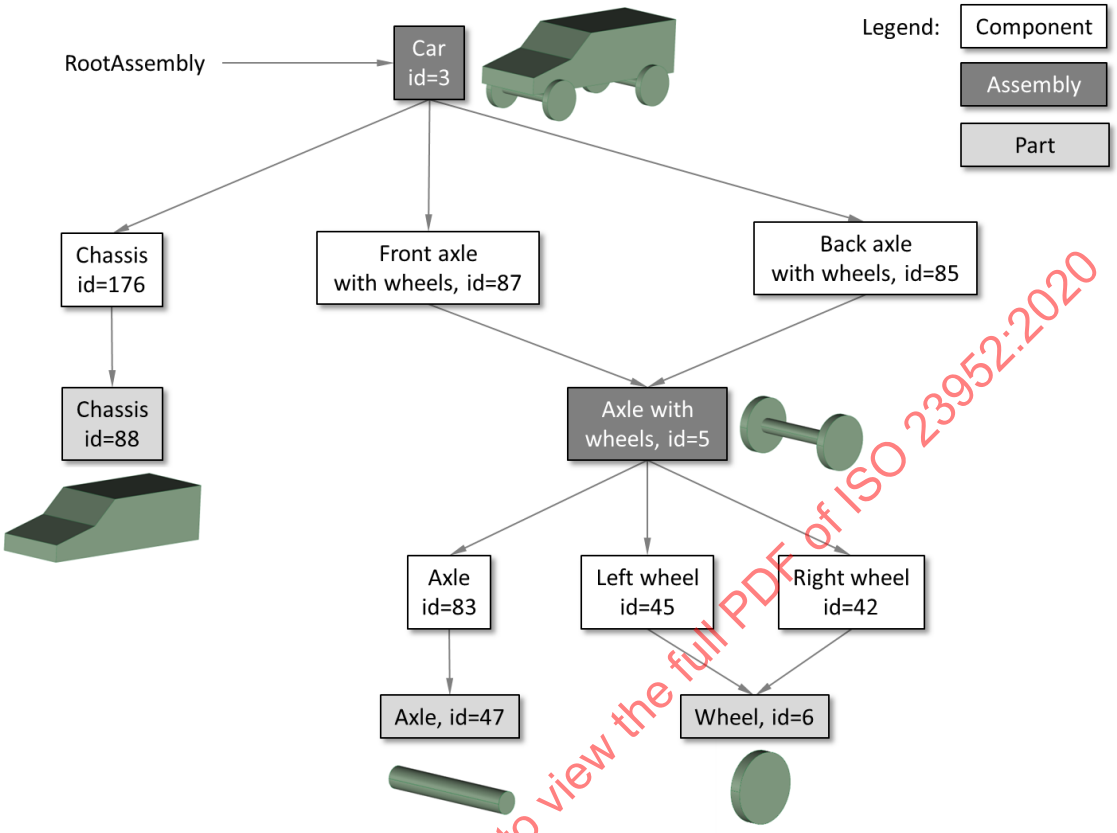


Figure 168 – Product directed acyclic graph

Figure 168 shows the product directed acyclic graph representing the relationships between parts, assemblies and components that describe the product content.

A part defines one body or a number of bodies (also known as a “multi-body part”), and can be instantiated multiple times in the CAD scene. The use of part instances simplifies the need to maintain identical model elements and reduces the total amount of scene data.

An assembly defines one or more bodies and may also include other components (instances of other parts or sub-assemblies). An assembly itself can be instantiated multiple times in the CAD scene. The use of assemblies simplifies the need to maintain identical model elements and reduces the total amount of scene data.

Assembly Fields (product graph relationships):

Field Name	Data Type	Description
ComponentIds	ArrayReferenceType	The array of component identifiers used in the product definition.

Component defines a single instance of a part or an assembly.

Fields of component (product graph relationships):

Field Name	Data Type	Description
Transform	ElementRefernceType	The identifier of the transformation matrix that maps the component into the product.
Part or Assembly	ArrayReferenceType	The identifier of a part or assembly to be instantiated.

Root of the product directed acyclic graph:

Root	Description
Product/RootPart	The starting point of the product directed acyclic graph for a part. The graph contains only one node because the part cannot contain references to any components or assemblies.
Product/RootAssembly	The starting point of the product directed acyclic graph for an assembly.
Product/RootComponent	The starting point of the product directed acyclic graph for a component.

7.5.4.2 References in Assemblies (AsmPaths)

AsmPath defines information about an assembly path; the path is used to identify entities within an assembly, as seen in Figure 169. The assembly path is a sequence of identifiers of the scene components that contain the entity of interest. The sequence defines the path from the root to the entity parent component. The id of the parent component shall be the last element in this sequence.

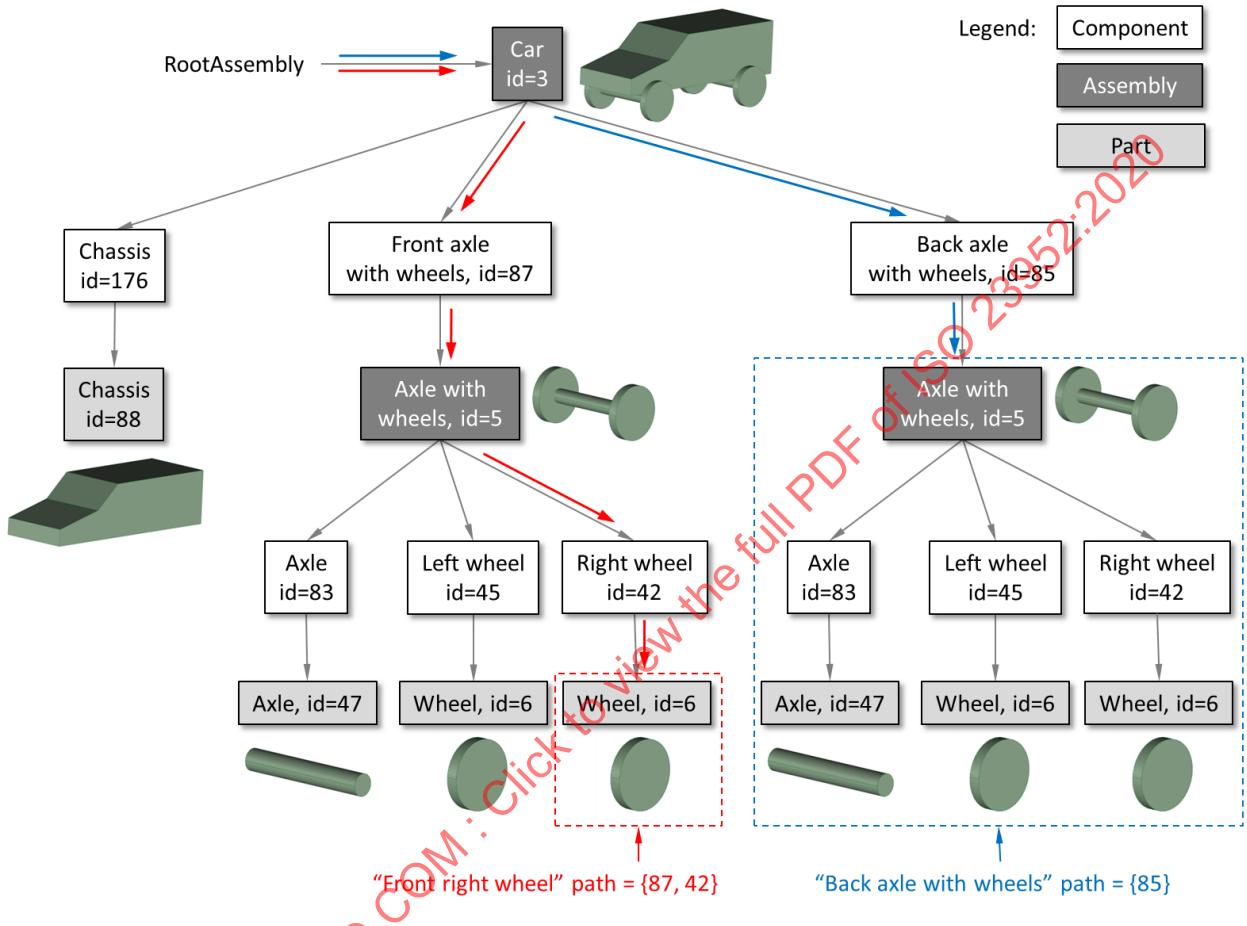


Figure 169 – Unfolded product tree

AsmPath definition:

$$AsmPath = \{ComponentIds_0, \dots, ComponentIds_{n-1}\}$$

$$n - number\ of\ elements\ in\ ComponentIds$$

$$ComponentIds_0 - this\ component\ does\ not\ have\ a\ parent\ component$$

$$ComponentIds_{i-1} - the\ nearest\ parent\ component\ for\ ComponentIds_i, \qquad i \in [1, n-1]$$

AsmPath Fields:

Field Name	Data Type	Description
@id	QIFIdType	The QIF identifier.
ComponentIds	ArrayReferenceType	The array of identifiers of the scene components that contain the entity of interest. This array defines the path

		from the root to the parent component. The id of the parent component must be the last element of this array.

Unfolded Part instances within the Product:

Part	Part id	Instance	AsmPath
Chassis	88	Chassis	176
Axle	47	Front axle	87, 83
Axle	47	Back axle	85, 83
Wheel	6	Front left wheel	87, 45
Wheel	6	Front right wheel	87, 42
Wheel	6	Back left wheel	85, 45
Wheel	6	Back right wheel	85, 42

Unfolded Assembly instances within the Product:

Assembly	Assembly id	Instance	AsmPath
Car	3	Car	<empty>
Axle with wheels	5	Front axle with wheels	87
Axle with wheels	5	Back axle with wheels	85

In QIF all assembly paths used for referencing are collected in the table AsmPaths only when needed. Every assembly path has its own unique QIFId that is included in one or more referencing elements of QIFReferenceFullType. This provides a clear, unambiguous and compact way of referencing part entities within an assembly of any complexity. See Figure 170 for an example.

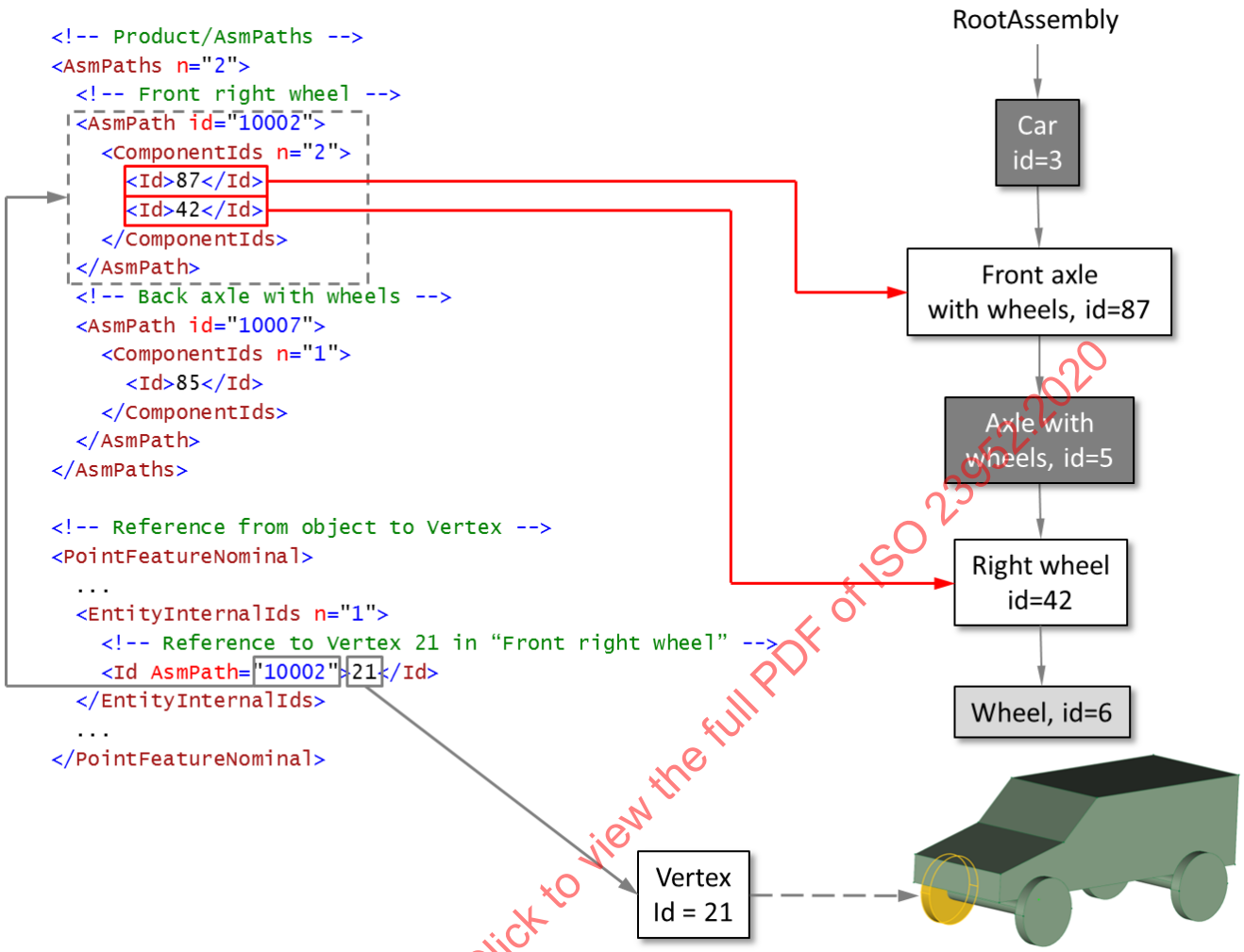


Figure 170 – Reference of a part entity within an assembly

Example:

```
<!-- Product/AsmPaths -->
<AsmPaths n="2">
  <!-- Front right wheel -->
  <AsmPath id="10002">
    <ComponentIds n="2">
      <Id>87</Id>
      <Id>42</Id>
    </ComponentIds>
  </AsmPath>
  <!-- Back axle with wheels -->
  <AsmPath id="10007">
    <ComponentIds n="1">
      <Id>85</Id>
    </ComponentIds>
  </AsmPath>
</AsmPaths>

<!-- Reference from an object to a Vertex -->
<PointFeatureNominal>
```



```
  ...
  <EntityInternalIds n="1">
    <!-- Reference to the Vertex 21 in "Front right wheel" -->
    <Id AsmPath="10002">21</Id>
  </EntityInternalIds>
  ...
</PointFeatureNominal>
```

7.5.4.3 Positioning of parts, sub-assemblies and bodies

Parts and subassemblies may be instantiated multiple times via components. Positioning of a component (part or subassembly) in 3D space is defined by a transformation specified in the Transform field of the component.

$$T(x) - transformation$$

$$T(x)\colon R_3 \to R_3$$

$$PA - part\ or\ subassembly$$

$$AsmPath - assembly\ path\ to\ PA$$

$$AsmPath = \{C_0, \dots, C_{n-1}\}$$

$$TC_i(x) - transformation\ associated\ with\ the\ i-th\ component,\ where\ x - a\ 3D\ point\ of\ PA$$

$$TC_i(x) = \begin{cases} transformation\ referenced\ by\ Transform, & if\ the\ Transform\ field\ is\ filled \\ identity\ transformation, & otherwise \end{cases}$$

Positioning of PA geometry inside the parent component C_{n-1} shall be calculated by applying transformation $TC_{C_{n-1}}$ to the PA entities. To calculate position and orientation of PA entities in Global Model Space, it is necessary to follow the entire instantiation chain (AsmPath) sequentially applying transformations coming from PA to the assembly root.

$$TPAmodel(x) - total\ transformation\ of\ a\ 3D\ point\ defined\ in\ PA\ to\ Global\ Model\ Space$$

$$TPAmodel(x) = \left(TC_{C_0} \circ TC_{C_1} \circ \dots \circ TC_{C_{n-1}}\right)(x), \qquad x - the\ 3D\ point\ defined\ in\ PA$$

A body can be included in only one part or assembly (the DefinitionInternal/BodyIds field in definition of Part/Assembly). Positioning of bodies is defined by the transformation specified in the Transform field.

$$B - a\ body\ referenced\ from\ a\ part\ or\ a\ subassembly\ PA$$

$$TB_i(x) - transformation\ associated\ with\ the\ i-th\ body, \qquad where\ x - a\ 3D\ point\ of\ B$$

$$TB_i(x) = \begin{cases} transformation\ referenced\ by\ Transform, & Transform\ presented\ in\ the\ i-th\ body \\ identity\ transformation, & otherwise \end{cases}$$

To calculate the position and orientation of the B geometry inside PA it is necessary to apply the TB_B transformation to the B geometry.

To calculate the position and orientation of the B entities in Global Model Space, it is necessary to apply the $TPAmodel$ transformations as described above.

$$TBmodel_i(x) \; - transformation\ of\ a\ 3D\ point\ defined\ in\ the\ i_{th}\ body\ Global\ Model\ Space$$

$$TBmodel_B(x) = (TPAmodel \circ TB_B)(x), \qquad x - a\ 3D\ point\ defined\ in\ B$$

7.5.4.4 External Definitions of Parts and Assemblies

Figure 171 shows how a part or assembly can also contain external definitions such as files, images, drawings, CAD files of native or neutral formats, physical model or prototype, etc.

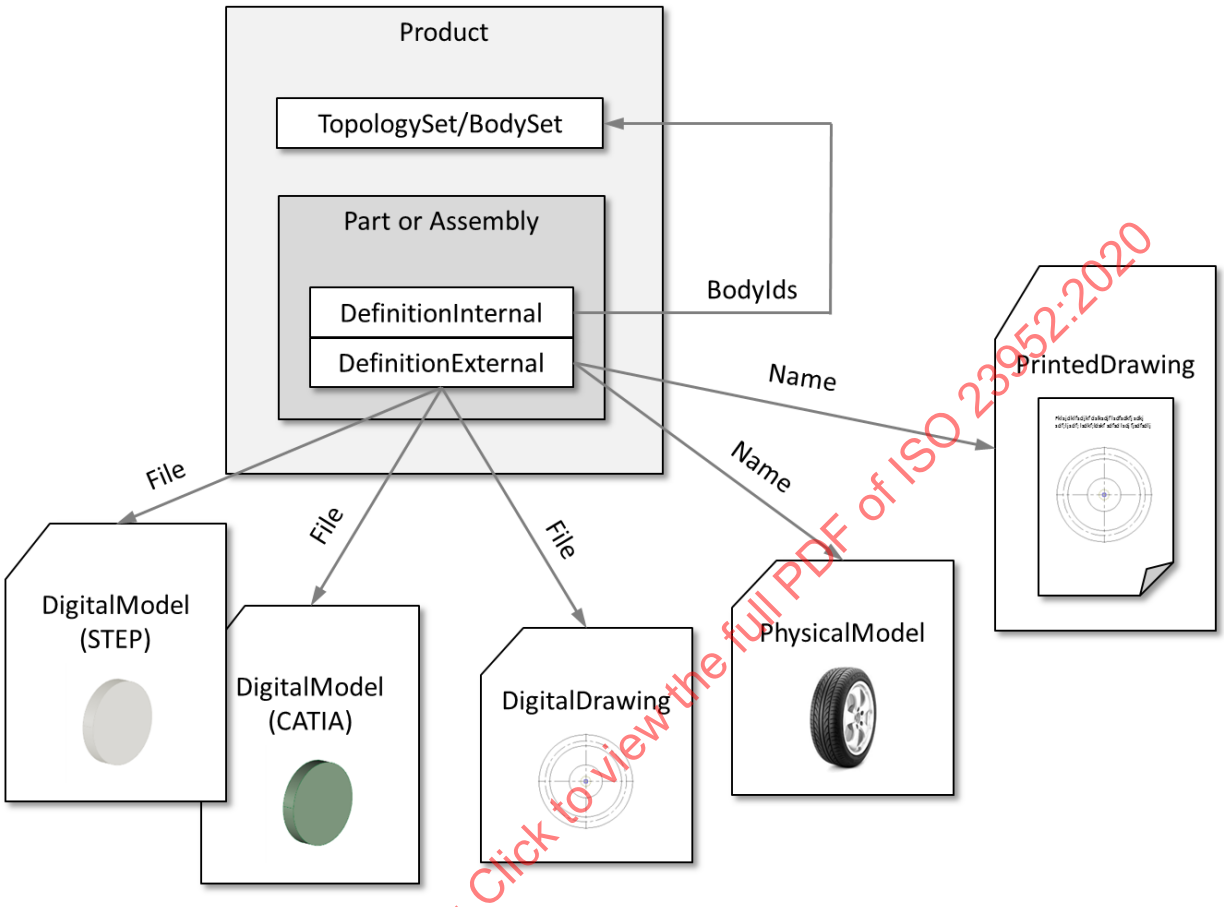


Figure 171 – Multiple representations for parts and assemblies

7.5.4.4.1 *PrintedDrawing*

PrintedDrawing defines information about a printed drawing of a product. This may be on paper, mylar, or some other physical media.

Fields:

Field Name	Data Type	Description
Name	xs:string	The name of the model.
Version	xs:string	The version of the model associated with product being inspected.
Description	xs:string	A description of the model.
Author/Name	xs:string	The name of the author.
Author/Organization	xs:string	The name of author's organization.
DrawingNumber	xs:string	The drawing number of the printed drawing associated with product being inspected.

AdditionalChanges	xs:string	The description or references to descriptions of any additional changes to the drawing beyond what is included in the Version.
Location	xs:string	A description of the physical location of the printed drawing.

Example:

```
<PrintedDrawing id="1">
  <Name>Widget Drawing</Name>
  <Version>2</Version>
  <Description>Widget Drawing rev02</Description>
  <DrawingNumber>12345</DrawingNumber>
  <Location>Engineering</Location>
</PrintedDrawing>
```

7.5.4.4.2 *DigitalDrawing*

The DigitalDrawing defines an electronic version of a drawing. It has only 2D drawing information. Any geometric dimensioning and tolerancing information is drawn with annotation symbols.

Fields:

Field Name	Data Type	Description
Name	xs:string	The name of the model
File/Name	xs:token	The fully qualified identifier (URI) of the file.
File/Version	xs:token	The version number of the file.
File/Format	DigitalModelFormatType	The file format.
Application/Name	xs:string	The name of the software application wherein the model was most recently edited.
Application/Organization	xs:string	The name of the organization that created the software application wherein the model was most recently edited.
Application/AddonName	xs:string	The name of the software add-on application wherein the model was most recently edited.
Application/AddonOrganization	xs:string	The name of the organization that created the software add-on application wherein the model was most recently edited.
Author/Name	xs:string	The name of the author.
Author/Organization	xs:string	The name of author's organization.
ApplicationSource/Name	xs:string	The name of the software application wherein the model was created.

ApplicationSource/Organization	xs:string	The name of the organization that created the software application wherein the model was created.
ApplicationSource/AddonName	xs:string	The name of the software add-on application wherein the model was created.
ApplicationSource/AddonOrganization	xs:string	The name of the organization that created the software add-on application wherein the model was created.
Description	xs:string	A description of the model.
Entities	EntitiesExternalType	A list of instances of the EntityExternalType associated with the model. Only those entities from the model that need to be referenced should be included in this list.

Example:

```
<DigitalDrawing id="5">
  <Name>Drawing</Name>
  <File>
    <Name>c:/models/drawing_133.catdrawing</Name>
    <Format>
      <DigitalModelFormatEnum>CATIA</DigitalModelFormatEnum>
    </Format>
  </File>
  <Description>Sample drawing</Description>
</DigitalDrawing>
```

7.5.4.4.3 *DigitalModel*

The DigitalModel defines a digital data model that represents information about an assembly or part.

Fields:

Field Name	Data Type	Description
Name	xs:string	The name of the model
File/Name	xs:token	The fully qualified identifier (URI) of the file.
File/Version	xs:token	The version number of the file.
File/Format	DigitalModelFormatType	The file format.
Application/Name	xs:string	The name of the software application wherein the model was most recently edited.
Application/Organization	xs:string	The name of the organization that created the software application wherein the model was most recently edited.

Application/AddonName	xs:string	The name of the software add-on application wherein the model was most recently edited.
Application/AddonOrganization	xs:string	The name of the organization that created the software add-on application wherein the model was most recently edited.
Author/Name	xs:string	The name of the author person.
Author/Organization	xs:string	The name of author's organization.
ApplicationSource/Name	xs:string	The name of the software application wherein the model was created.
ApplicationSource/Organization	xs:string	The name of the organization that created the software application wherein the model was created.
ApplicationSource/AddonName	xs:string	The name of the software add-on application wherein the model was created.
ApplicationSource/AddonOrganization	xs:string	The name of the organization that created the software add-on application wherein the model was created.
Description	xs:string	A description of the model.
Units	OtherUnitsType	Specifies the units used in the model.
GDT	GDTEnumType	Specifies the presence of geometric dimensioning and tolerancing information in model.
Topology	TopologyEnumType	Specifies the presence of topology information in model.
Entities	EntitiesExternalType	A list of instances of the EntityExternalType associated with the model. Only those entities from the model that need to be referenced should be included in this list.

Example:

```
<DigitalModel id="6">
  <Name>Model</Name>
  <File>
    <Name>c:/models/model_12.sat</Name>
    <Format>
      <DigitalModelFormatEnum>ACIS</DigitalModelFormatEnum>
    </Format>
  </File>
  <Description>Sample drawing</Description>
  <Units>
    <LinearUnit>
```



```
      <UnitName>mm</UnitName>
    </LinearUnit>
  </Units>
  <GDT>MACHINEREAD</GDT>
  <Topology>PRESENT</Topology>
  <Entities>
    <Entity id="101">
      <EntityId>1903</EntityId>
      <Name>Wheel body</Name>
    </Entity>
    <Entity id="102">
      <EntityId>4324</EntityId>
      <Name>Chassis body</Name>
    </Entity>
  </Entities>
</DigitalModel>
```

7.5.4.4.4 *PhysicalModel*

The PhysicalModel is an actual, hands-on, physical, model or prototype of a part that is used to communicate features, datum reference frames, characteristics, or other information. Examples of physical models include actual part instances, part instances made from other materials, and 3D printing and stereolithography models.

Fields:

Field Name	Data Type	Description
Name	xs:string	The name of the model.
Version	xs:string	The version of the model associated with product being inspected.
Description	xs:string	A description of the model.
Author/Name	xs:string	The name of the author.
Author/Organization	xs:string	The name of author's organization.
Location	xs:string	A description of the physical location of the physical model
ModelNumber	xs:string	The model number of the physical model being inspected.

Example:

```
<PhysicalModel id="7">
  <Name>Wheel</Name>
  <Version>2</Version>
  <Location>Warehouse 2, position 24A/10</Location>
  <ModelNumber>23876-2372-33</ModelNumber>
</PhysicalModel>
```

7.5.4.5 Layers

A layer is a “slice” of the model containing a set of model entities independent of the referencing model hierarchy. The user can specify an arbitrary number of layers for one model. A model entity can be included in a single layer or multiple layers.

Fields:

Field Name	Data Type	Description
@applyColor	xs:boolean	Shows if the layer color supersedes colors of the model entities associated with this layer.
@index	xs:unsignedInt	The layer index.
ElementIds	ArrayReferenceFullType	An array of entity identifiers presents in this this layer.

Example:

```
<Layer id="33" applyColor="0" hidden="0" label="Layer 1" index="12">
  <ElementIds n="4">
    <Id>23</Id>
    <Id>24</Id>
    <Id>1324</Id>
    <Id>1323</Id>
  </ElementIds>
</Layer>
```

7.5.4.6 Part Notes

PartNote describes a part (standard) note – a treelike note used for annotating of a whole model or a separate model entity. In QIF all part notes are collected in the PartNoteSet. Normally part notes are visible in the model tree, and in some systems are also visualized in the graphical window so the part notes are implemented in QIF as drawable entities with the corresponding set of attributes.

Fields:

Field Name	Data Type	Description
Text	xs:string	A text of the current node of the part note.
PartNoteIds	ArrayReferenceType	An array of identifiers of nested/children part notes.

Example:

```
<PartNote id="1427" label="NIST Test Case 1 CATIA V5R21 RB">
  <PartNoteIds n="7">
    <Id>1428</Id>
    <Id>1429</Id>
    <Id>1430</Id>
    <Id>1431</Id>
    <Id>1432</Id>
    <Id>1433</Id>
    <Id>1434</Id>
  </PartNoteIds>
```



```
</PartNote>
<PartNote id="1428" label="Part Number">
  <Text>NIST Test Case 1 CATIA V5R21 RB</Text>
</PartNote>
<PartNote id="1429" label="Revision">
  <Text>B</Text>
</PartNote>
<PartNote id="1430" label="Nomenclature">
  <Text>NIST Complex Test Case 1 V5R21</Text>
</PartNote>
<PartNote id="1431" label="Source">
  <Text>Unknown</Text>
</PartNote>
<PartNote id="1432" label="Modeled By">
  <Text>Rich Eckenrode, RECON Services.com</Text>
</PartNote>
<PartNote id="1433" label="CAGE Code">
  <Text>64JW1</Text>
</PartNote>
<PartNote id="1434" label="Company">
  <Text>RECON Services Incorporated</Text>
</PartNote>
```

7.5.4.7 Notes

Note describes an annotation note, which may be visualized in the graphical window as a 3D annotation or as a flat-to-screen note.

Fields:

Field Name	Data Type	Description
@form	NoteFormEnumType	This attribute specifies the note form: '3D' - note defined on the 3D annotation plane; 'SCREEN' - note defined as flat to screen;
EntityInternalIds	ArrayReferenceFullType	An array of identifiers of CAD entities associated with this note.
EntityExternalIds	ArrayReferenceFullType	An array of instance identifiers of EntityExternalType associated with this note.
Text	xs:string	A text of the note.

Example:

```
<Note id="1455" label="Note_10" form="3D">
  <EntityInternalIds n="1">
    <Id>2195</Id>
  </EntityInternalIds>
  <Text>B</Text>
</Note>
```

Flag notes

NoteFlag describes a flag note, which is visualized by its flag note symbol and can be attached to model entities. The shape of a flag note (flag style) is defined in an associated display info object.

Fields:

Field Name	Data Type	Description
@form	NoteFormEnumType	This attribute specifies the note form: '3D' - note defined on the 3D annotation plane; 'SCREEN' - note defined as flat to screen;
EntityInternalIds	ArrayReferenceFullType	An array of identifiers of CAD entities associated with this flag note.
EntityExternalIds	ArrayReferenceFullType	An array of identifiers of instances of EntityExternalType associated with this flag note.
Text	xs:string	A text of the flag note (the flag note symbol).
TextHidden	xs:string	A hidden text of the flag note.
URI	xs:anyURI	A Uniform Resource Identifier for the information, which may be, for example, a file or a web site.

Example:

```
<NoteFlag id="731" label="Flag Note_1">
  <EntityInternalIds n="1">
    <Id>1096</Id>
  </EntityInternalIds>
  <Text>1</Text>
</NoteFlag>
```


7.5.5 Transformations

In order to support all varieties of systems and formats, QIF allows connecting transformations to all topological and 3D geometric elements. However, it is recommended to limit the use of transformations to high-level objects such as sub-assemblies, parts and bodies. That will help eliminate any potential interoperability issues.

Transformation defines positioning of a 3D point specifying rotation and translation.

$$T(p) - transformation$$

$$T(p)\colon R_3 \to R_3$$

$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}, \qquad Origin = \begin{bmatrix} Origin_x \\ Origin_y \\ Origin_z \end{bmatrix}, \qquad R = \begin{bmatrix} XDirection_x & XDirection_y & XDirection_z \\ YDirection_x & YDirection_y & YDirection_z \\ ZDirection_x & ZDirection_y & ZDirection_z \end{bmatrix}$$

$$T(p) = pR + Origin$$

Fields:

Field Name	Data Type	Description
Rotation/XDirection	UnitVectorSimpleType	An orthogonal basis of the Cartesian coordinate system. Direction X.
Rotation/YDirection	UnitVectorSimpleType	An orthogonal basis of the Cartesian coordinate system. Direction Y.
Rotation/ZDirection	UnitVectorSimpleType	An orthogonal basis of the Cartesian coordinate system. Direction Z.
Origin	PointSimpleType	An origin of the coordinate system.
Attributes	AttributesType	User defined attributes (typified, binary array, or XML structured).
Name	xs:token	The name of the transform.

Example:

```
<Transform id="348">
  <Rotation>
    <XDirection>0.9405 -0.3396 0.0</XDirection>
    <YDirection>0.3318 0.9191 -0.2123</YDirection>
    <ZDirection>0.0721 0.1996 0.9772</ZDirection>
  </Rotation>
  <Origin>0.0798 -0.2104 -0.0984</Origin>
</Transform>
```

7.5.6 Auxiliary data

7.5.6.1 Point

PointAuxiliary describes an auxiliary point, a point that does not directly participate in forming a body, but in a definition of supplemental model entities, e.g. the center of a sphere, as seen in Figure 172.

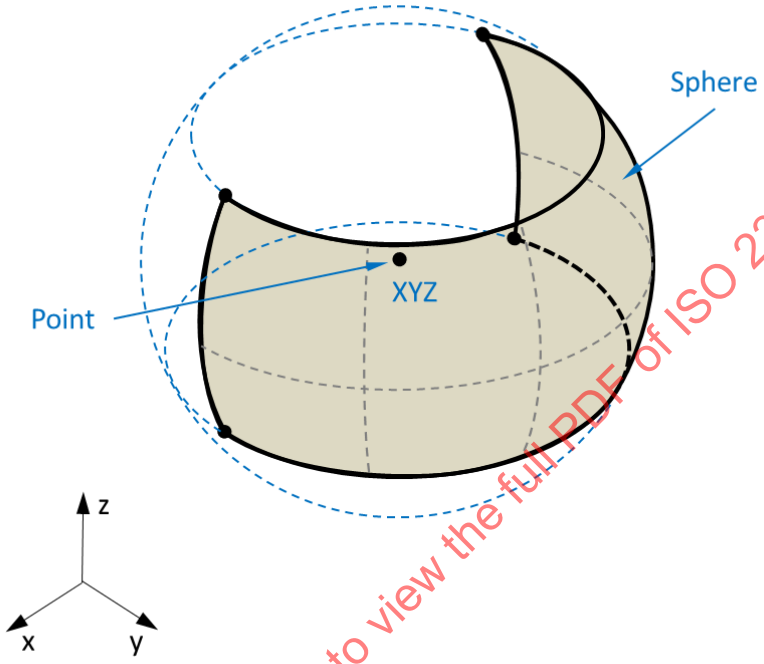


Figure 172 – Point

Fields:

Field Name	Data Type	Description
XYZ	PointType	The Cartesian three-dimensional coordinates of the 3D point.

Example:

```
<PointAuxiliary id="112">
  <XYZ>1.32 9.23 4.22</XYZ>
</PointAuxiliary>
```

7.5.6.2 Line

LineAuxiliary describes an auxiliary line, a line that does not directly participate in forming a body, but in a definition of supplemental model entities, e.g. the axis of a cylinder, as seen in Figure 173.

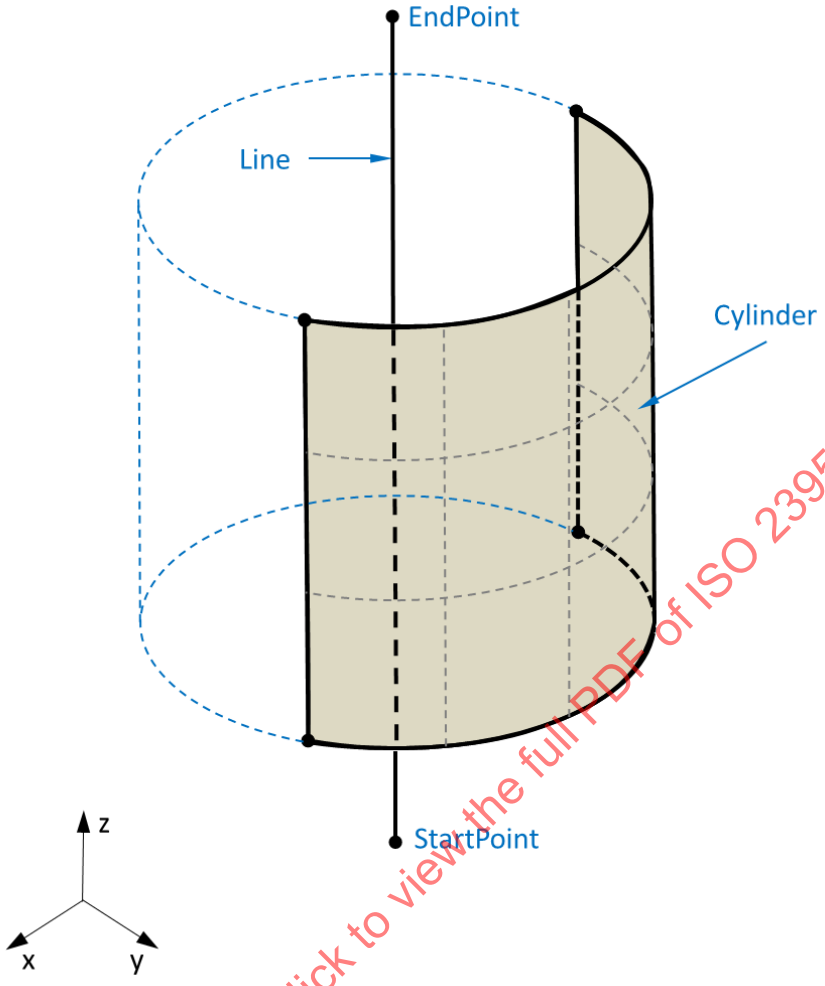


Figure 173 – Line

Fields:

Field Name	Data Type	Description
StartPoint	PointSimpleType	The beginning point of the line segment.
EndPoint	PointSimpleType	The ending point of the line segment.
LineStyle	LineStyleEnumType	The line style of the line segment.
LineStyle/@thickness	xs:positiveInteger	The thickness of the line in pixels.

Example:

```
<LineAuxiliary id="21">
  <StartPoint>1.32 9.23 4.22</StartPoint>
  <EndPoint>3.45 0.33 5.40</EndPoint>
</LineAuxiliary>
```

7.5.6.3 Reference Plane

PlaneReference describes an auxiliary plane, a plane that does not directly participate in forming of a body, but in a definition of supplemental model entities, e.g. the central plane of a feature, as seen in Figure 174.

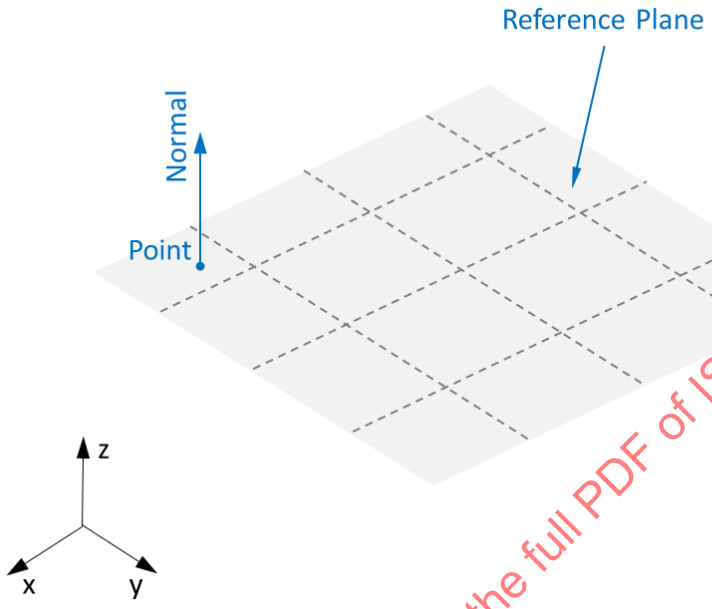


Figure 174 – Reference Plane

Fields:

Field Name	Data Type	Description
Plane/Point	PointType	The plane origin.
Plane/Normal	UnitVectorType	The unit normal vector of the plane.

Example:

```
<PlaneReference id="23">
  <Plane>
    <Point>10.0 12.0 23.1</Point>
    <Normal>1.0 0.0 0.0</Normal>
  </Plane>
</PlaneReference>
```

7.5.6.4 Coordinate System

CoordinateSystem describes the Cartesian 3D coordinate system in the model space, as seen in Figure 175. The CoordinateSystem is defined as a model entity.

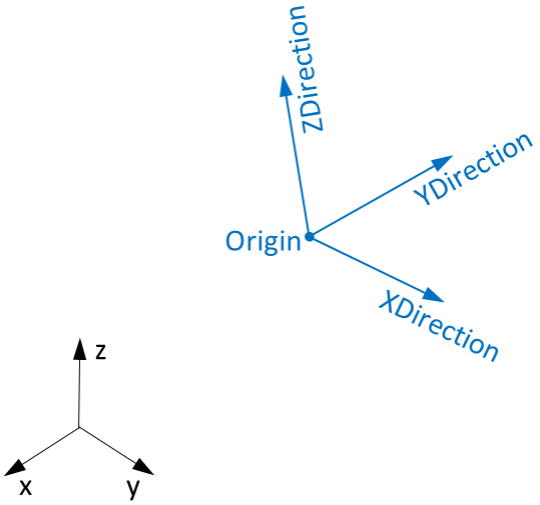


Figure 175 – Coordinate System

$$ZDirection = XDirection \times YDirection$$

$$(XDirection \cdot YDirection) = 0$$

$$(XDirection \cdot ZDirection) = 0$$

$$(YDirection \cdot ZDirection) = 0$$

Fields:

Field Name	Data Type	Description
CoordinateSystemCore/Rotation/XDirection	UnitVectorSimpleType	An orthogonal basis of the Cartesian coordinate system. Direction X.
CoordinateSystemCore/Rotation/YDirection	UnitVectorSimpleType	An orthogonal basis of the Cartesian coordinate system. Direction Y.
CoordinateSystemCore/Rotation/ZDirection	UnitVectorSimpleType	An orthogonal basis of the Cartesian coordinate system. Direction Z.
CoordinateSystemCore/Origin	PointSimpleType	The origin of the coordinate system.

Example:

```
<CoordinateSystem id="33" label="CS1">
  <CoordinateSystemCore>
    <Rotation>
      <XDirection>0.9405 -0.3396 0.0</XDirection>
      <YDirection>0.3318 0.9191 -0.2123</YDirection>
```

```
      <ZDirection>0.0721 0.1996 0.9772</ZDirection>
    </Rotation>
    <Origin>0.0798 -0.2104 -0.0984</Origin>
  </CoordinateSystemCore>
</CoordinateSystem>
```


7.5.7 Visualization data

The VisualizationSet represents a container for storing all visualization data used for displaying PMI entities in a 3D scene.

7.5.7.1 Line style

The LineStyleType defines a line style used for visualization of cross-sections hatching and auxiliary lines.

The names from enumeration LineStyleEnumType define sequences of dashes and dots forming a line pattern.

DOT < SHORT_DASH < DASH < MEDIUM_DASH < MEDIUM_LONG_DASH < LONG_DASH

The QIF line styles:

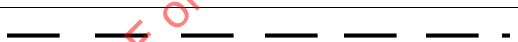
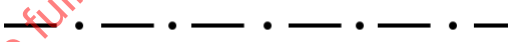
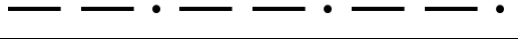
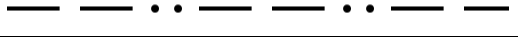
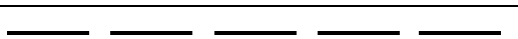
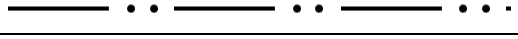
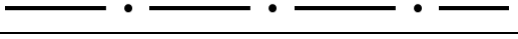
Style Name	Pattern
SOLID	
DASH	
DOT	
DASH_DOT	
DASH_DOT_DOT	
DASH_DOT_DOT_DOT	
DASH_DASH_DOT	
DASH_DASH_DOT_DOT	
DASH_DASH_DOT_DOT_DOT	
SHORT_DASH	
LONG_DASH	
MEDIUM_LONG_DASH	
MEDIUM_DASH_SHORT_DASH_SHORT_DASH	
LONG_DASH_SHORT_DASH	
LONG_DASH_SHORT_DASH_SHORT_DASH	
LONG_DASH_DOT_DOT	
LONG_DASH_DOT	
MEDIUM_DASH_DOT_SHORT_DASH_DOT	
SPARSE_DOT	

Table 6 – Line Styles

Fields:

Field Name	Data Type	Description
.	LineStyleEnumType	The line style.
@thickness	xs:positiveInteger	The thickness of line in pixels.

7.5.7.2 Fonts

All fonts used for visualization of 3D annotations are collected in the Fonts, an *element* of the VisualizationSet.

Font fields:

Field Name	Data Type	Description
@index	xs:unsignedInt	An index identifying a font within the Fonts table.
@bold	xs:boolean	This attribute shows if the text shall be bold.
@italic	xs:boolean	This attribute shows if the text shall be italicized.
@underline	xs:boolean	This attribute shows if the text shall be underlined.
Attributes	AttributesType	User defined attributes (typified, binary array, or XML structured).
Name	xs:string	The font name.
Size	NaturalType	The font size defined in points (typography, 1 pt = 1/72 inch).
Alignment	AlignmentEnumType	The optional alignment *element* defines the note alignment, ALIGNMENT_LEFT is the default value

Example:

```
<Font index="1" italic="1">
  <Name>Microsoft Sans Serif</Name>
  <Size>8</Size>
  <Alignment>ALIGNMENT_CENTER</Alignment>
</Font>
```

7.5.7.3 Special symbols

In order to store special symbols in text strings (PMIDisplay/Texts/Text) the following decorated symbol names shall be used:

Decorated Symbol Name	Image	Description
{STRAIGHTNESS}	—	Straightness
{FLATNESS}	▱	Flatness
{CIRCULARITY}	○	Circularity/Roundness
{CYLINDRICITY}	⌭	Cylindricity
{PROFILE_LINE}	⌒	Profile of a Line

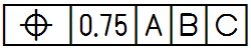
{PROFILE_SURFACE}	⌓	Profile of a Surface
{ANGULARITY}	∠	Angularity
{PERPENDICULARITY}	⊥	Perpendicularity
{PARALLELISM}	//	Parallelism
{POSITION}	⌖	Position
{CONCENTRICITY}	◎	Concentricity
{SYMMETRY}	⌯	Symmetry
{RUNOUT_CIRCULAR}	↗	Circular Runout
{RUNOUT_TOTAL}	⌰	Total Runout
{DIAMETER}	Ø	Diameter
{DIAMETER_SPHERICAL}	SØ	Spherical Diameter
{RADIUS}	R	Radius
{RADIUS_SPHERICAL}	SR	Spherical Radius
{RADIUS_CONTROLLED}	CR	Controlled Radius
{BETWEEN}	⟷	Between
{CONICAL_TAPER}	⌲	Conical Taper
{SLOPE}	⌯	Slope
{COUNTERBORE}	⌴	Counterbore
{SPOTFACE}	⌴	Spotface
{COUNTERSINK}	⌵	Countersink
{DEPTH}	↧	Depth
{SQUARE}	□	Square
{PLUS_MINUS}	±	Plus Minus
{DEGREE}	°	Degree
{ST}	ⓈⓉ	Statistical Tolerance
{CF}	ⓒⓕ	Continuous Feature
{M}	Ⓜ	At Maximum Material Condition
{L}	Ⓛ	At Least Material Condition
{P}	Ⓟ	Projected Tolerance Zone
{T}	Ⓣ	Tangent Plane
{F}	Ⓕ	Free State
{U}	Ⓤ	Unequally Disposed Profile
{I}	Ⓘ	Independency
{TRANSLATION}	▷	Translation

{BRACKET_CURLED_LEFT}	{	Left Curled Bracket
{BRACKET_CURLED_RIGHT}	}	Right Curled Bracket
{BR}		Line Break
{WELD_FILLET_ARROW}		Fillet Arrow Side
{WELD_FILLET_OTHER}		Fillet Other Side
{WELD_FILLET_BOTH}		Fillet Both Sides
{WELD_FILLET_BOTH_STAGGERED}		Fillet Both Sides Staggered
{WELD_SPOT_ARROW}		Spot Arrow Side
{WELD_SPOT_OTHER}		Spot Other Side
{WELD_SPOT_NO}		Spot No Side
{WELD_STUD_ARROW}		Stud Arrow Side
{WELD_SEAM_ARROW}		Seam Arrow Side
{WELD_SEAM_OTHER}		Seam Other Side
{WELD_SEAM_NO}		Seam No Side
{WELD_BACK_ARROW}		Back Arrow Side
{WELD_BACK_OTHER}		Back Other Side
{WELD_SURFACING_ARROW}		Surfacing Arrow Side
{WELD_SQUARE_ARROW}		Square Arrow Side
{WELD_SQUARE_OTHER}		Square Other Side
{WELD_SQUARE_BOTH}		Square Both Sides
{WELD_V_ARROW}		V Arrow Side
{WELD_V_OTHER}		V Other Side
{WELD_V_BOTH}		V Both Sides
{WELD_BEVEL_ARROW}		Bevel Arrow Side
{WELD_BEVEL_OTHER}		Bevel Other Side
{WELD_BEVEL_BOTH}		Bevel Both Sides
{WELD_U_ARROW}		U Arrow Side
{WELD_U_OTHER}		U Other Side
{WELD_U_BOTH}		U Both Sides
{WELD_J_ARROW}		J Arrow Side
{WELD_J_OTHER}		J Other Side
{WELD_J_BOTH}		J Both Sides
{WELD_FLARE_BEVEL_ARROW}		Flare Bevel Arrow Side
{WELD_FLARE_BEVEL_OTHER}		Flare Bevel Other Side

{WELD_FLARE_BEVEL_BOTH}		Flare Bevel Both Sides
{WELD_SCARF_V_ARROW}		Scarf Arrow Side
{WELD_SCARF_V_OTHER}		Scarf Other Side
{WELD_SCARF_V_BOTH}		Scarf Both Sides
{WELD_CONTOUR_FLAT_ARROW}		Contour Flat Arrow Side
{WELD_CONTOUR_FLAT_OTHER}		Contour Flat Other Side
{WELD_CONTOUR_FLAT_ANGLED_ARROW}		Contour Angled Flat Arrow Side
{WELD_CONTOUR_FLAT_ANGLED_OTHER}		Contour Angled Flat Other Side
{WELD_CONTOUR_CONCAVE_ARROW}		Contour Concave Arrow Side
{WELD_CONTOUR_CONCAVE_OTHER}		Contour Concave Other Side
{WELD_CONTOUR_CONCAVE_ANGLED_ARROW}		Contour Angled Concave Arrow Side
{WELD_CONTOUR_CONCAVE_ANGLED_OTHER}		Contour Angled Concave Other Side
{WELD_CONTOUR_CONVEX_ARROW}		Contour Convex Arrow Side
{WELD_CONTOUR_CONVEX_OTHER}		Contour Convex Other Side
{WELD_CONTOUR_CONVEX_ANGLED_ARROW}		Contour Angled Convex Arrow Side
{WELD_CONTOUR_CONVEX_ANGLED_OTHER}		Contour Angled Convex Other Side
{WELD_MELT_ARROW}		Melt Arrow Side

Table 7 – Special Symbols

Example:




```
<Texts lineHeight="4.5" n="5" fontIndex="1">
  <Text>
    <Data>{POSITION}</Data>
    <XY>0 0</XY>
  </Text>
  <Text>
    <Data>0.75</Data>
    <XY>1.64658 0</XY>
  </Text>
  <Text>
    <Data>A</Data>
    <XY>3.80537 0</XY>
  </Text>
  <Text>
    <Data>B</Data>
    <XY>4.79375 0</XY>
  </Text>
  <Text>
    <Data>C</Data>
    <XY>5.85390 0</XY>
  </Text>
```

```
</Texts>
```

7.5.7.4 Annotation View

The Annotation View defines an annotation view specifying a default text direction and normal vector of the visualization planes for assigned 3D annotations, as seen in Figure 176. All Annotation Views are collected in the AnnotationViewSet, which is an *element* of the ViewSet. Each 3D annotation intended to be visualized shall be assigned to an annotation view. One 3D annotation may be assigned to only one annotation view; however, one or many 3D annotations may be assigned to one annotation view.

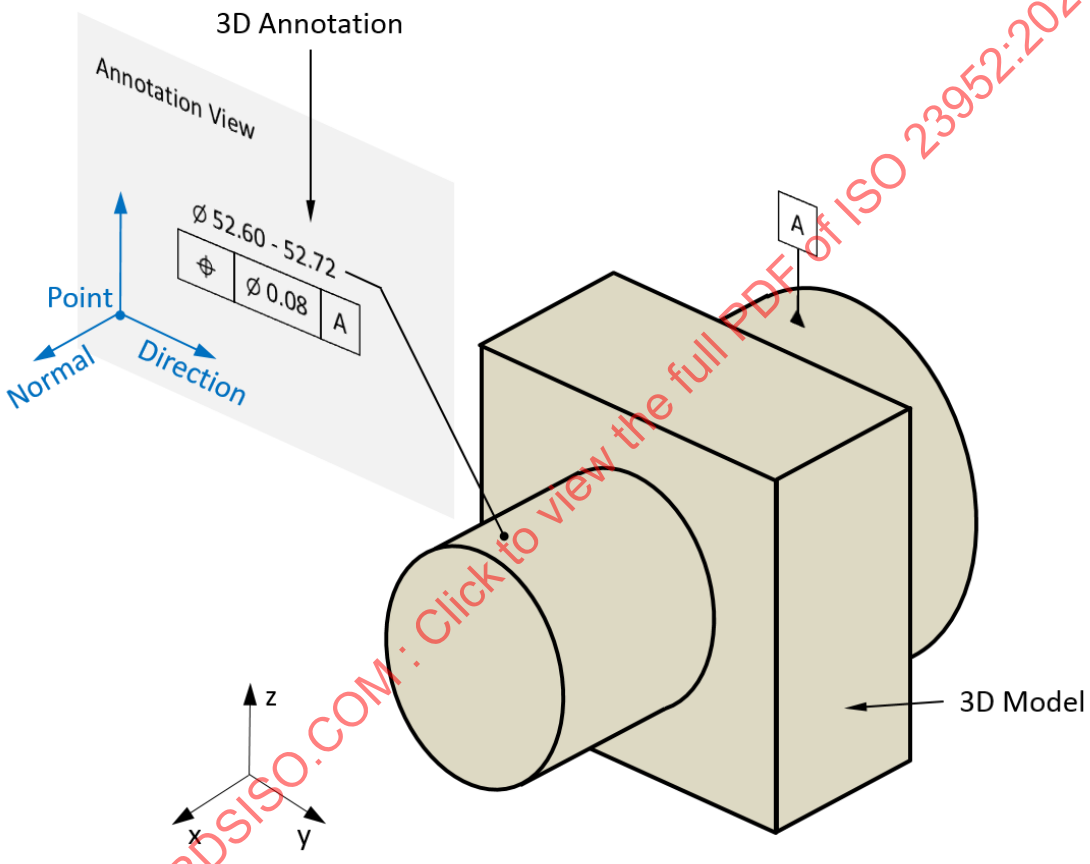


Figure 176 – Annotation View

:

Field Name	Data Type	Description
@id	QIFIdType	The unique model entity identifier.
@label	xs:string	The annotation view label.
Normal	UnitVectorType	The unit normal vector of the visualization plane.
Direction	UnitVectorType	The default text direction defined by a unit vector perpendicular to the normal vector.

Example:


```
<AnnotationView id="3404">
  <Normal>0 0 1</Normal>
  <Direction>1 0 0</Direction>
</AnnotationView>
```

7.5.7.5 PMI display information

The PMIDisplay defines a block of display data used for visualization of a 3D annotation, like those seen in Figure 177 and Figure 178. All PMI display information entities are collected in the PMIDisplaySet, which is an *element* of the VisualizationSet. Visualization plane is defined by Annotation View.

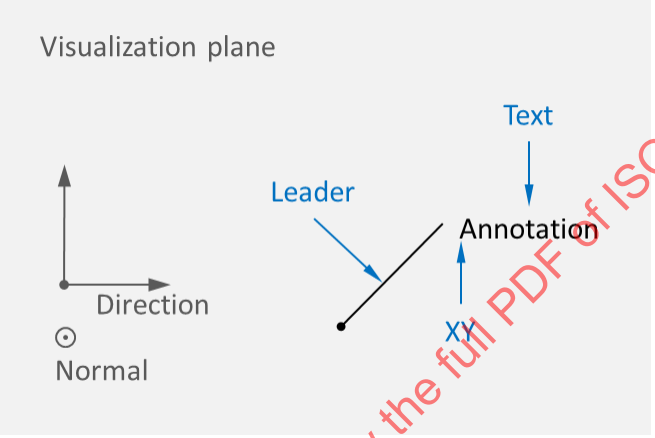


Figure 177 – Text

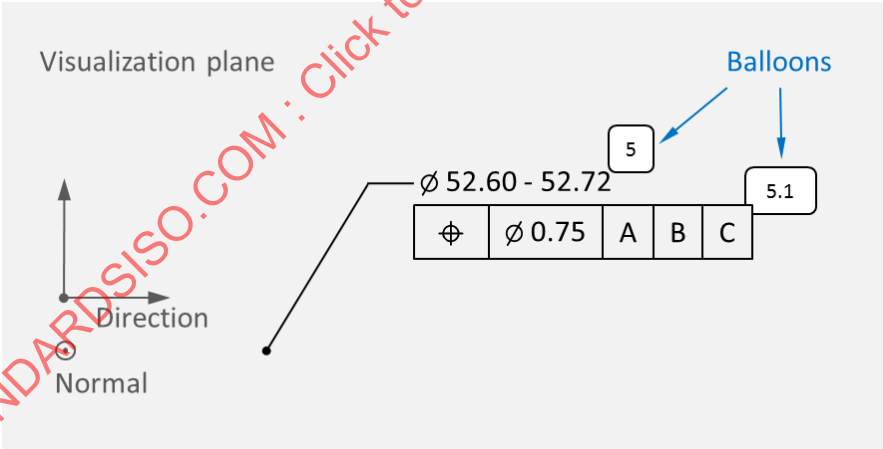


Figure 178 – Balloons

Fields:

Field Name	Data Type	Description
Color	ColorType	The RGB color type is a triplet of integer numbers: the red-component, the green-component and the blue-component.
Plane	PlanePMIDisplayType	A visualization plane defined by annotation view.

Plane/AnnotationViewId	ElementReferenceType	The annotation view identifier.
Plane/Point	PointType	An origin point of the plane.
Plane/Direction	UnitVectorType	The direction of the PMI, different from the default direction.
Texts	TextsType	A set of 3D annotation text lines.
Texts/@fontIndex	xs:unsignedInt	An index identifying a font within the FontSet table.
Texts/@lineHeight	xs:double	A height of the text line defined in the model units.
Texts/Text	TextType	A string.
Leader or LeaderExtend or LeaderCircular or LeaderDoubleHead or LeaderDoubleHeadCircular or LeaderDoubleHeadExtend	LeaderType or LeaderExtendType or LeaderCircularType or LeaderDoubleHeadType or LeaderDoubleHeadCircularType or LeaderDoubleHeadExtendType	Information about a leader type.
WitnessLines	WitnessLinesType	Witness lines.
Frames	FramesType	The visualization frames.
Graphics	GraphicsType	The additional visualization data. It allows specifying user-defined PMI data that have no semantic representation, or if the semantic representation cannot be written into QIF file.
Reference	ElementReferenceFullType	Reference to an annotation entity.

Example:

```
<PMIDisplay>
  <Plane>
    <AnnotationViewId>
      <Id>3404</Id>
    <AnnotationViewId>
    <Origin>282.215308328645 -154.21715304523 3.53553390593274</Origin>
  </Plane>
  <Texts lineHeight="4.5" n="3" fontIndex="1">
    <Text>
      <Data>{PERPENDICULARITY}</Data>
      <XY>0 0</XY>
    </Text>
    <Text>
      <Data>1.5</Data>
      <XY>1.64658203125 0</XY>
```



```
    </Text>
    <Text>
      <Data>A</Data>
      <XY>3.37275390625 0</XY>
    </Text>
  </Texts>
  <LeaderExtend>
    <StartPoint>-94.734069824219 -102.512253206787</StartPoint>
    <EndPoint>-10.8 2.7</EndPoint>
    <HeadForm>DOT_FILLED</HeadForm>
    <HeadHeight>3.6</HeadHeight>
    <PointExtension>-1.8 2.7</PointExtension>
  </LeaderExtend>
  <Frames n="3">
    <FrameRectangular>
      <XY>-0.2 -0.2</XY>
      <Width>1.64658203125</Width>
      <Height>1</Height>
    </FrameRectangular>
    <FrameRectangular>
      <XY>1.44658203125 -0.2</XY>
      <Width>1.726171875</Width>
      <Height>1</Height>
    </FrameRectangular>
    <FrameRectangular>
      <XY>3.17275390625 -0.2</XY>
      <Width>0.98837890625</Width>
      <Height>1</Height>
    </FrameRectangular>
  </Frames>
  <Reference>
    <Id>2320</Id>
  </Reference>
</PMIDisplay>
```

7.5.7.5.1 *Leader*

Leader describes a leader of a 3D annotation, as seen in Figure 179.

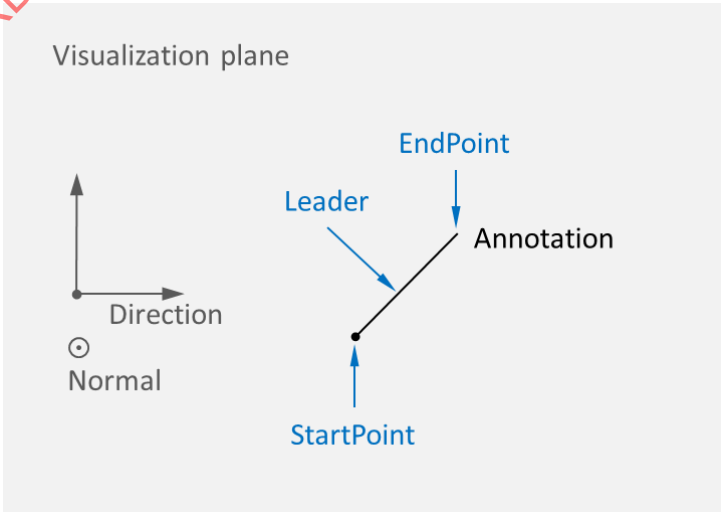


Figure 179 – Leader

Fields:

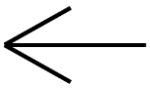
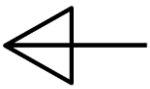
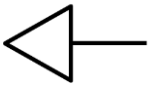
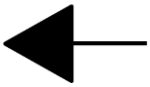
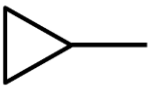
Field Name	Data Type	Description
StartPoint	Point2dSimpleType	The beginning point of the 2D line segment.
EndPoint	Point2dSimpleType	The ending point of the 2D line segment.
HeadForm	LeaderTypeEnumType	A form of the leader head.
HeadHeight	xs:double	A size of the leader head.

Example:

```
<Leader>
  <StartPoint>2.647 -36.261</StartPoint>
  <EndPoint>2.647 -1.800</EndPoint>
  <HeadForm>TRIANGLE_FILLED</HeadForm>
  <HeadHeight>3.600</HeadHeight>
</Leader>
```

7.5.7.5.2 *Leader head types*

LeaderHeadFormEnumType describes forms of the leader head.

Leader type	Image
NONE	
ARROW_OPEN	
ARROW_UNFILLED	
ARROW_BLANKED	
ARROW_FILLED	
TRIANGLE_BLANKED	
TRIANGLE _FILLED	

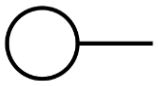
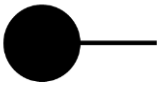
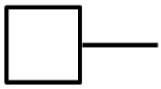
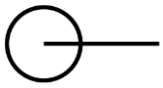
DOT_BLANKED	
DOT_FILLED	
BOX_BLANKED	
BOX_FILLED	
DIMENSION_ORIGIN	
SYMBOL_SLASH	
SYMBOL_INTEGRAL	
SYMBOL_CROSS	

Table 8 – Leader Head Types

7.5.7.5.3 *Double head leader*

LeaderDoubleHead describes a double head leader, as described in Figure 180.

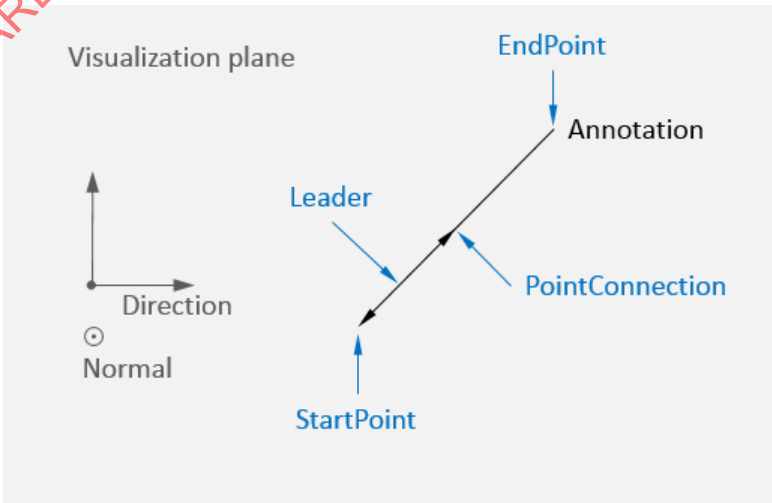


Figure 180 – Double head leader

Fields:

Field Name	Data Type	Description
StartPoint	Point2dSimpleType	The beginning point of the 2D line segment.
EndPoint	Point2dSimpleType	The ending point of the 2D line segment.
HeadForm	LeaderTypeEnumType	The first head form.
HeadHeight	xs:double	A size of the leader head.
HeadForm2	LeaderTypeEnumType	The second head form.
PointConnection	Point2dSimpleType	This field specifies 2D coordinates of the connection point.

Example:

```
<LeaderDoubleHead>
  <StartPoint>-73.225 2.066</StartPoint>
  <EndPoint>0 2.066</EndPoint>
  <HeadForm>ARROW_FILLED</HeadForm>
  <HeadHeight>4.5</HeadHeight>
  <HeadForm2>ARROW_FILLED</HeadForm2>
  <PointConnection>-38.225 2.066</PointConnection>
</LeaderDoubleHead>
```

7.5.7.5.4 *Extended leader*

LeaderExtend describes an extended leader, which has a break point and consists of two segments, as seen in Figure 181.

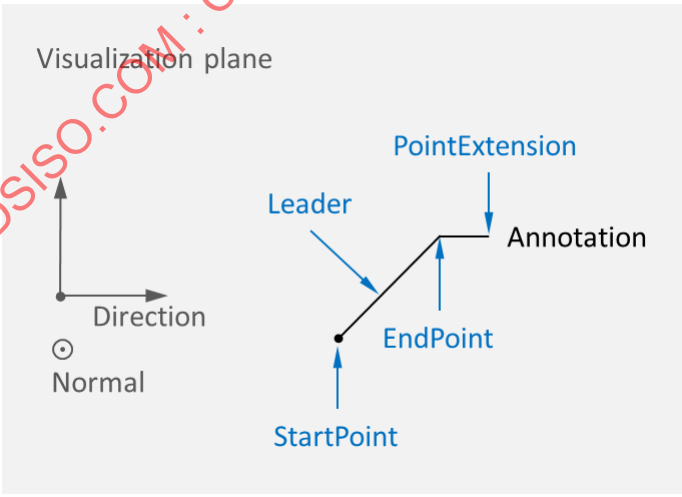


Figure 181 – Extended leader

Fields:

Field Name	Data Type	Description
StartPoint	Point2dSimpleType	The beginning point of the 2D line segment.

EndPoint	Point2dSimpleType	The ending point of the 2D line segment.
HeadForm	LeaderTypeEnumType	A form of the leader head.
HeadHeight	xs:double	A size of the leader head.
PointExtension	Point2dSimpleType	This field specifies 2D coordinates of the extension point.
Modifier	LeaderModifierEnumType	A type of leader modifier.

Example:

```
<LeaderExtend>
  <StartPoint>-94.73 -102.51</StartPoint>
  <EndPoint>-10.80 2.70</EndPoint>
  <HeadForm>DOT_FILLED</HeadForm>
  <HeadHeight>3.6</HeadHeight>
  <PointExtension>-1.80 2.70</PointExtension>
  <Modifier>ALL_AROUND</Modifier>
</LeaderExtend>
```

7.5.7.5.5 *Double head extended leader*

LeaderDoubleHeadExtend describes a double head extended leader, which has a break point and consists of two segments, as seen in Figure 182.

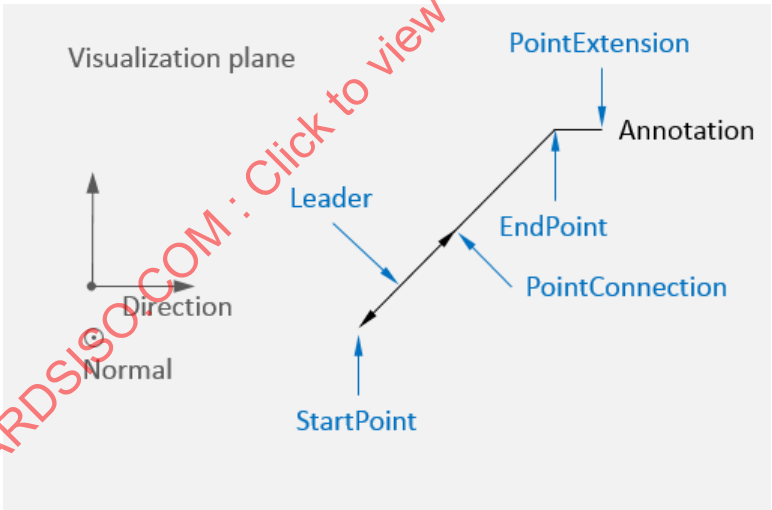


Figure 182 – Double head extended leader

Fields:

Field Name	Data Type	Description
StartPoint	Point2dSimpleType	The beginning point of the 2D line segment.
EndPoint	Point2dSimpleType	The ending point of the 2D line segment.
HeadForm	LeaderTypeEnumType	The first head form.
HeadHeight	xs:double	A size of the leader head.

HeadForm2	LeaderTypeEnumType	The second head form.
PointConnection	Point2dSimpleType	This field specifies 2D coordinates of the connection point.
PointExtension	Point2dSimpleType	This field specifies 2D coordinates of the extension point.

Example:

```
<LeaderDoubleHeadExtend>
  <StartPoint>-73.225 2.066</StartPoint>
  <EndPoint>0 2.066</EndPoint>
  <HeadForm>ARROW_FILLED</HeadForm>
  <HeadHeight>4.5</HeadHeight>
  <HeadForm2>ARROW_FILLED</HeadForm2>
  <PointConnection>-38.225 2.066</PointConnection>
  <PointExtension>15.000 10.000</PointExtension>
</LeaderDoubleHeadExtend>
```

7.5.7.5.6 *Circular leader*

LeaderCircular describes a circular leader that is less than a semicircle, as seen in Figure 183.

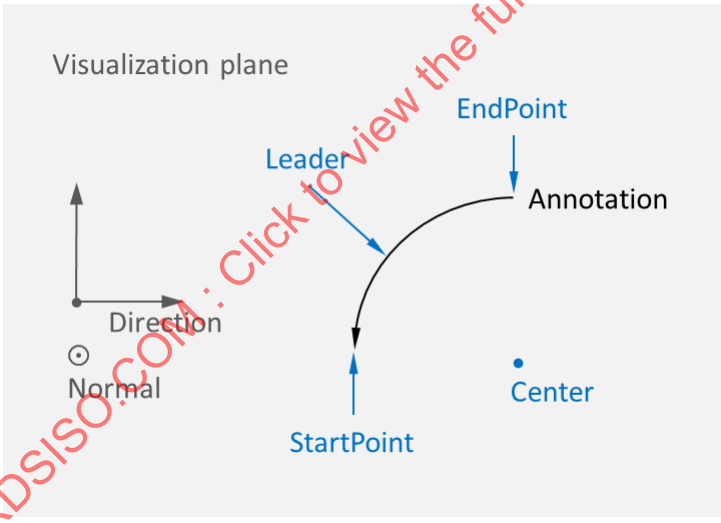


Figure 183 – Circular leader

Fields:

Field Name	Data Type	Description
StartPoint	Point2dSimpleType	The beginning point of the 2D line segment.
EndPoint	Point2dSimpleType	The ending point of the 2D line segment.
HeadForm	LeaderTypeEnumType	The first head form.
HeadHeight	xs:double	A size of the leader head.
Center	Point2dSimpleType	This field specifies 2D coordinates of the leader central point.

Example:

```
<LeaderCircular>
  <StartPoint>0.0 0.0</StartPoint>
  <EndPoint>10.0 10.0</EndPoint>
  <HeadForm>ARROW_FILLED</HeadForm>
  <HeadHeight>1.0</HeadHeight>
  <Center>10.0 0.0</Center>
</LeaderCircular>
```

7.5.7.5.7 *Double head circular leader*

LeaderDoubleHeadCircular describes a double head circular leader, as seen in Figure 184.

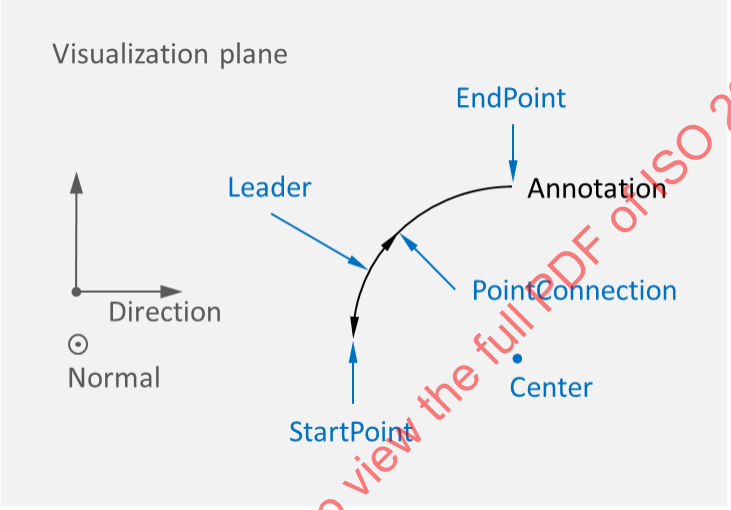


Figure 184 – Double head circular leader

Fields:

Field Name	Data Type	Description
StartPoint	Point2dSimpleType	The beginning point of the 2D line segment.
EndPoint	Point2dSimpleType	The ending point of the 2D line segment.
HeadForm	LeaderTypeEnumType	The first head form.
HeadHeight	xs:double	A size of the leader head.
HeadForm2	LeaderTypeEnumType	The second head form.
PointConnection	Point2dSimpleType	This field specifies 2D coordinates of the connection point.
Center	Point2dSimpleType	This field specifies 2D coordinates of the leader central point.

Example:

```
<LeaderDoubleHeadCircular>
  <StartPoint>0.0 0.0</StartPoint>
  <EndPoint>10.0 10.0</EndPoint>
  <HeadForm>ARROW_FILLED</HeadForm>
```

```
  <HeadHeight>1.0</HeadHeight>
  <HeadForm2>ARROW_FILLED</HeadForm2>
  <PointConnection>2.93 7.07</PointConnection>
  <Center>10.0 0.0</Center>
</LeaderDoubleHeadCircular>
```

7.5.7.5.8 *Witness Lines*

WitnessLines describes witness lines, as seen in Figure 185 and Figure 186.

7.5.7.5.8.1 Linear Witness Lines

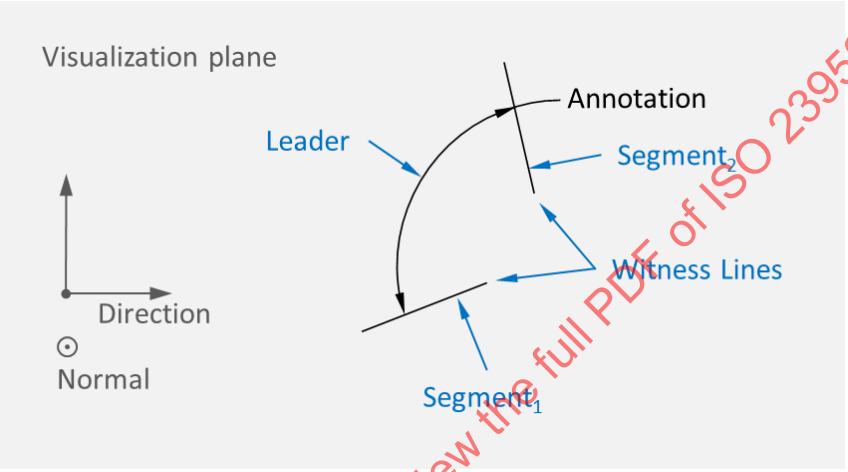


Figure 185 – Witness Lines

Fields:

Field Name	Data Type	Description
@width	xs:double	The width of the witness lines in points.
Segment1/StartPoint	Point2dSimpleType	The beginning point of the first 2D line segment.
Segment1/EndPoint	Point2dSimpleType	The ending point of the first 2D line segment.
ZextensionPoint1	PointSimpleType	The optional ZextensionPoint1 *element* is an extension point of the first witness line; it lies on the feature in cases when the annotation plane does not intersect the feature.
Segment2/StartPoint	Point2dSimpleType	The beginning point of the second 2D line segment.
Segment2/EndPoint	Point2dSimpleType	The ending point of the second 2D line segment.
ZextensionPoint2	PointSimpleType	The optional ZextensionPoint2 *element* is an extension point of the second witness line; it lies on the feature in cases when the annotation plane does not intersect the feature.

Example:

```
<WitnessLines width="1">
```



```
  <Segment1>
    <StartPoint>-73.225 87.343</StartPoint>
    <EndPoint>-73.225 -0.358</EndPoint>
  </Segment1>
  <ZextensionPoint1>-73.225 0 22.344</ZextensionPoint1>

  <Segment2>
    <StartPoint>-38.225 87.343</StartPoint>
    <EndPoint>-38.225 -0.358</EndPoint>
  </Segment2>
</WitnessLines>
```

7.5.7.5.8.2 Circular Witness Line

A Circular Witness Line must have an angle of less than or equal to 180 degrees, as seen in Figure 186.

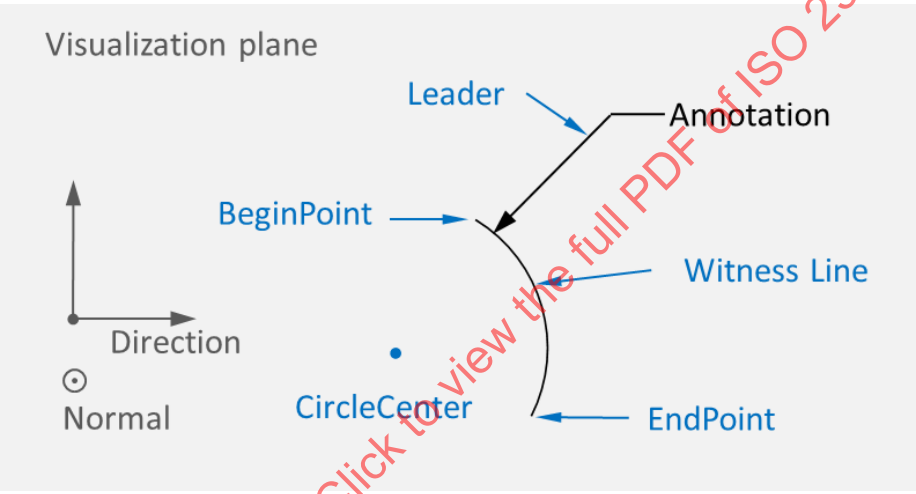


Figure 186 – Circular Witness Line

Fields:

Field Name	Data Type	Description
@width	xs:double	The width of the witness lines in points.
BeginPoint	LineSegment2dType	The BeginPoint *element* is the begin point of the circular witness line.
EndPoint	LineSegment2dType	The EndPoint *element* is the end point of the circular witness line.
CircleCenter	LineSegment2dType	The CircleCenter *element* is the center of the circular witness line.
CircleRadius	DoublePositiveType	The CircleRadius *element* is the radius of the circular witness line.

Example:

```
<WitnessLines width="1">
  <BeginPoint>0.0 0.0</BeginPoint>
  <EndPoint>10.0 10.0</EndPoint>
```

```
  <CircleCenter>10.0 0.0</CircleCenter>
  <CircleRadius>2.5</CircleRadius>
</WitnessLines>
```

7.5.7.5.9 *Rectangular frame*

FrameRectangular describes a rectangular frame, as seen in Figure 187.

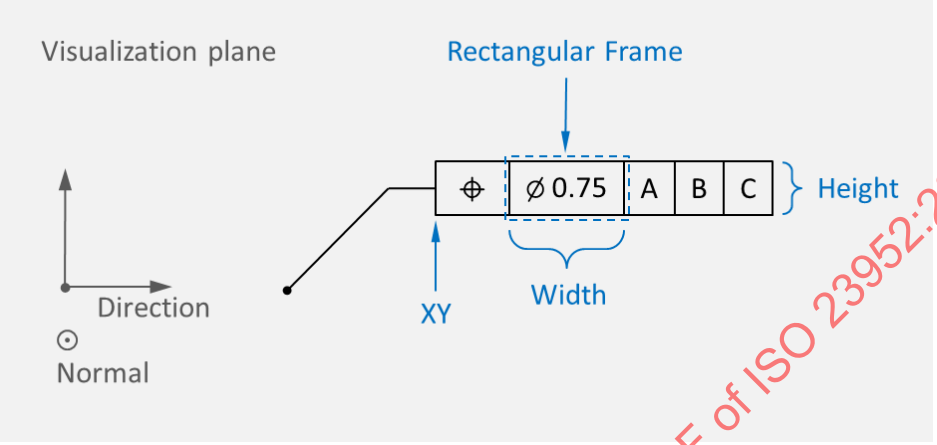


Figure 187 – Rectangular frame

Fields:

Field Name	Data Type	Description
XY	Point2dSimpleType	2D coordinates of the frame left bottom corner point (the anchor point of the rectangular frame).
Width	xs:double	The frame width.
Height	xs:double	The frame height.

Example:

```
<FrameRectangular>
  <XY>-0.2 -0.2</XY>
  <Width>1.6</Width>
  <Height>1.0</Height>
</FrameRectangular>
```

7.5.7.5.10 *Circular frame*

FrameCircularType describes a circular frame, which is normally used for visualization of datum targets, as seen in Figure 188.

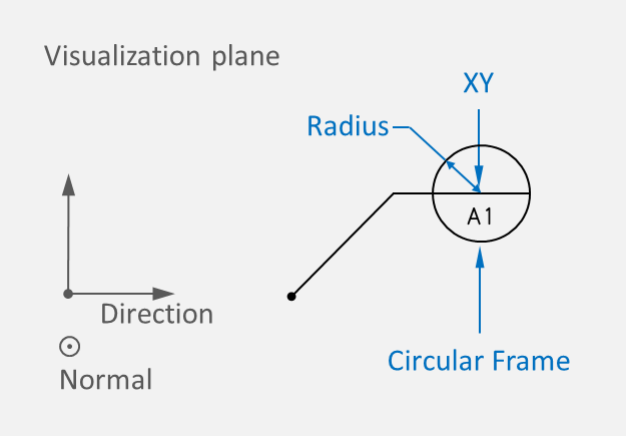


Figure 188 – Circular frame

Fields:

Field Name	Data Type	Description
@crossed	xs:boolean	The optional crossed attribute shows if the frame shall be crossed with the middle line that separates the circular frame into halves (top and bottom).
XY	Point2dSimpleType	2D coordinates of the frame center (the anchor point of the circular frame).
Radius	xs:double	The frame radius.

Example:

```
<FrameCircular>
  <XY>-0.2 -0.2</XY>
  <Radius>7.3</Radius>
</FrameCircular>
```

7.5.7.5.11 *Flag frame*

FrameFlag describes a flag frame, which is normally used for visualization of flag notes, as seen in Figure 189.

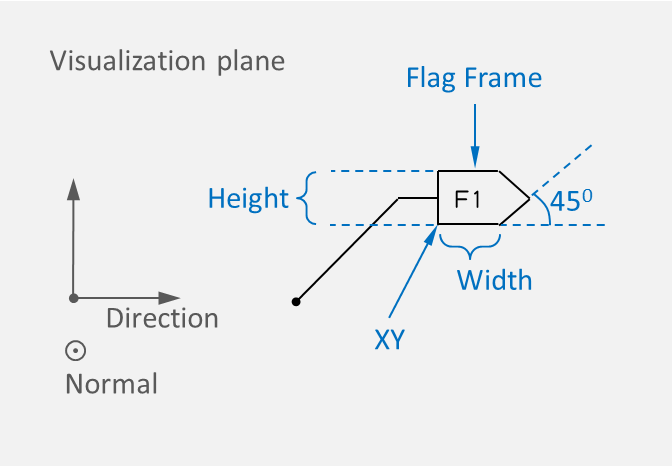


Figure 189 – Flag frame

Fields:

Field Name	Data Type	Description
@right	xs:boolean	This attribute shows if the flag frame has the triangle *element* at the right side.
XY	Point2dSimpleType	2D coordinates of the frame left bottom corner point (the anchor point of the flag frame).
Width	xs:double	The frame width.
Height	xs:double	The frame height.

Example:

```
<FrameFlag right="1">
  <XY>-0.2 -0.2</XY>
  <Width>19.2</Width>
  <Height>4.1</Height>
</FrameFlag>
```

7.5.7.5.12 *Triangular form frame*

FrameTriangle defines a triangular form frame as seen in Figure 190.

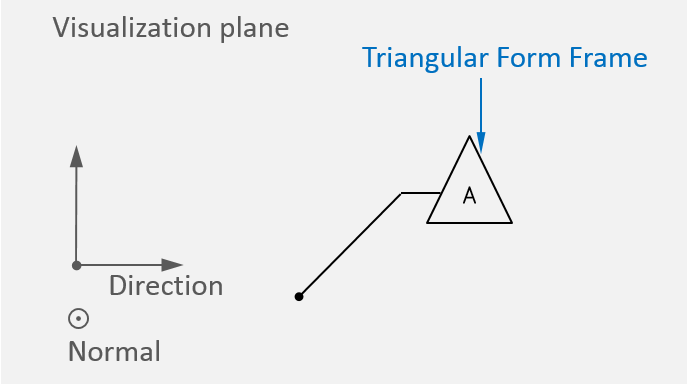


Figure 190 – Triangular form frame

Fields:

Field Name	Data Type	Description
Points	ArrayPoint2dType	A 2D polyline that describes the frame shape. The array includes 3 points.

Example:

```
<FrameTriangle>
  <Point>0.294677734375 1.33232421874998</Point>
  <Point>1.03935546875 -0.199999999999977</Point>
  <Point>-0.449999999999998 -0.199999999999977</Point>
</FrameTriangle>
```

7.5.7.5.13 *Pentagonal form frame*

FramePentagonal defines a pentagonal form frame, as seen in Figure 191.

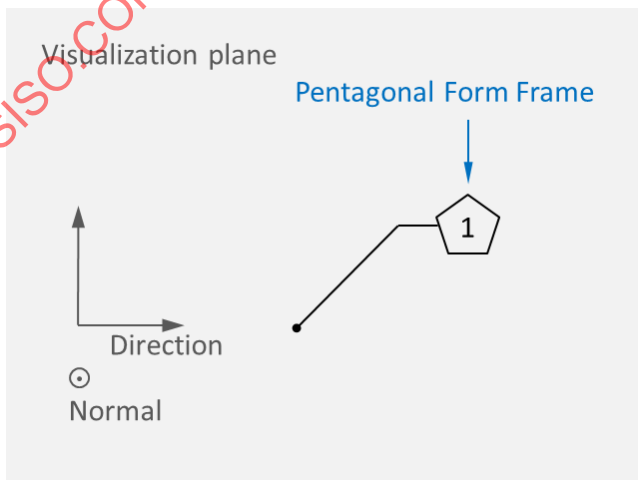


Figure 191 – Pentagonal form frame

Fields:

Field Name	Data Type	Description

Points	ArrayPoint2dType	A 2D polyline that describes the frame shape. The array includes 5 points.

Example:

```
<FramePentagonal>
  <Point>0.5 0</Point>
  <Point>1.5 0</Point>
  <Point>2 1</Point>
  <Point>1 2</Point>
  <Point>0 1</Point>
</FramePentagonal>
```

7.5.7.5.14 *Hexagonal form frame*

FrameHexagonal defines a hegaxonal form frame, as seen in Figure 192.

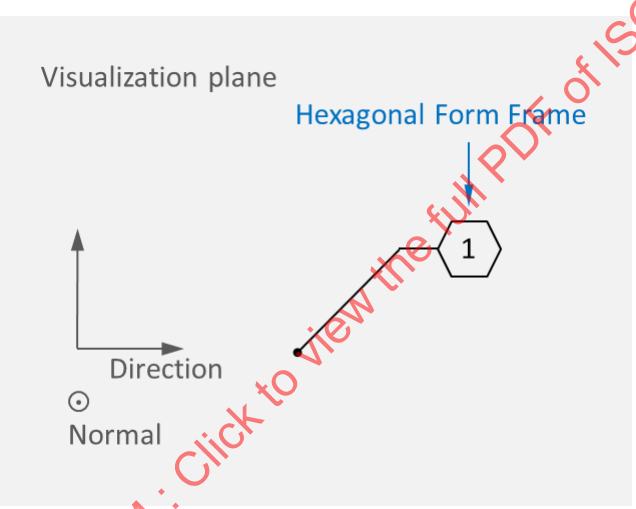


Figure 192 – Hegaxonal form frame

Fields:

Field Name	Data Type	Description
Points	ArrayPoint2dType	A 2D polyline that describes the frame shape. The array includes 6 points.

Example

```
<FrameHexagonal>
  <Point>0.5 0</Point>
  <Point>1.0 0</Point>
  <Point>1.5 0.75</Point>
  <Point>1.0 1.5</Point>
  <Point>0.5 1.5</Point>
  <Point>0 0.75</Point>
</FrameHexagonal>
```

7.5.7.5.15 *Octagonal form frame*

FrameOctagonal defines a hegaxonal form frame, as seen in Figure 193.

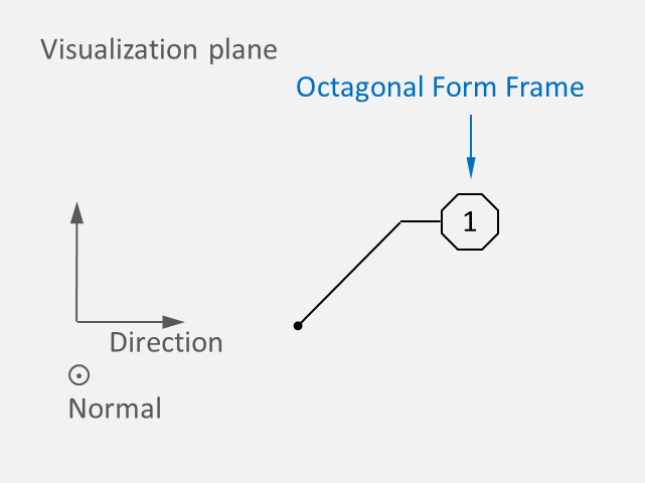


Figure 193 – Octagonal form frame

Fields:

Field Name	Data Type	Description
Points	ArrayPoint2dType	A 2D polyline that describes the frame shape. The array includes 8 points.

Example:

```
<FrameOctagonal>
  <Point>0.5 0</Point>
  <Point>1.5 0</Point>
  <Point>2 0.5</Point>
  <Point>2 1.5</Point>
  <Point>1.5 2</Point>
  <Point>0.5 2</Point>
  <Point>0 1.5</Point>
  <Point>0 0.5</Point>
</FrameOctagonal>
```

7.5.7.5.16 *Weld Symbol form frame*

FrameWeldSymbol defines a weld symbol frame, as seen in Figure 194.

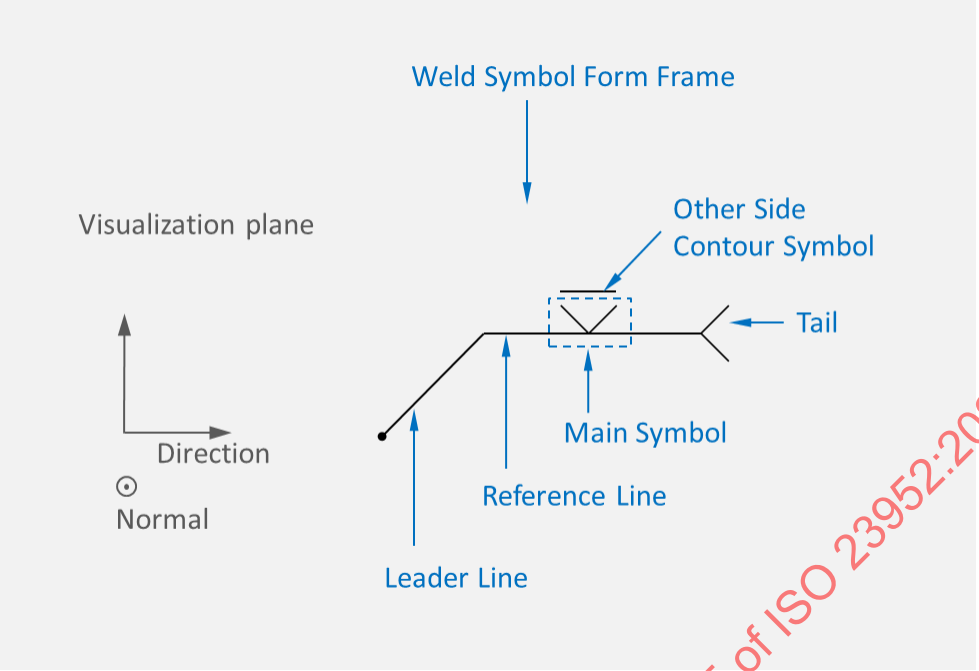


Figure 194 – Weld Symbol frame

Fields:

Field Name	Data Type	Description
ReferenceLineBeginPoint	Point2dSimpleType	A 2D point that defines the begin point of the reference line.
ReferenceLIneEndPoint	Point2dSimpleType	A 2D point that defines the end point of the reference line.
Tail	FrameWeldSymbolTailType	2D points that define the tail of the reference line.
MainSymbol	MainWeldSymbolType	The main symbol that is placed on the reference line.
SupplementarySymbol	WeldSupplementarySymbolType	The supplementary symbol that is placed on the reference line.
ArrowSideContourSymbol	WeldContourSymbolType	The contour symbol that is placed on the arrow side.
OtherSideContourSymbol	WeldContourSymbolType	The contour symbol that is placed on the other side.

Example:

```
<FrameWeldSymbol>
  <ReferenceLineBeginPoint>1.0 1.0</ReferenceLineBeginPoint>
  <ReferenceLineEndPoint>4.0 1.0</ReferenceLineEndPoint>
  <Tail>
    <UpperPoint>4.5 1.5</UpperPoint>
    <LowerPoint>4.5 0.5</LowerPoint>
  </Tail>
  <MainSymbol>
    <Origin>2.0 1</Origin>
```



```
    <Symbol>WELD_FILLET_OTHER_SIDE</Symbol>
  </MainSymbol>
  <OtherSideContourSymbol>
    <Origin>2.0 1.75</Origin>
    <Symbol>WELD_CONTOUR_FLAT</Symbol>
  </OtherSideContourSymbol>
</FrameWeldSymbol>
```

7.5.7.5.17 *Irregular form frame*

FrameIrregular defines an irregular form frame, as seen in Figure 195. A FrameIrregular is a closed shape only if the first and last points in Points are the same.

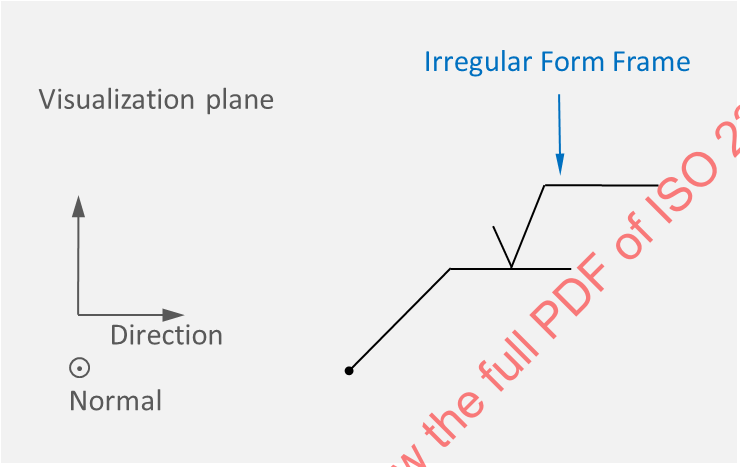


Figure 195 – Irregular form frame

Field Name	Data Type	Description
Points	ArrayPoint2dType	A 2D polyline that describes the frame shape.

Example:

```
<FrameIrregularForm>
  <Points n="4">
    10 1
    10.5 0
    11 2
    14 2
  </Points>
</FrameIrregularForm>
```

7.5.7.5.18 *Graphic presentation*

Graphics defines additional graphical presentation data, as seen in Figure 196. It allows specifying user-defined PMI data that have no semantic representation, or if the semantic representation cannot be written into QIF file.

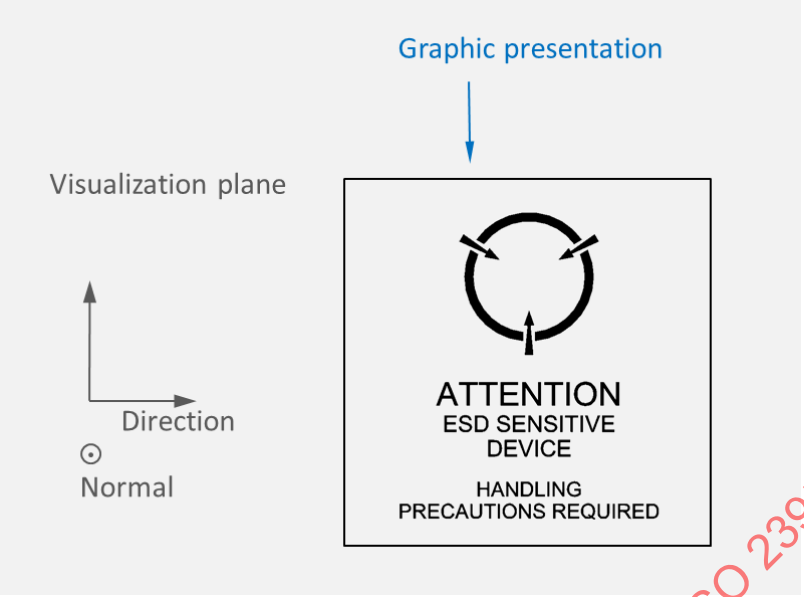


Figure 196 – Graphic presentation

Fields:

Field Name	Data Type	Description
Polylines	Polylines2dType	The set of 2D polylines.
Polylines/Polyline/Points or Polylines/Polyline/PointsBinary	ArrayPoint2dType or ArrayBinaryType	The array of 2D polyline points.
Areas	Areas2dType	The set of 2D areas.
Areas/Area/Loops	Loops2dType	The set of loops defining the area. The first loop is outer, the rest are inner loops.
Areas/Area/Loops/Loop/Points or Areas/Area/Loops/Loop/PointsBinary	ArrayPoint2dType or ArrayBinaryType	The array of 2D points of a closed polyline.
Areas/Area/Triangulation	Triangulation2dType	The area defined in the triangulation form.
Areas/Area/Triangulation/Vertices or Areas/Area/Triangulation/VerticesBinary	ArrayPoint2dType or ArrayBinaryType	The array of 2D vertices of a triangulation.
Areas/Area/Triangulation/Triangles or Areas/Area/Triangulation/TrianglesBinary	ArrayI3Type or ArrayBinaryType	The array of indices of the triangle vertices. The number of the array *element*s corresponds to the number of triangles in the mesh. Each *element* of this array is a triplet of integer numbers: index of the first vertex, index of the second vertex and index of the third vertex. All three vertex indices of a triangle must

		be different and shall lie in the range [0, ..., number of vertices - 1].

Example:

```
<Graphics>
  <Polylines n="1">
    <Polyline>
      <Points n="2">
        25 0
        25 10
      </Points>
    </Polyline>
  </Polylines>
  <Areas n="2">
    <Area>
      <Loops n="2">
        <Loop>
          <Points n="5">
            30 0
            40 0
            40 10
            30 10
            30 0
          </Points>
        </Loop>
        <Loop>
          <Points n="5">
            31 1
            31 9
            39 9
            39 1
            31 1
          </Points>
        </Loop>
      </Loops>
    </Area>
    <Area>
      <Triangulation>
        <Vertices n="4">
          10 0
          20 0
          20 10
          10 10
        </Vertices>
        <Triangles n="2">
          0 1 2
          0 2 3
        </Triangles>
      </Triangulation>
    </Area>
  </Areas>
</Graphics>
```

7.5.7.6 Saved View

SavedView describes a saved view to facilitate presentation of the model and its 3D annotations, as seen in Figure 197.

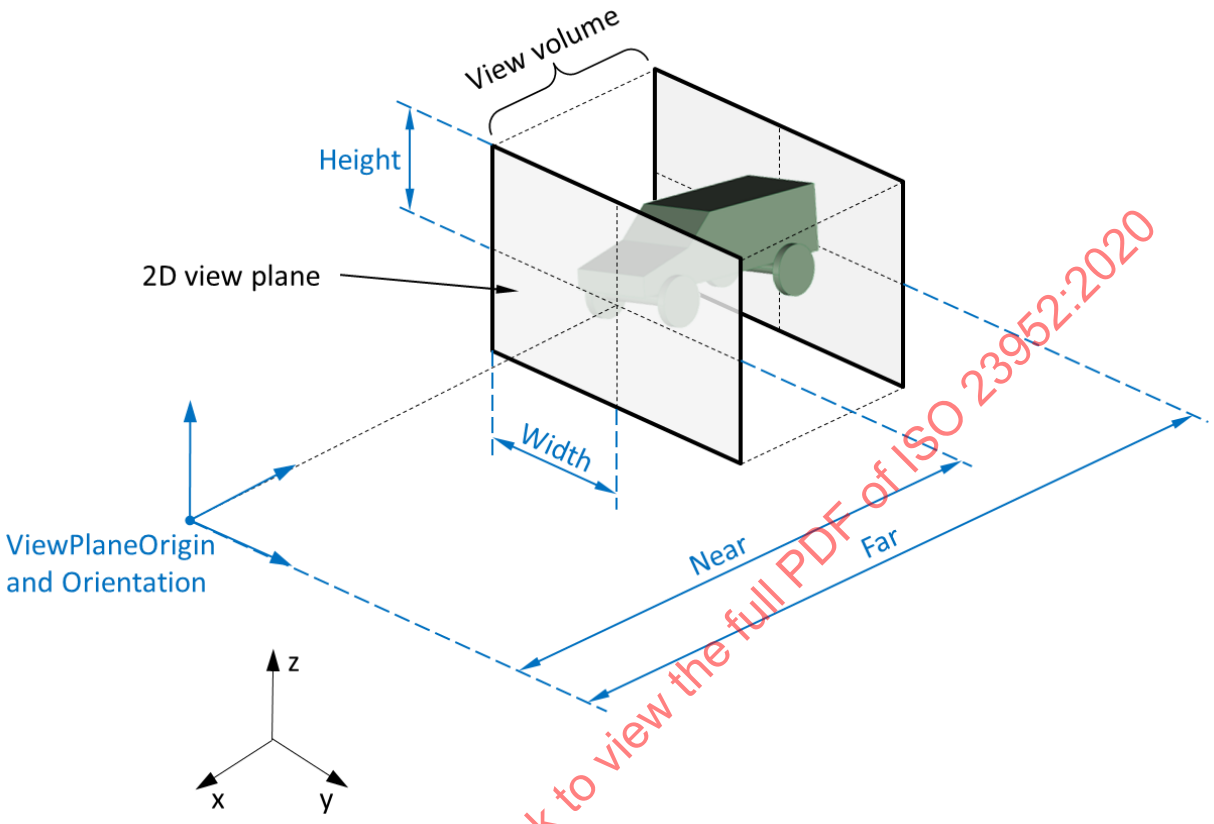


Figure 197 – Saved view

Fields:

Field Name	Data Type	Description
@id	QIFIdType	Unique model entity identifier.
@label	xs:string	Label of the Saved View.
ActiveView	xs:boolean	Defines an active view.
AnnotationVisibleIds	ArrayReferenceFullType	An array of identifiers of model annotations that shall be visible in this saved view.
AnnotationHiddenIds	ArrayReferenceFullType	An array of identifiers of model annotations that shall be hidden in this saved view.
BodyIds	ArrayReferenceFullType	An array of identifiers of model bodies that shall be visible in this saved view. If the BodyIds *element* is absent, then all model bodies are visible in this saved view.
ComponentIds	ArrayReferenceFullType	An array of identifiers of model components that shall be visible in this saved view. If the ComponentIds *element* is absent, then all

		model components are visible in this saved view.
SimplifiedRepresentationId	QIDReferenceActiveType	An identifier of the simplified representation.
ExplodedViewId	QIDReferenceActiveType	An identifier of the exploded view.
DisplayStyleId	QIDReferenceActiveType	An identifier of the display style, defining color, transparency and display form.
ZoneSectionId	QIDReferenceActiveType	An identifier of the zone section, which defines zone section planes, separating the model into parts, and logical operations between them.
CameraIds	ArrayReferenceActiveType	An array of identifiers of orthographic cameras.

Example:

```
<SavedView id="2371" label="MBD_0">
  <ActiveView>true</ActiveView>
  <AnnotationVisibleIds n="5">
    <Id>2325</Id>
    <Id>2328</Id>
    <Id>2331</Id>
    <Id>2334</Id>
    <Id>2337</Id>
  </AnnotationVisibleIds>
  <SimplifiedRepresentationId>2377</SimplifiedRepresentationId>
  <ExplodedViewId>2380</ExplodedViewId>
  <DisplayStyleId>2394</DisplayStyleId>
  <ZoneSectionId>2395</ZoneSectionId>
  <CameraIds n="1">
    <Id>2372</Id>
  </CameraIds>
</SavedView>
```


7.5.7.6.1 *Simplified Representation*

The SimplifiedRepresentation defines a simplified model representation, as seen in Figure 198. All simplified representations are collected in one set SimplifiedRepresentationSet, which is an *element* of the ViewSet.

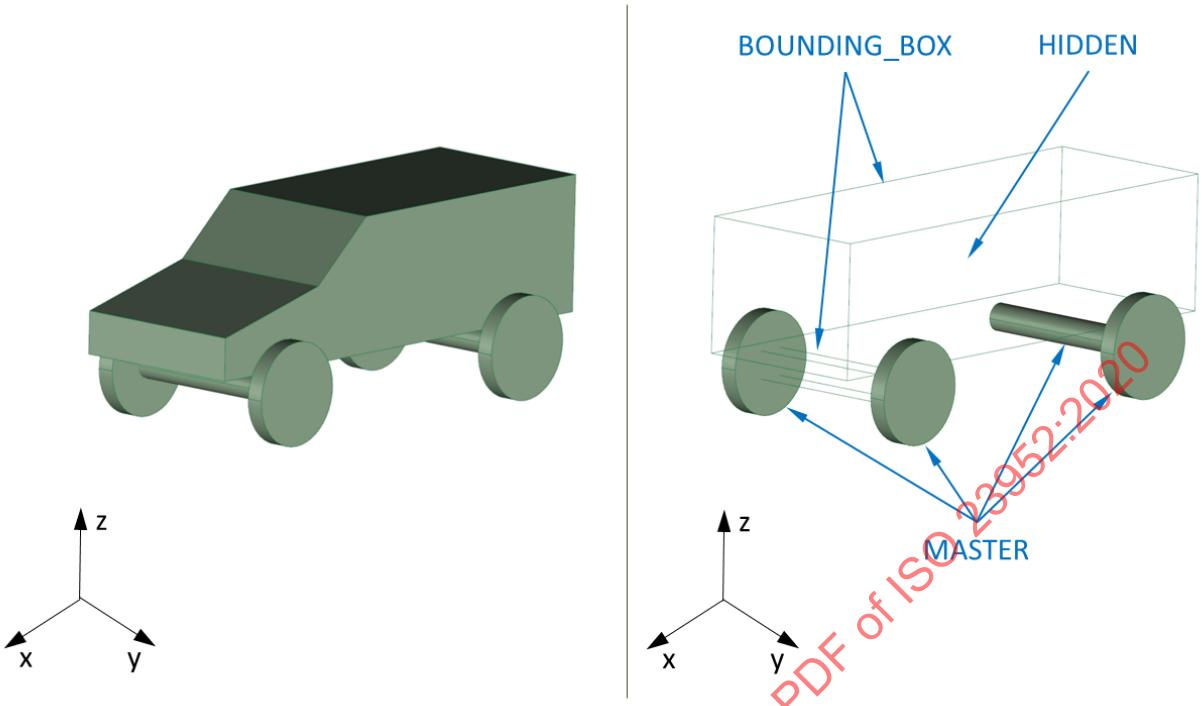


Figure 198 – Simplified Representation

Fields:

Field Name	Data Type	Description
@id	QIFIdType	The unique model entity identifier.
@label	xs:string	The label of the simplified representation.
Form	SimplifiedRepresentationFormEnumType	The default representation form.
SimplifiedRepresentationGroups	SimplifiedRepresentationGroupsType	Groups of components and bodies with defined simplified representation forms.
SimplifiedRepresentationGroups/SimplifiedRepresentationGroup/Form	SimplifiedRepresentationFormEnumType	The representation form defined for the group of components and bodies.
SimplifiedRepresentationGroups/SimplifiedRepresentationGroup/ComponentIds	ArrayReferenceFullType	An array of identifiers of model components.
SimplifiedRepresentationGroups/SimplifiedRepresentationGroup/BodyIds	ArrayReferenceFullType	The array of identifiers of model bodies.

Example:

```
<SimplifiedRepresentationSet n="1">
  <SimplifiedRepresentation id="10000">
```



```
    <Form>MASTER</Form>
  <SimplifiedRepresentationGroups n="2">
    <SimplifiedRepresentationGroup>
      <Form>BOUNDING_BOX</Form>
      <BodyIds n="2">
        <Id>114</Id>
        <Id>127</Id>
      </BodyIds>
    </SimplifiedRepresentationGroup>
    <SimplifiedRepresentationGroup>
      <Form>HIDE</Form>
      <ComponentIds n="1">
        <Id>1242</Id>
      </ComponentIds>
      <BodyIds n="1">
        <Id>128</Id>
      </BodyIds>
    </SimplifiedRepresentationGroup>
  </SimplifiedRepresentationGroups>
  </SimplifiedRepresentation>
</SimplifiedRepresentationSet>
```

7.5.7.6.2 *Exploded View*

The ExplodedViewType defines an exploded view - a sequence of translate and rotate operations applied to the groups of components and bodies showing each object separated from others, as seen in Figure 199. All exploded views are collected in the ExplodedViewSet, which is an *element* of the ViewSet.

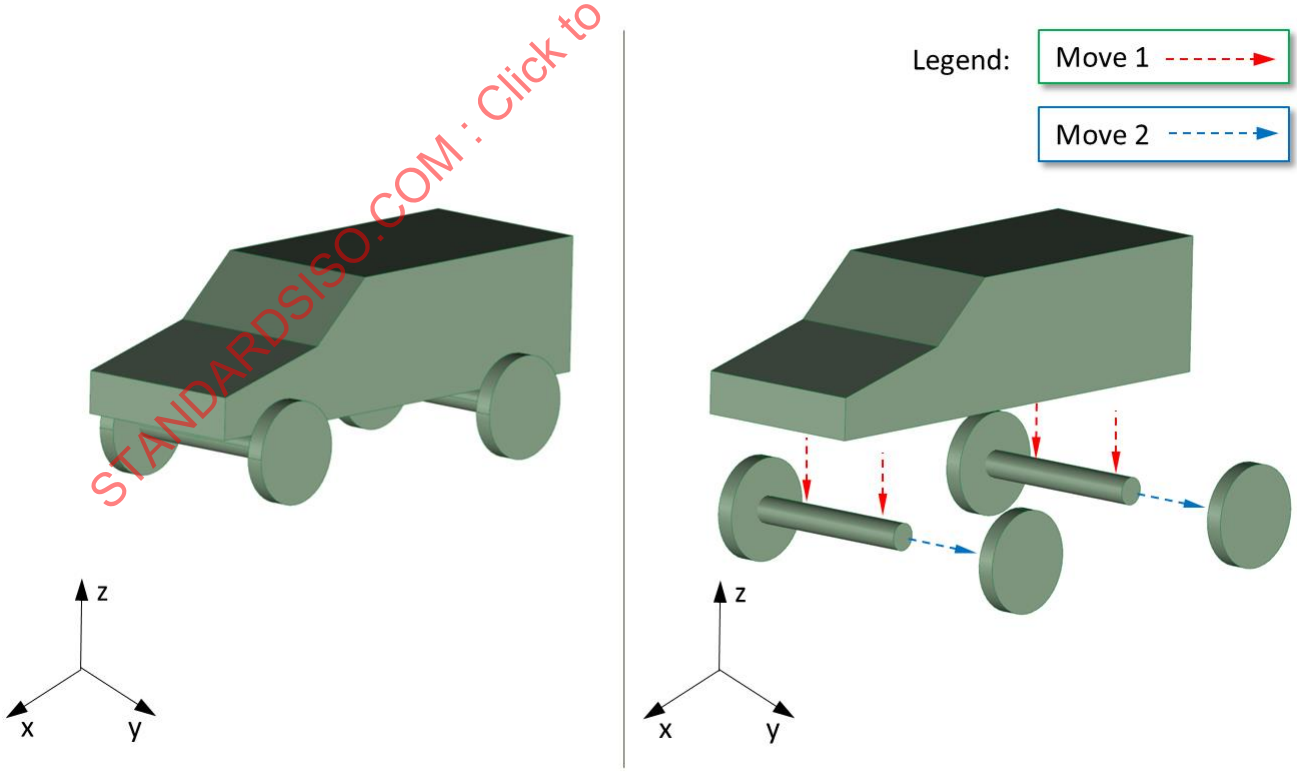


Figure 199 – Exploded View

Fields:

Field Name	Data Type	Description
@id	QIFIdType	The unique model entity identifier.
@label	xs:string	The label of the exploded view.
MoveGroups	ExplodedViewMoveGroupsType	The sequence of translate and rotate operations applied to the groups of components and bodies.
MoveGroups/MoveGroup /Translate	ExplodedViewTranslateType	The translation operator defined by a unit vector and a translation distance.
MoveGroups/MoveGroup /Translate/Direction	UnitVectorType	The translation direction.
MoveGroups/MoveGroup /Translate/Value	LinearValueType	The translation distance.
MoveGroups/MoveGroup /Rotate	ExplodedViewRotateType	The rotation operator defined by a rotation axis and rotation angle.
MoveGroups/MoveGroup /Rotate/Axis	UnitVectorType	The rotation axis.
MoveGroups/MoveGroup /Rotate/Angle	AngularValueType	The rotation angle.
MoveGroups/MoveGroup /ComponentIds	ArrayReferenceFullType	The array of component identifiers.
MoveGroups/MoveGroup /BodyIds	ArrayReferenceFullType	The array of body identifiers.

Example:

```
<ExplodedViewSet n="1">
  <ExplodedView id="10001">
    <MoveGroups n="2">
      <MoveGroup>
        <Translate>
          <Direction>1 0 0</Direction>
          <Value>20</Value>
        </Translate>
        <ComponentIds n="2">
          <Id>123</Id>
          <Id>124</Id>
        </ComponentIds>
      </MoveGroup>
      <MoveGroup>
        <Rotate>
          <Axis>0 1 0</Axis>
          <Angle>45</Angle>
        </Rotate>
        <ComponentIds n="1">
          <Id>125</Id>
        </ComponentIds>
      </MoveGroup>
    </MoveGroups>
  </ExplodedView>
```



```
</ExplodedViewSet>
```

7.5.7.6.3 *Display Style*

The DisplayStyleType defines a display style of saved view *element*s, such as color, transparency and visualization form, as seen in Figure 200. All display styles are collected in the DisplayStyleSet, which is an *element* of the ViewSet.

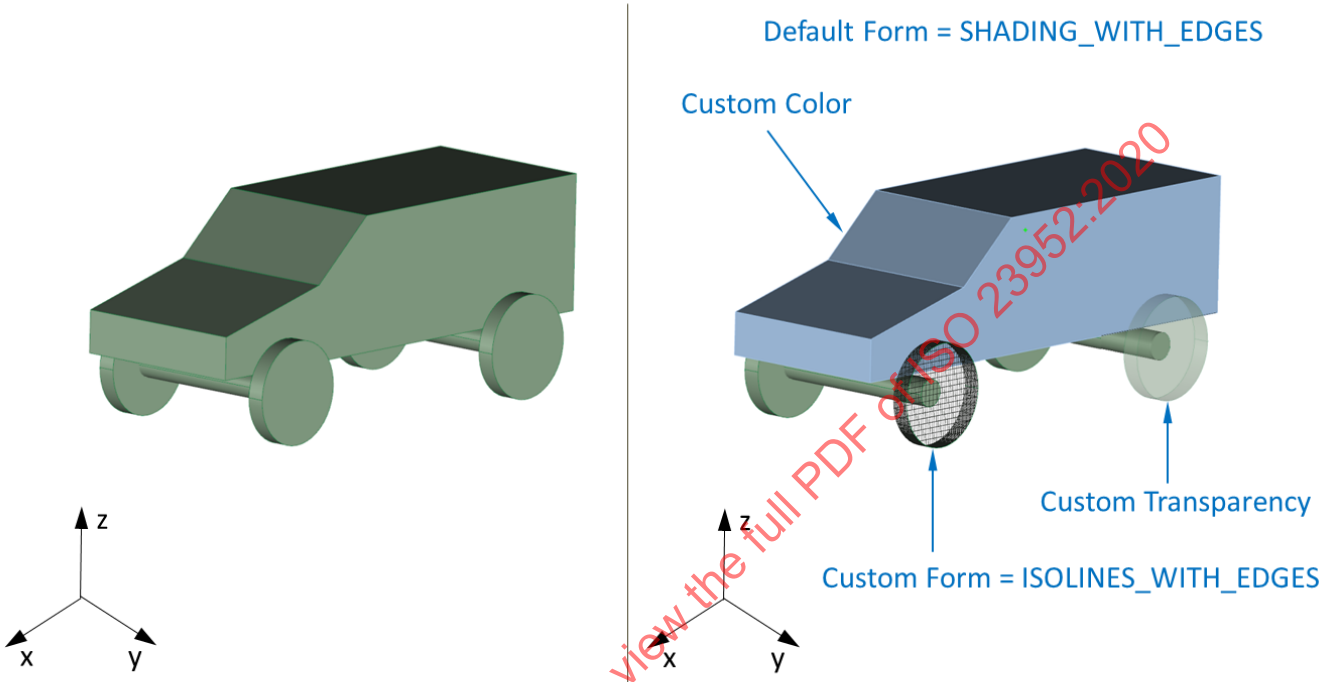


Figure 200 – Display Style

Fields:

Field Name	Data Type	Description
@id	QIFIdType	The unique model entity identifier.
@label	xs:string	The label of the display style.
Mode/Form	DisplayStyleFormEnumType	The default visualization mode.
Mode/Color	ColorType	The default drawing color.
Mode/Transparency	TransparencyType	The default transparency value.
DisplayStyleGroups	DisplayStyleGroupsType	The array of the groups of *element*s with a defined display style.
DisplayStyleGroups/DisplayStypeGroup/Mode	DisplayStyleModeType	The display style for the group of *element*s.
DisplayStyleGroups/DisplayStypeGroup/Mode/Form	DisplayStyleFormEnumType	The visualization mode.
DisplayStyleGroups/DisplayStypeGroup/Mode/Color	ColorType	The drawing color.

DisplayStyleGroups/DisplayStypeGroup/Mode/Transparency	TransparencyType	The transparency value.
DisplayStyleGroups/DisplayStypeGroup/ComponentIds	ArrayReferenceFullType	The array of identifiers of model components.
DisplayStyleGroups/DisplayStypeGroup/BodyIds	ArrayReferenceFullType	The array of identifiers of model bodies.

Example:

```
<DisplayStyleSet n="1">
  <DisplayStyle id="10002">
    <Mode>
      <Form>SHADING_WITH_EDGES</Form>
    </Mode>
    <DisplayStyleGroups n="2">
      <DisplayStyleGroup>
        <Mode>
          <Form>WIREFRAME</Form>
          <Color>255 0 0</Color>
        </Mode>
        <BodyIds n="2">
          <Id>128</Id>
          <Id>129</Id>
        </BodyIds>
      </DisplayStyleGroup>
      <DisplayStyleGroup>
        <Mode>
          <Form>SHADING_WITH_EDGES</Form>
          <Transparency>0.5</Transparency>
        </Mode>
        <ComponentIds n="1">
          <Id>12834</Id>
        </ComponentIds>
      </DisplayStyleGroup>
    </DisplayStyleGroups>
  </DisplayStyle>
</DisplayStyleSet>
```


7.5.7.6.4 *Zone Section*

The ZoneSectionType defines a zone section – a set of section planes that separates the model into parts and the logical operations between them, as seen in Figure 201 through Figure 207. Figure 206 show an XML code snippet of a Zone Section with a logical operations tree.

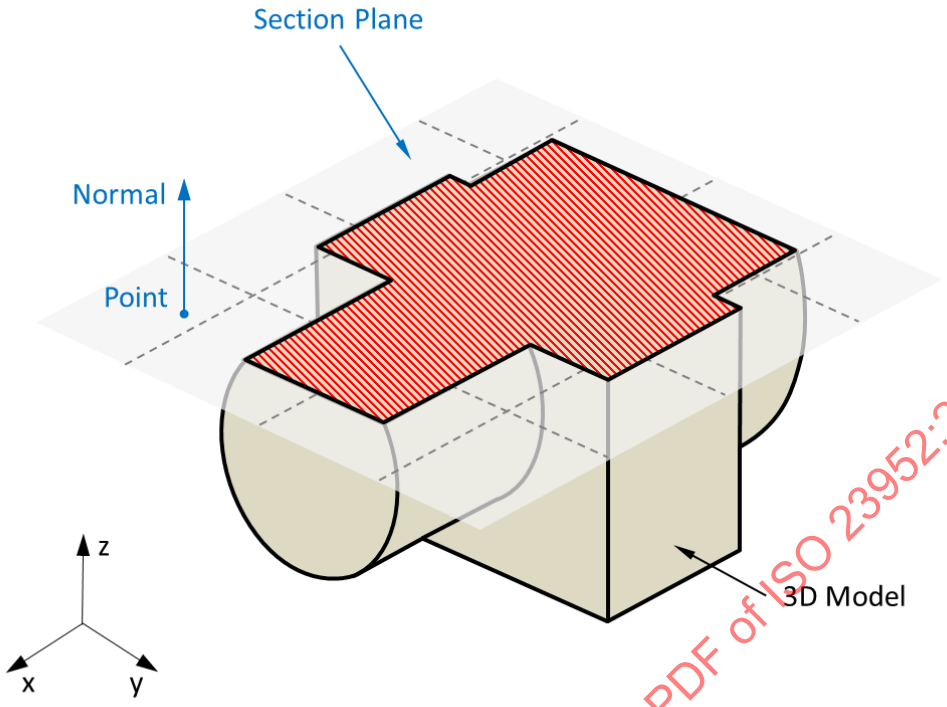


Figure 201 – Zone Section with one section plane

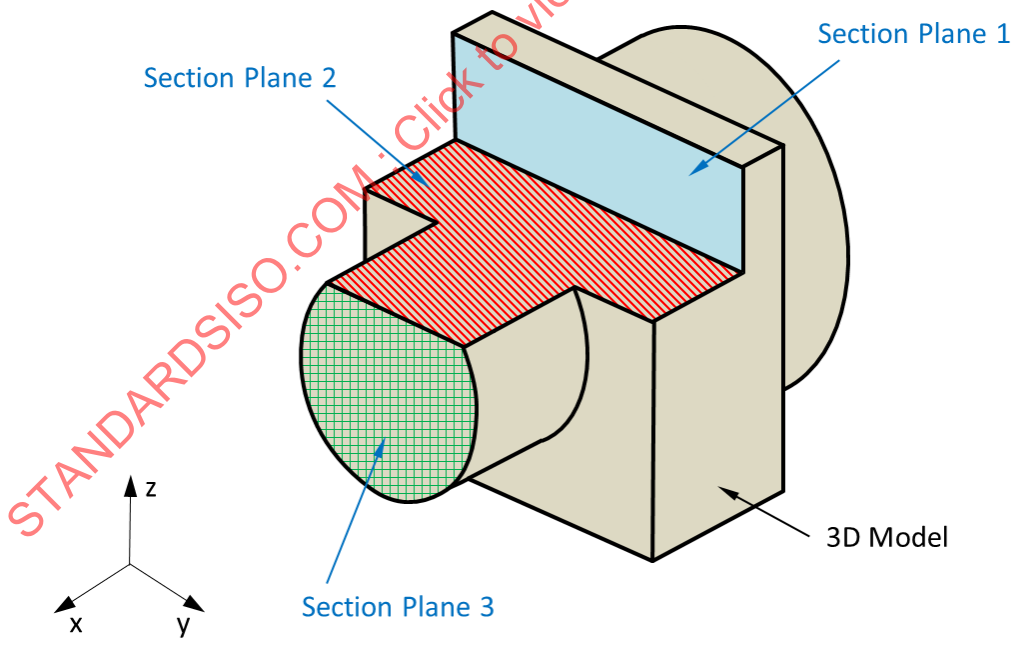


Figure 202 – Zone Section with three section planes

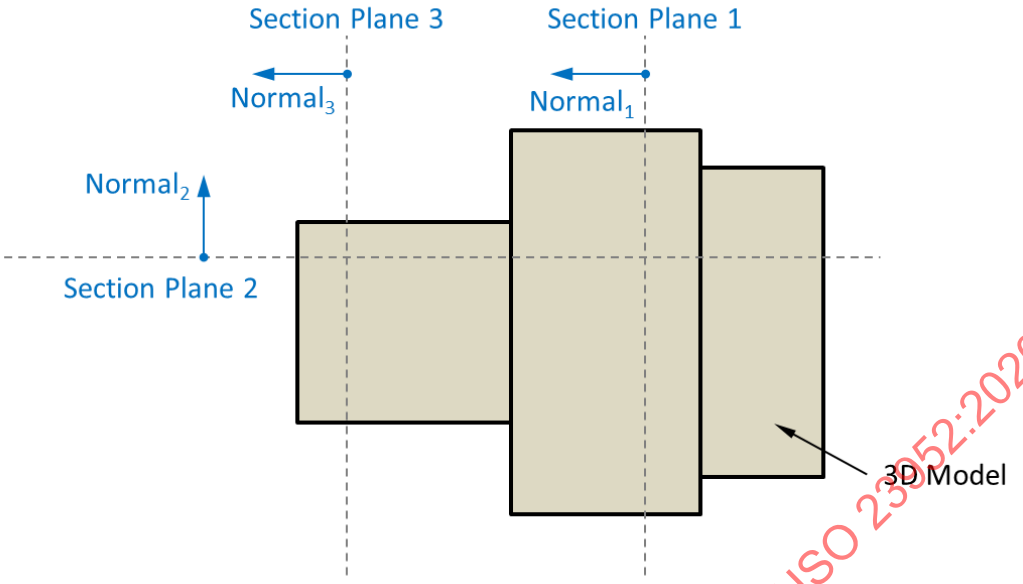


Figure 203 – Positions of the section planes

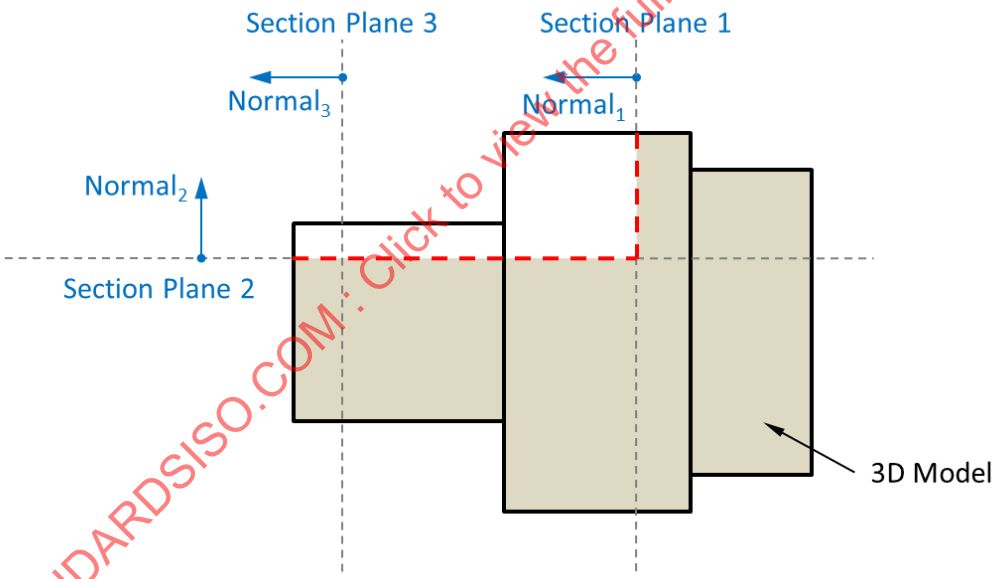


Figure 204 – The result of Section Plane 1 AND Section Plane 2

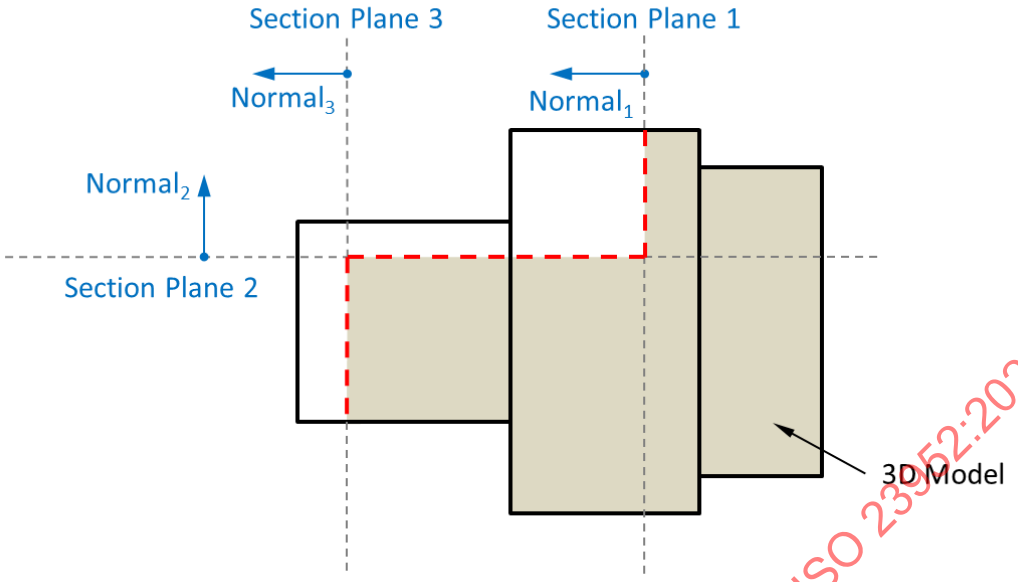


Figure 205 – The result of (Section Plane 1 AND Section Plane 2) OR Section Plane 3

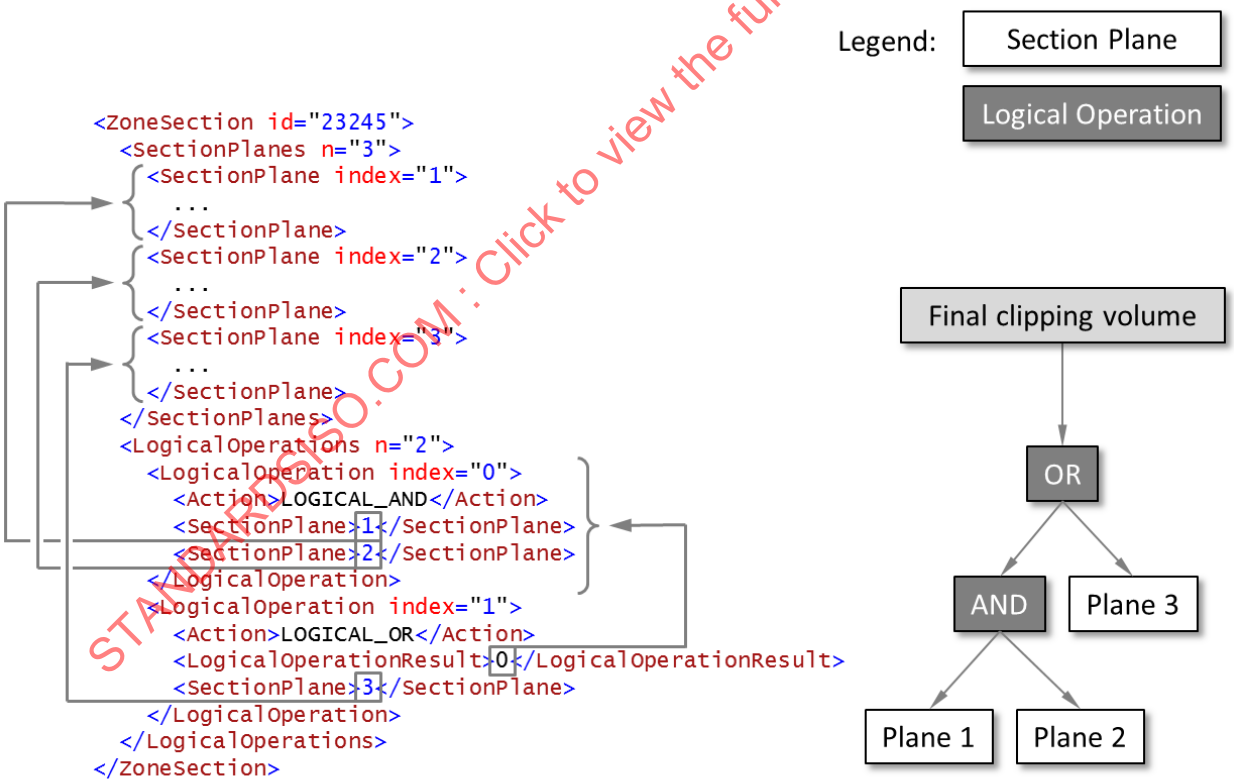


Figure 206 – Logical operations tree

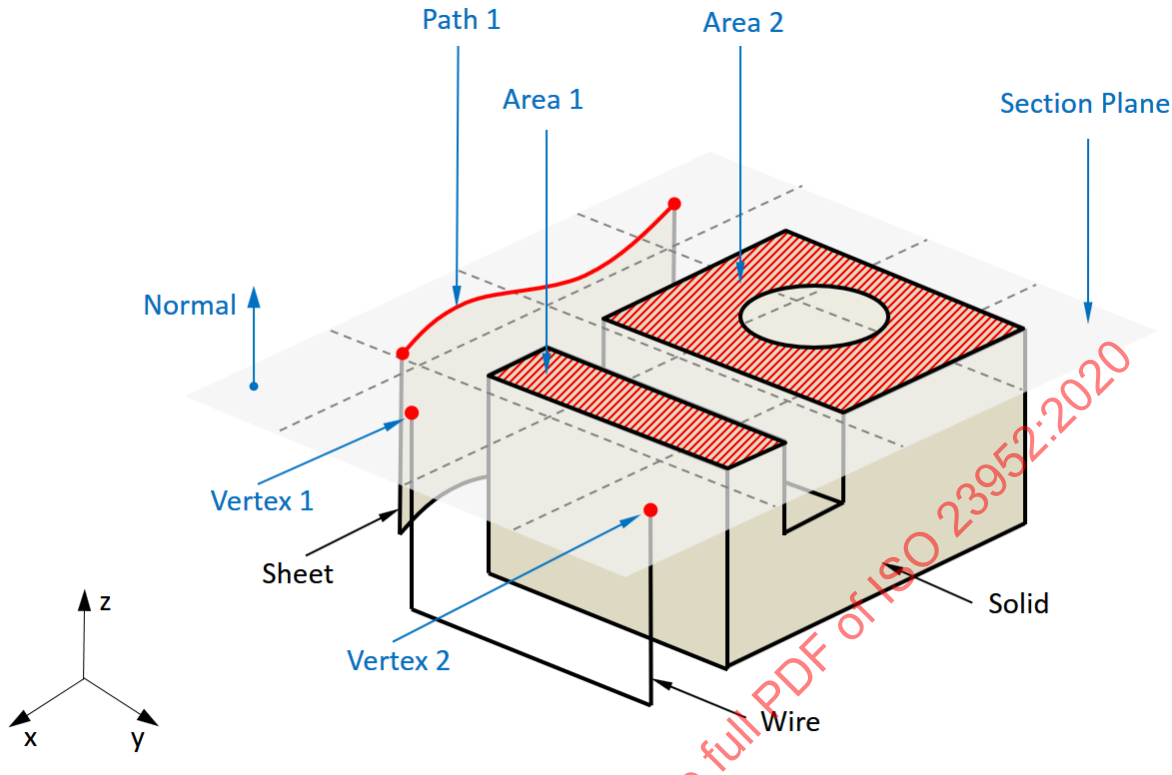


Figure 207 – Sections of wire, sheet and solid bodies

Fields:

Field Name	Data Type	Description
@id	QIFIdType	Unique model entity identifier.
@label	xs:string	The zone section label.
@hatching	xs:boolean	Hatching On/Off.
SectionPlanes	ZoneSectionPlanesType	An array of the section planes.
SectionPlanes/SectionPlane	ZoneSectionPlaneType	The section plane.
SectionPlanes/SectionPlane/@index	xs:unsignedInt	An index identifying a zone section plane within the SectionPlanes set.
SectionPlanes/SectionPlane/Plane	PlaneXType	The plane.
SectionPlanes/SectionPlane/SectionGroups	SectionGroupsType	The groups of cross-sections.
SectionPlanes/SectionPlane/SectionGroups/SectionGroup	SectionGroupType	The cross-section group.
SectionPlanes/SectionPlane/SectionGroups/SectionGroup/BodyId	QIFReferenceFullType	The body identifier.
SectionPlanes/SectionPlane/SectionGroups/SectionGroup/Areas	SectionAreasType	An array of the section areas, defined by loops and hatching style.
SectionPlanes/SectionPlane/SectionGroups/SectionGroup/Areas/Area	SectionAreaType	The section area.

SectionPlanes/SectionPlane/SectionGroups/SectionGroup/Areas/Area/HatchStyleId	QIFReferenceType	The identifier of the hatching style for the section area.
SectionPlanes/SectionPlane/SectionGroups/SectionGroup/Areas/Area/Loops	SectionLoopsType	A set of section closed loops - boundary of the area.
SectionPlanes/SectionPlane/SectionGroups/SectionGroup/Areas/Area/Loops/Loop	SectionPathType	The section closed loop.
SectionPlanes/SectionPlane/SectionGroups/SectionGroup/Areas/Area/Loops/Loop/Edges	SectionEdgesType	An array of oriented edges.
SectionPlanes/SectionPlane/SectionGroups/SectionGroup/Areas/Area/Loops/Loop/Edges/Edge	EdgeOrientedType	The oriented edge.
SectionPlanes/SectionPlane/SectionGroups/SectionGroup/Paths	SectionPathsType	The section paths.
SectionPlanes/SectionPlane/SectionGroups/SectionGroup/Paths/Path	SectionPathType	The section path.
SectionPlanes/SectionPlane/SectionGroups/SectionGroup/Paths/Path/Edges	SectionEdgesType	An array of oriented edges.
SectionPlanes/SectionPlane/SectionGroups/SectionGroup/Paths/Path/Edges/Edge	EdgeOrientedType	The oriented edge.
SectionPlanes/SectionPlane/SectionGroups/SectionGroup/Vertices	ArrayReferenceType	The section vertices.
LogicalOperations	LogicalOperationsType	A logical operations tree.
LogicalOperations/LogicalOperation	LogicalOperationType	The logical operation.
LogicalOperations/LogicalOperation/@index	xs:unsignedInt	An index identifying a logical operation within the LogicalOperations set.
LogicalOperations/LogicalOperation/Action	LogicalOperationEnumType	The logical operation type: LOGICAL_AND or LOGICAL_OR.
LogicalOperations/LogicalOperation/SectionPlane	xs:unsignedInt	The index of section plane.
LogicalOperations/LogicalOperation/LogicalOperationResult	xs:unsignedInt	The index of logical operation.

Example:

```
<ZoneSection id="23244">
  <SectionPlanes n="1">
    <SectionPlane index="0">
      <Plane>
        <Point>0 0 0</Point>
        <Normal>0 0 1</Normal>
        <Direction>1 0 0</Direction>
      </Plane>
      <SectionGroups n="2">
        <SectionGroup>
          <BodyId>128</BodyId>
```

```
          <Areas n="1">
            <Area>
              <HatchStyleId>1</HatchStyleId>
              <Loops n="2">
                <Loop>
                  <Edges n="4">
                    <Edge turned="1">
                      <Id>3467</Id>
                    </Edge>
                    <Edge turned="1">
                      <Id>3468</Id>
                    </Edge>
                    <Edge>
                      <Id>2783</Id>
                    </Edge>
                    <Edge>
                      <Id>2344</Id>
                    </Edge>
                  </Edges>
                </Loop>
                <Loop>
                  <Edges n="1">
                    <Edge>
                      <Id>2923</Id>
                    </Edge>
                  </Edges>
                </Loop>
              </Loops>
            </Area>
          </Areas>
        </SectionGroup>
        <SectionGroup>
          <BodyId>129</BodyId>
          <Paths n="1">
            <Path>
              <Edges n="3">
                <Edge>
                  <Id>3460</Id>
                </Edge>
                <Edge turned="1">
                  <Id>3568</Id>
                </Edge>
                <Edge>
                  <Id>1283</Id>
                </Edge>
              </Edges>
            </Path>
          </Paths>
        </SectionGroup>
      </SectionGroups>
    </SectionPlane>
  </SectionPlanes>
</ZoneSection>
```

7.5.7.6.5 *Hatching style*

The HatchStyleType defines a hatching style used for visualization of cross-section areas, as seen in Figure 208 and Figure 209. All hatching styles are collected in the HatchStyleSet that is an *element* of the ViewSet.

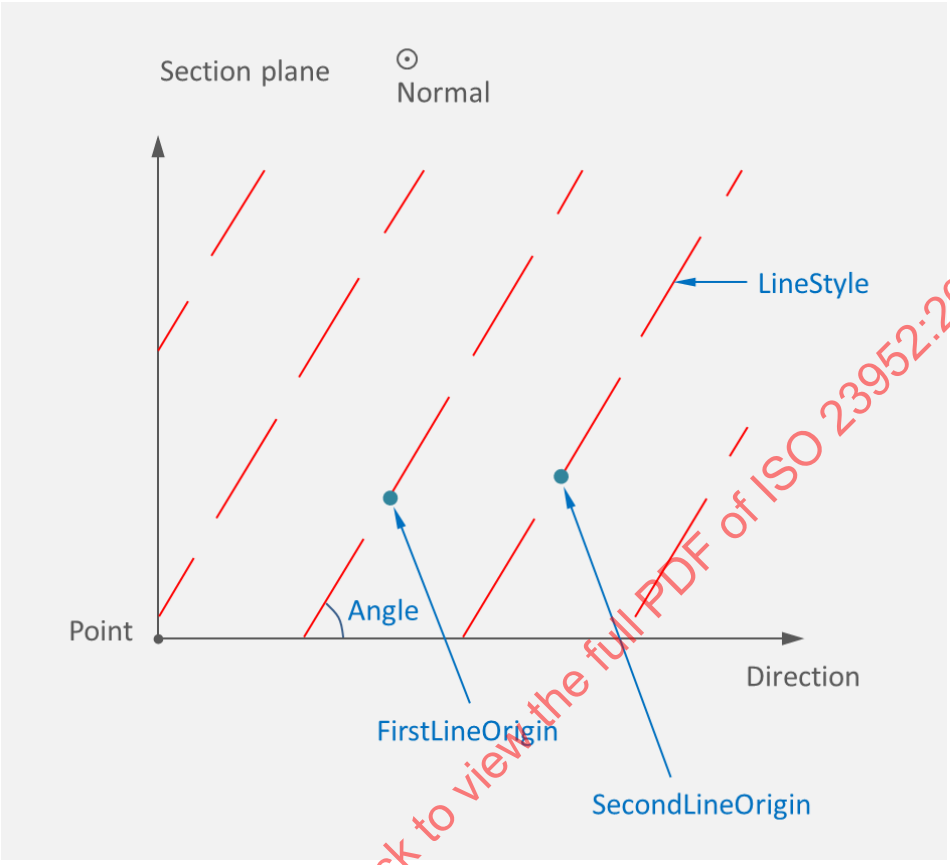


Figure 208 – Hatching Pattern

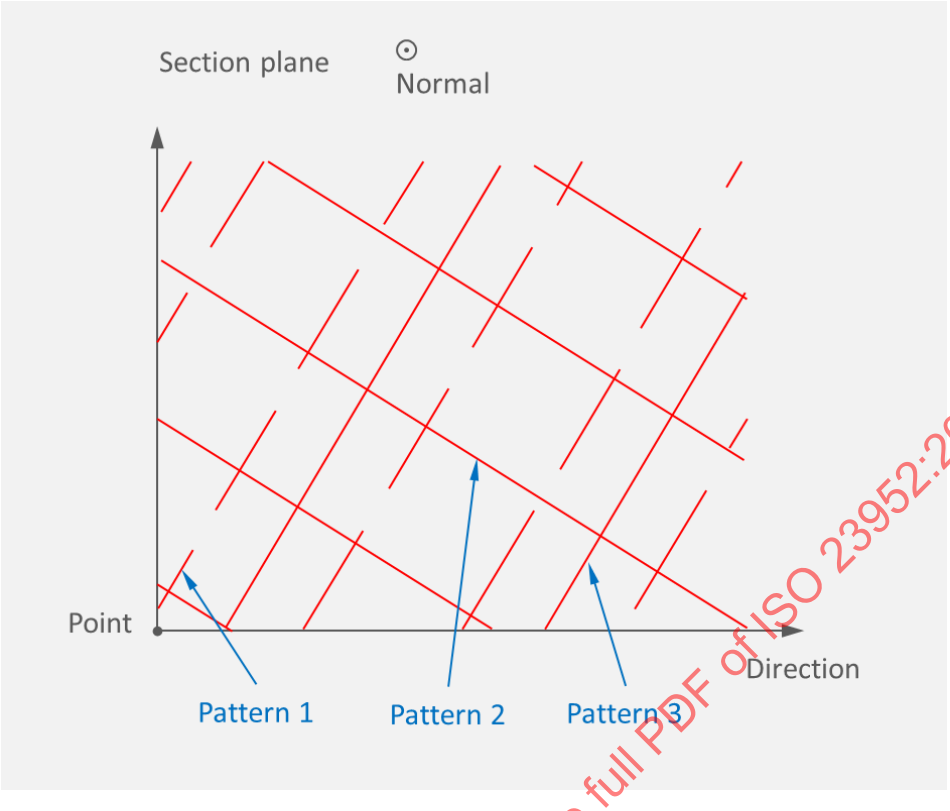


Figure 209 – Combination of patterns

Fields:

Field Name	Data Type	Description
@id	QIFIdType	The unique model entity identifier.
@label	xs:string	The label of the hatching style.
Form	HatchStyleFormEnumType	The form of the hatching.
Color	ColorType	The color of the hatching.
Patterns	HatchPatternsType	The array of hatching patterns.
Patterns/Pattern/LineStyle	HatchPatternLineStyleEnumType	The line style of the hatching.
Patterns/Pattern/LineStyle/@thickness	xs:positiveInteger	The line thickness in pixels.
Patterns/Pattern/FirstLineOrigin	Point2dSimpleType	The origin of the first hatching line.
Patterns/Pattern/SecondLineOrigin	Point2dSimpleType	The origin of the second hatching line.
Pattern/Angle	AngularValueType	The angle of the hatching lines.
Pattern/Color	ColorType	The color of the hatching lines.

Example:

```
<HatchStyleSet n="1">
  <HatchStyle id="1" label="hatch style 1">
```



```
    <Form>PATTERN</Form>
    <Color>0 0 0</Color>
    <Patterns n="3">
      <Pattern>
        <LineStyle>SOLID</LineStyle>
        <FirstLineOrigin>0 0</FirstLineOrigin>
        <SecondLineOrigin>4 0</SecondLineOrigin>
        <Angle>45</Angle>
      </Pattern>
      <Pattern>
        <LineStyle>DASH</LineStyle>
        <FirstLineOrigin>0 0</FirstLineOrigin>
        <SecondLineOrigin>4 0</SecondLineOrigin>
        <Angle>135</Angle>
      </Pattern>
      <Pattern>
        <LineStyle>DOT</LineStyle>
        <FirstLineOrigin>2 0</FirstLineOrigin>
        <SecondLineOrigin>6 0</SecondLineOrigin>
        <Angle>45</Angle>
      </Pattern>
    </Patterns>
  </HatchStyle>
</HatchStyleSet>
```

7.5.7.6.6 *Camera*

Camera describes a camera – projection of the 3D model space to 2D window, as seen in Figure 210 and Figure 211.

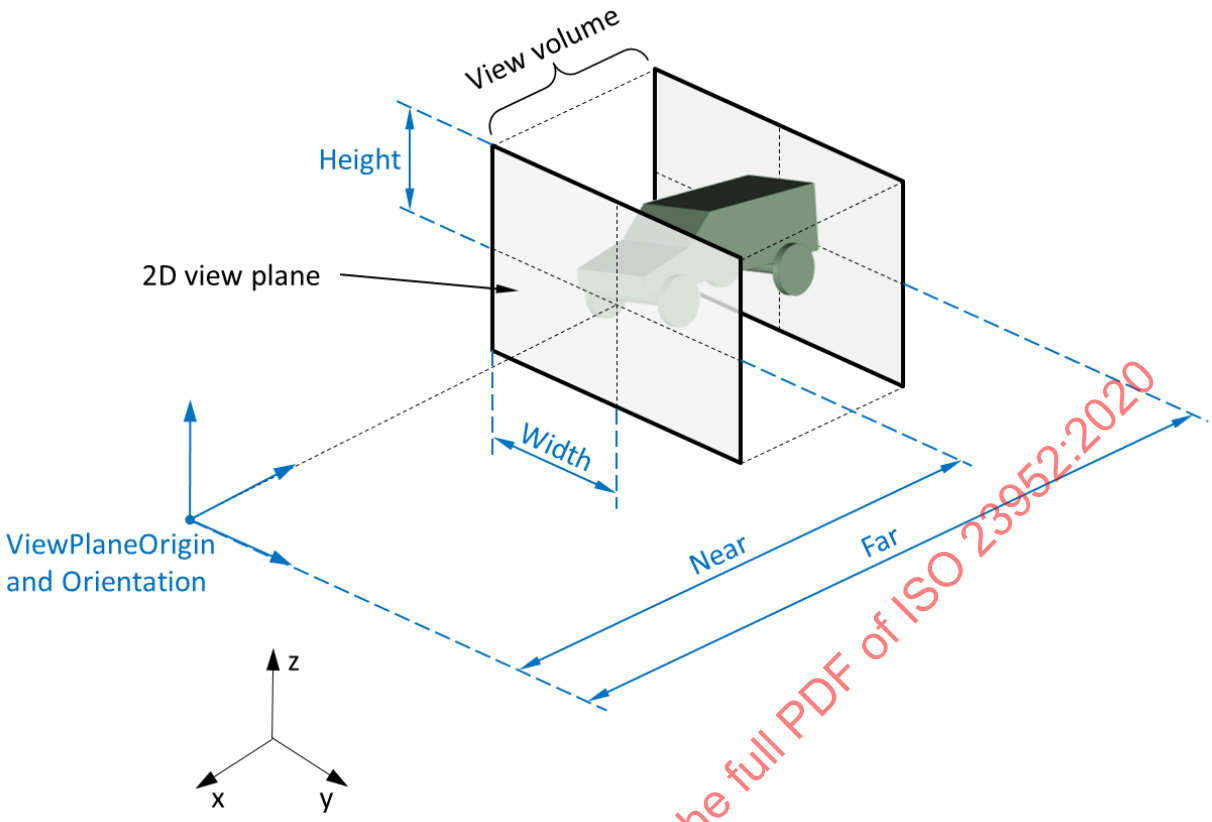


Figure 210 – Orthographic Camera

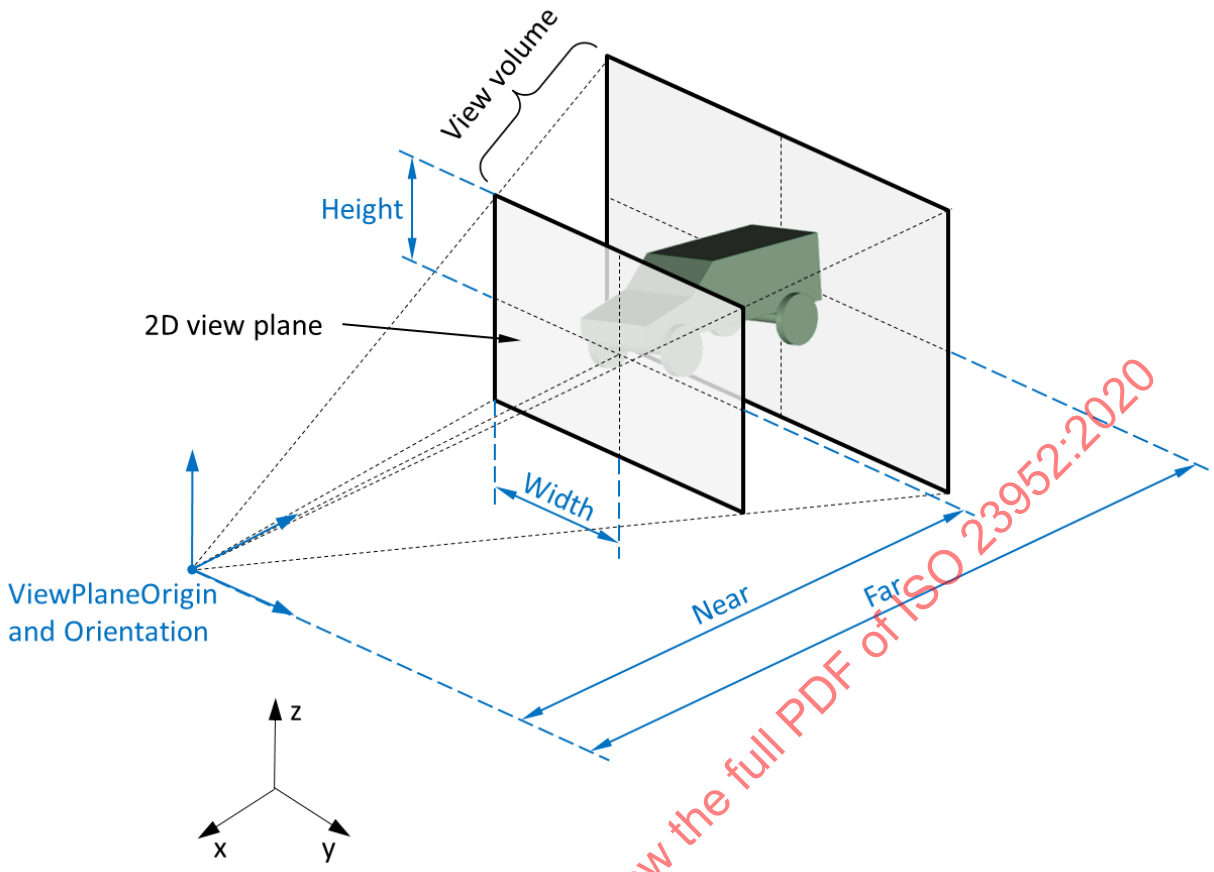


Figure 211 – Perspective Camera

$$Width = Height * Ratio$$

Fields:

Field Name	Data Type	Description
@form	CameraFormEnumType	This attribute specifies the camera type: 'ORTHOGRAPHIC' or 'PERSPECTIVE'.
ViewPlaneOrigin	PointSimpleType	The view plane origin.
Orientation	QuaternionType	The rotation of the view plane around the view plane origin.
Ratio	xs:double	The aspect ratio of the view plane (normally it corresponds to the viewport).
Near	LinearValueType	The distance from the view plane to the near clipping plane.
Far	LinearValueType	The distance from the view plane to the far clipping plane.
Height	xs:double	Half of the top to bottom extent of the 2D view plane.

Example:

```
<Camera id="348">
```

```
  <ViewPlaneOrigin>0.467 0.587 0.020</ViewPlaneOrigin>
  <Orientation>
    <Value>0.285 0.267 -0.185 -0.901</Value>
  </Orientation>
  <Ratio>1.465</Ratio>
  <Near>-2.449</Near>
  <Far>2.449</Far>
  <Height>1.079</Height>
</Camera>
```

7.5.8 Validation properties

7.5.8.1 Edge validation properties

Edge validation properties include total length, centroid, bounding box and points with tangent vectors.

Edge/Validation fields:

Field Name	Data Type	Description
Length	xs:double	The edge length.
Centroid	PointSimpleType	The edge centroid.
Box	BoundingBoxAxisAlignedType	The edge bounding box.
Points/Points or Points/PointsBinary	ArrayPointType or ArrayBinaryType	The set of points that lie on the edge.
Points/Directions or Points/DirectionsBinary	ArrayUnitVectorType or ArrayBinaryType	The set of tangent vectors defined at the points. The number of array *elements* corresponds to the number of points.

Example:

```
<Edge id="403">
  <Validation>
    <Length>100</Length>
    <Centroid>50 0 0</Centroid>
    <Box>
      <PointMin>0 0 0</PointMin>
      <PointMax>100 0 0</PointMax>
    </Box>
    <Points>
      <Points n="3">
        0 0 0
        50 0 0
        100 0 0
      </Points>
      <Directions n="3">
        1 0 0
        1 0 0
        1 0 0
      </Directions>
```



```
    </Points>
  </Validation>
  <Curve>
    <Id>390</Id>
  </Curve>
  <VertexBeg>
    <Id>396</Id>
  </VertexBeg>
  <VertexEnd>
    <Id>400</Id>
  </VertexEnd>
</Edge>
```

7.5.8.2 Face validation properties

Face validation properties include total area, centroid, bounding box and points with normal vectors.

Face/Validation, FaceMesh/Validation fields:

Field Name	Data Type	Description
Area	xs:double	The face area.
Centroid	PointSimpleType	The face centroid.
Box	BoundingBoxAxisAlignedType	The face bounding box.
Points/Points or Points/PointsBinary	ArrayPointType or ArrayBinaryType	The set of points that lie on the face.
Points/Directions or Points/DirectionsBinary	ArrayUnitVectorType or ArrayBinaryType	The set of normal vectors defined at the points. The number of array *elements* corresponds to the number of points.

Example:

```
<Face id="174">
  <Validation>
    <Area>10000</Area>
    <Centroid>50 50 0</Centroid>
    <Box>
      <PointMin>0 0 0</PointMin>
      <PointMax>100 100 0</PointMax>
    </Box>
    <Points>
      <Points n="5">
        0 0 0
        100 0 0
        100 100 0
        0 100 0
        50 50 0
      </Points>
      <Directions n="5">
        0 0 1
        0 0 1
        0 0 1
```

```
          0 0 1
          0 0 1
        </Directions>
      </Points>
    </Validation>
    <Surface>
      <Id>102</Id>
    </Surface>
    <LoopIds n="1">
      <Id>113</Id>
    </LoopIds>
</Face>
```

7.5.8.3 Body validation properties

Body validation properties include total area, centroid, volume, bounding box, points with normal vectors that lie on faces and points with tangent vectors that lie on edges.

Body/Validation fields:

Field Name	Data Type	Description
Area	xs:double	The body area.
Centroid	PointSimpleType	The body centroid.
Volume	xs:double	The body volume.
Box	BoundingBoxAxisAlignedType	The body bounding box.
FacePoints/Points or FacePoints/PointsBinary	ArrayPointType or ArrayBinaryType	The set of points that lie on the faces.
FacePoints/Directions or FacePoints/DirectionsBinary	ArrayUnitVectorType or ArrayBinaryType	The set of normal vectors defined at the face points. The number of array *elements* corresponds to the number of face points.
EdgePoints/Points or EdgePoints/PointsBinary	ArrayPointType or ArrayBinaryType	The set of points that lie on the edges.
EdgePoints/Directions or EdgePoints/DirectionsBinary	ArrayUnitVectorType or ArrayBinaryType	The set of tangent vectors defined at the edge points. The number of array *elements* corresponds to the number of edge points.

Example:

```
<Body id="127">
  <Validation>
    <Area>60000.0</Area>
    <Centroid>50 50 50</Centroid>
    <Volume>1000000.0</Volume>
    <Box>
      <PointMin>0 0 0</PointMin>
      <PointMax>100 100 100</PointMax>
    </Box>
    <FacePoints>
```



```
      <Points n="2">
        50 50 0
        50 50 100
      </Points>
      <Directions n="2">
        0 0 -1
        0 0 1
      </Directions>
    </FacePoints>
    <EdgePoints>
      <Points n="2">
        50 0 0
        50 100 0
      </Points>
      <Directions n="2">
        1 0 0
        1 0 0
      </Directions>
    </EdgePoints>
  </Validation>
  <FaceIds n="1">
    <Id>175</Id>
  </FaceIds>
  <LoopIds n="1">
    <Id>126</Id>
  </LoopIds>
  <EdgeIds n="4">
    <Id>487</Id>
    <Id>495</Id>
    <Id>503</Id>
    <Id>511</Id>
  </EdgeIds>
  <VertexIds n="8">
    <Id>484</Id>
    <Id>486</Id>
    <Id>492</Id>
    <Id>494</Id>
    <Id>500</Id>
    <Id>502</Id>
    <Id>508</Id>
    <Id>510</Id>
  </VertexIds>
</Body>
```


7.5.8.4 Part/Assembly validation properties

Part/Assembly validation properties include total area, centroid, volume, bounding box, points with normal vectors that lie on faces, points with tangent vectors that lie on edges, and centroid and bounding box defined for each part/assembly instance.

Validation fields:

Field Name	Data Type	Description
Area	xs:double	The part/assembly area.

Centroid	PointSimpleType	The part/assembly centroid.
Volume	xs:double	The part/assembly volume.
Box	BoundingBoxAxisAlignedType	The part/assembly bounding box.
FacePoints/Points or FacePoints/PointsBinary	ArrayPointType or ArrayBinaryType	The set of points that lie on the faces.
FacePoints/Directions or FacePoints/DirectionsBinary	ArrayUnitVectorType or ArrayBinaryType	The set of normal vectors defined at the face points. The number of array *elements* corresponds to the number of face points.
EdgePoints/Points or EdgePoints/PointsBinary	ArrayPointType or ArrayBinaryType	The set of points that lie on the edges.
EdgePoints/Directions or EdgePoints/DirectionsBinary	ArrayUnitVectorType or ArrayBinaryType	The set of tangent vectors defined at the edge points. The number of array *elements* corresponds to the number of edge points.
Instances	ValidationPartAssemblyInstancesType	The array of instance validation properties.
Instances/Instance/AsmPathId	QIFReferenceType	The identifier of the assembly path of this part/assembly.
Instances/Instance/Centroid	PointSimpleType	The instance centroid.
Instances/Instance/Box	BoundingBoxAxisAlignedType	The instance bounding box.

Example:

```
<Part id="1930" label="crank-arm">
  <Validation>
    <Area>60000.0</Area>
    <Centroid>50 50 50</Centroid>
    <Volume>1000000.0</Volume>
    <Box>
      <PointMin>0 0 0</PointMin>
      <PointMax>100 100 100</PointMax>
    </Box>
    <FacePoints>
      <Points n="2">
        50 50 0
        50 50 100
      </Points>
      <Directions n="2">
        0 0 -1
        0 0 1
      </Directions>
    </FacePoints>
    <EdgePoints>
      <Points n="2">
```



```
        50 0 0
        50 100 0
      </Points>
      <Directions n="2">
        1 0 0
        1 0 0
      </Directions>
    </EdgePoints>
    <Instances n="2">
      <Instance>
        <AsmPathId>1123</AsmPathId>
        <Centroid>150 50 50</Centroid>
        <Box>
          <PointMin>100 0 0</PointMin>
          <PointMax>200 100 100</PointMax>
        </Box>
      </Instance>
      <Instance>
        <AsmPathId>1124</AsmPathId>
        <Centroid>50 50 150</Centroid>
        <Box>
          <PointMin>0 0 100</PointMin>
          <PointMax>100 100 200</PointMax>
        </Box>
      </Instance>
    </Instances>
  </Validation>
  <BodyIds n="4">
    <Id>1929</Id>
    <Id>1921</Id>
    <Id>1913</Id>
    <Id>1899</Id>
  </BodyIds>
</Part>
```

7.5.9 High level description of the product data

The QIF MBD model includes the information items from the Product.xsd schema file and many of the schema files in the QIF Library. The QIF Library files are incorporated into the schema by a chain of "*include*" directives starting in the Product.xsd schema file. A diagram of the highest-level *element*s of the QIF MBD model are seen in Figure 212.

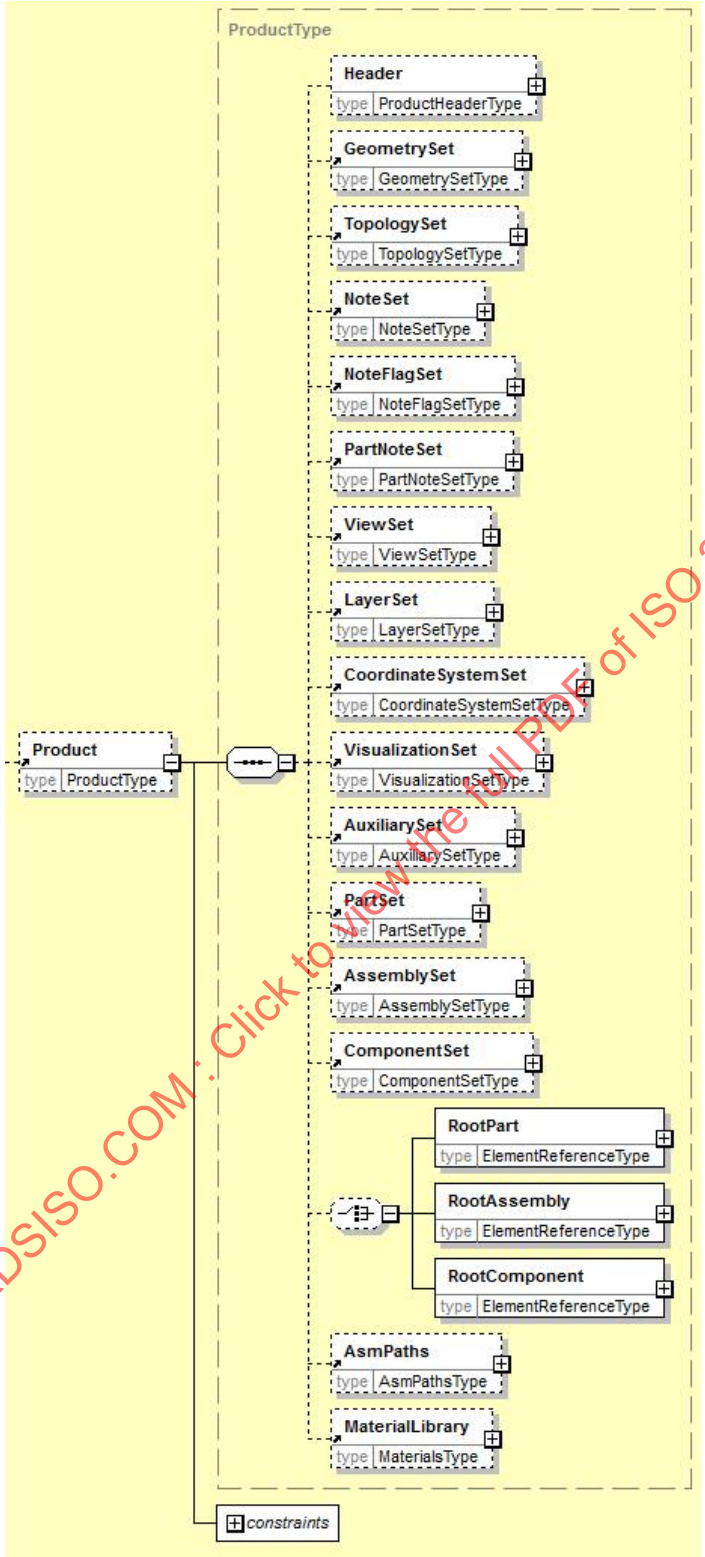


Figure 212 – High level view of QIF MBD highest level *elements*

8 QIF Plans information model

8.1 Foreword

The Quality Information Framework (QIF) information model was developed by domain experts from the manufacturing quality (that is metrology) community representing a wide variety of industries and quality measurement needs. Specifically for the QIF Plans work, past and current contributors include:

Main:

- Honeywell Federal Manufacturing & Technology
- Lockheed Martin Missiles and Fire Systems
- Mitutoyo America
- National Institute of Standards and Technology
- Origin International Inc.
- QIF Solutions
- Renishaw plc

Support:

- Capvidia
- Deere & Co.
- DISCUS Software
- Hexagon Metrology
- Horst Engineering
- IPI Solutions
- ITI transenData
- Metrosage LLC
- PAS Technology
- Rolls-Royce
- Siemens
- University of North Carolina at Charlotte
- Validation Technologies

This clause was written by the QIF Plans Working Group and given final approval for ANSI review by the DMSC's Quality Measurement Standards (QMS) Committee.

This version of the QIF Plans information model contains significant additions to the previous standard, which was published as ANSI/QIF Part 4 – 2015. These include:

- Numerical-valued variables may be declared, set, and evaluated. The value of a variable may be used in arithmetic expressions.
- The "If" action group has been added. It includes zero to many ElseIfs and an ElseDo. It behaves the way "If" behaves in some widely-used programming languages.
- A "While" action group has been added. It behaves the way "while" behaves in some widely-used programming languages.
- Expressions were defined for use in rules in version 2.1, but they were not used in plans. In QIF version 3.0, expressions are used in plans in the tests made in "If" and "While".
- The ability to reference a measurement taken while executing a plan has been added. This was done by defining the VirtualMeasurementType with an id and putting an optional VirtualMeasurement *element* in the CharacteristicItemBaseType and the FeatureItemBaseType. Measured values can be obtained during plan execution by referencing the id in the VirtualMeasurement *element*.

- String (actually xs:token) expressions were added with very limited use. A test can be made whether two strings are identical.
- A “Halt” action was added so that execution of a plan can be stopped before completion.
- The LaserRadar measure feature method was added.

All additions were designed to be translatable into the DMIS programming language. Plans are not intended to be directly executable. As used above, “plan execution” really means execution of a program created from a plan.

8.2 Introduction

In brief, QIF Plans is an information model representing the unique QIF Application of measurement plans. QIF Plans defines the information necessary to support multiple levels of measurement planning, including:

- a simple Bill of Characteristics (BoC),
- measurement scope,
- inspection planning,
- CMM measurement planning,
- quality routing plans,
- first article inspection,
- manufacturing verification,
- on-machine gaging,
- final product acceptance.

Quality measurement plans are required by all quality departments to efficiently and effectively validate conformance of manufactured products according to design specifications. QIF provides the only standard format for defining measurement plans at any level above measurement programs. Without QIF, small, medium and enterprise manufacturers suffer from the need to transpose measurement information from one system to another. Older methodologies waste valuable time, allow for human mistakes, and cost money. QIF provides a common, standardized information definition for measurement planning to carry quality requirements information from one system to another.

The objective of the QIF Plans information model is to communicate all the necessary feature definition, product characteristics and related quality information required to measure and verify product requirements. Although the scope of QIF Plans places an emphasis on feature-based dimensional metrology and the dimensional measurement programs to be executed on dimensional measurement equipment, much attention was devoted toward addressing significant *element*s that also support attribute and other types of validations.

A QIF Plans file is an XML instance file conforming to the QIF Document schema and having a **Plan** *element* in the **QIFDocument** root *element*. The **Plan** *element* itself does not contain all information necessary and sufficient to perform inspection or generate a CMM program for performing inspection. Information regarding features, characteristics, product shape and structure (i.e., CAD data), measurement rules, and measurement resources is associated with the plan by being in other *element*s of the same **QIFDocument** or by being in one or more external QIF files. The plan references that information using QIF ids and QPIds. Non-QIF data in external files or on drawings may also be referenced.

Directly or by reference, a QIF Plan will consist of many categories of information. It will contain a list of features and an associated bill of characteristics (BoC) such as tolerances associated with

features. The plan can reference additional product information such as datums, datum reference frames, and measurement resources, as well as traceability information. At a minimum, a QIF Plan will define one or more actions that specify “what” to measure or validate. These actions can be further organized in a hierarchical group of actions. Also within QIF Plans, each action may reference one or many action methods on “how” to perform a measure or validation. In addition, an action may specify measurement resources (measurement devices, fixtures, and sensors).

8.3 Scope

8.3.1 Contents of this clause

This clause defines a QIF application information model called QIF Plans. The information model is described in a data dictionary and consists of definitions for data types, *element*s, the logical relationships between them, and the semantics of the quality information for the application of Quality Planning.

The XML schema definition language also allows for the definition of rules and checks that enforce constraints on the computer-to-computer writing and reading of instances of QIF Plans. The file that defines the top levels of the QIF Plans information model and contains some of the rules is QIFPlan.xsd. The bulk of the model is defined in the files of the QIF Library, many of which are used by the QIFPlan.xsd file. All QIF XML schema files are normative to the QIF standard and can be downloaded at www.qifstandards.org.

8.3.2 Workflow of QIF Plans data for manufacturing quality

Figure 2 shows a Model Based Product Verification Workflow activity diagram flowing from a product definition as the primary input to the reporting and analysis of measurement results. The work flow activities for a Quality Metrology Enterprise are contained in the blue boxes and consist of four major Quality Metrology Enterprise activities:

- Determine Measurement Requirements
- Define Measurement Process
- Execute Measurement Process
- Analyze & Report Quality Data

Between any two connected work activities there exists an interface in which digital information must be exchanged.

Assuming that the activity of “*Define Product*” has already been performed, the planning work activity of “*Determine Measurement Requirements*” receives product definition with Product and Manufacturing Information (PMI) as input which includes product characteristics and their criticalities. Then based upon known quality requirements and/or manufacturing process knowledge, measurement requirements are determined as a set of measurement criteria also known as a bill of characteristics (BOC). A product characteristic is typically a tolerance or specification applied to a feature or product that needs verification. This BOC may constitute a high level quality plan – a list of “what” needs to be inspected or verified. Next, given metrology resources and metrology knowledge, the “*Define Measurement Process*” activity augments the set of measurement requirements by defining an inspection plan on “how” to inspect or verify the items in the bill of characteristics. The “hows” may reference one or many measurement resources. The generation of the "hows" may be guided by a measurement rule set. This inspection plan will then drive the

activity of “*Execute Measurement Process*” via various measurement resources, which produces measurement results.

The QIF Plan conveys information output of the “*Determine Measurement Requirements*” activity by describing a measurement scope that lists all of the product characteristics and their features and the actions required to measure/validate them. Furthermore the QIF Plans conveys the output of “*Define Measurement Process*” by describing the methods on how to measure/validate the actions contained in the measurement scope.

8.3.3 QIF Plans information model

The QIF Plans information model focuses on actions, the "what" of inspection, and includes action methods, the "how" to measure. The "what" defines the product characteristics needing to be verified/evaluated and the corresponding features that need to be measured. The “how” augments the “what” with additional information such as the verification or measurement methods. "What" can also designate generic or specific measurement resources according to quality engineering business practices or by specified measurement rules. Information about inspection equipment, and detailed methods of inspection, (the "how"), can be generated at the time of initial QIF Plan generation or can be gradually augmented during further maturation of the QIF Plan. As a result, a QIF Plan has the advantage of specifying either only the actions of inspection, or it can be further augmented with details on the action methods to be used. This separation of "what" from "how" permits flexibility in configuring manufacturing computer aided quality systems. The work flow to the “what” stage is illustrated in Figure 213. The work flow through the “how” stage is shown in Figure 214.

8.3.4 QIF Plans scope

The QIF Plans information model was designed with the following scope and out-of-scope requirements.

In scope:

- Feature-Based
- Dimensional Metrology
- Non-Dimensional Metrology
- Product (that is, Part and/or Assembly)
- Product Characteristics
    - Dimensional Tolerances
    - Geometric Tolerances
    - User Defined Characteristic (attribute or variable)
    - Surface Textures
    - Thread Specifications
    - Flagged Notes
    - General Notes
- Characteristics Designator (that is, a human-readable product unique identifier (for example <PC007>) similar to the known practice of “ballooning”)
- Characteristics Criticality Class Levels (for example MINOR, MAJOR, CRITICAL)
- Characteristic QPId (that is, a persistent digital-readable universally unique identifier similar to the UUID)
- Datum Features, Datums, Datum Targets, and Datum Reference Frames
- Traceability Information
- Work Instructions (reference to a document, image, text, or video)
- Determine Measurement Scope (that is a list of actions of “what” to measure)

- Define Measurement Plan (a hierarchical list of actions of “what” to measure with associated methods on “how” to measure)
- Associations to Measurement Resources
- Consideration of Measurement Definitions that include accuracy statements and calibration statements
- Utilization of Measurement Rules
- Product Model-Based Definition and Model Entity Relationships
- Planning from a Model Based Product Definition Environment
- Planning from a Document/Drawing based Product Definition Environment

Out of scope:

- Reaction planning defining corrective action
- Sampling plans
- Low level execution plans or “DME Programming”

8.3.5 QIF Plans use cases

The QIF Plans 3.0 model was designed to support the following use cases.

8.3.5.1 Bill of Characteristics

A measurement requirement is often identified as a product characteristic. A product characteristic is typically a tolerance or specification applied to a feature or product that needs verification. A group of product characteristics for a product is called a Bill of Characteristics (BoC). A BoC is a complete listing of the product characteristic items required for verifying that a product meets requirements. A BoC can be considered as a simple measurement plan. From a QIF Characteristic Item, one can obtain the characteristic data, the associated feature Item, the criticality of the characteristic item, the human readable name of the characteristic item and the machine readable universally unique persistent identifier for the characteristic item, known as the QPId.

8.3.5.2 Measurement Scope

In the measurement scope use case, shown in Figure 213, the BoC defines “what” to measure. A BoC can include the following:

- Single Action: definition of a set of features and characteristics, and definition of one action stating Measure and Evaluate ALL defined Characteristics
- Specified Action Plan: definition of a set of features and characteristics, and definition of one action to measure and evaluate specified characteristics
- Set of Actions Plan: definition of a set of features and characteristics, plus definition of an Action Group that contains Actions to measure and/or evaluate a specified characteristic or measure a feature.

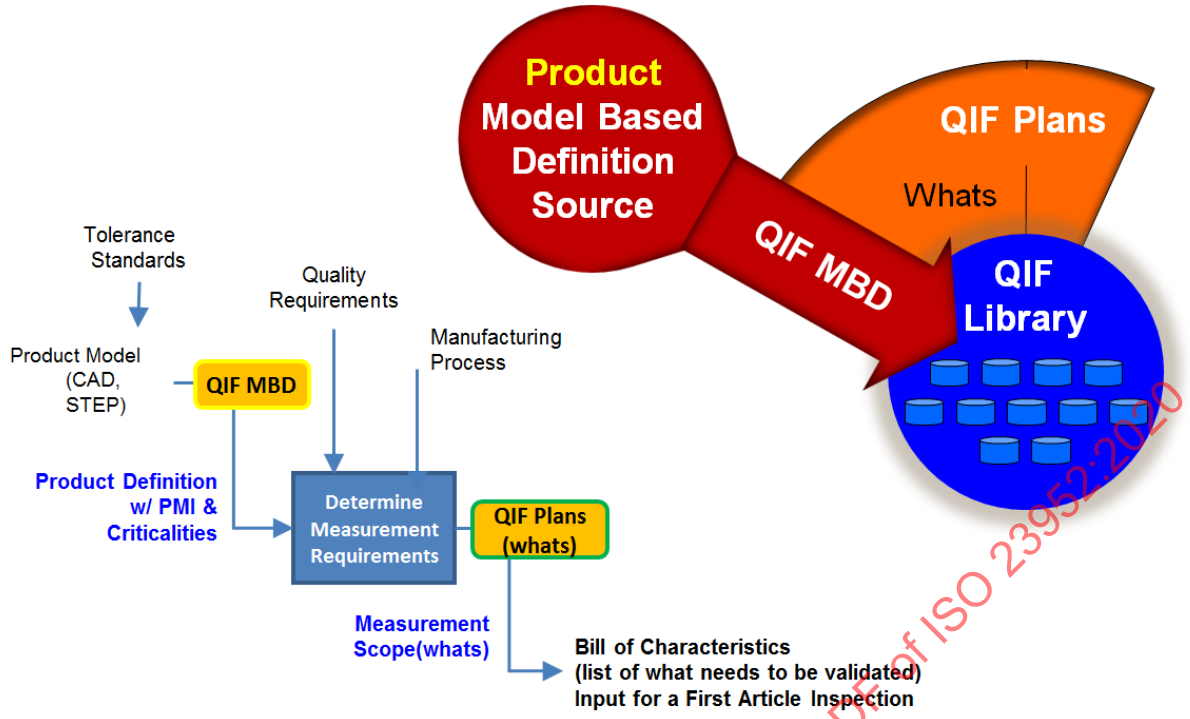


Figure 213 – Measurement Scope (e.g., Bill of Characteristics) with QIF Plans

8.3.5.3 Inspection Plan

The inspection plan use case, shown in Figure 214, defines “what” to measure with “how” to measure. An inspection plan can include the following:

- Unordered Plan of Actions with Methods
- Directed Plan of Actions with Methods which specifies a hierarchical order of actions

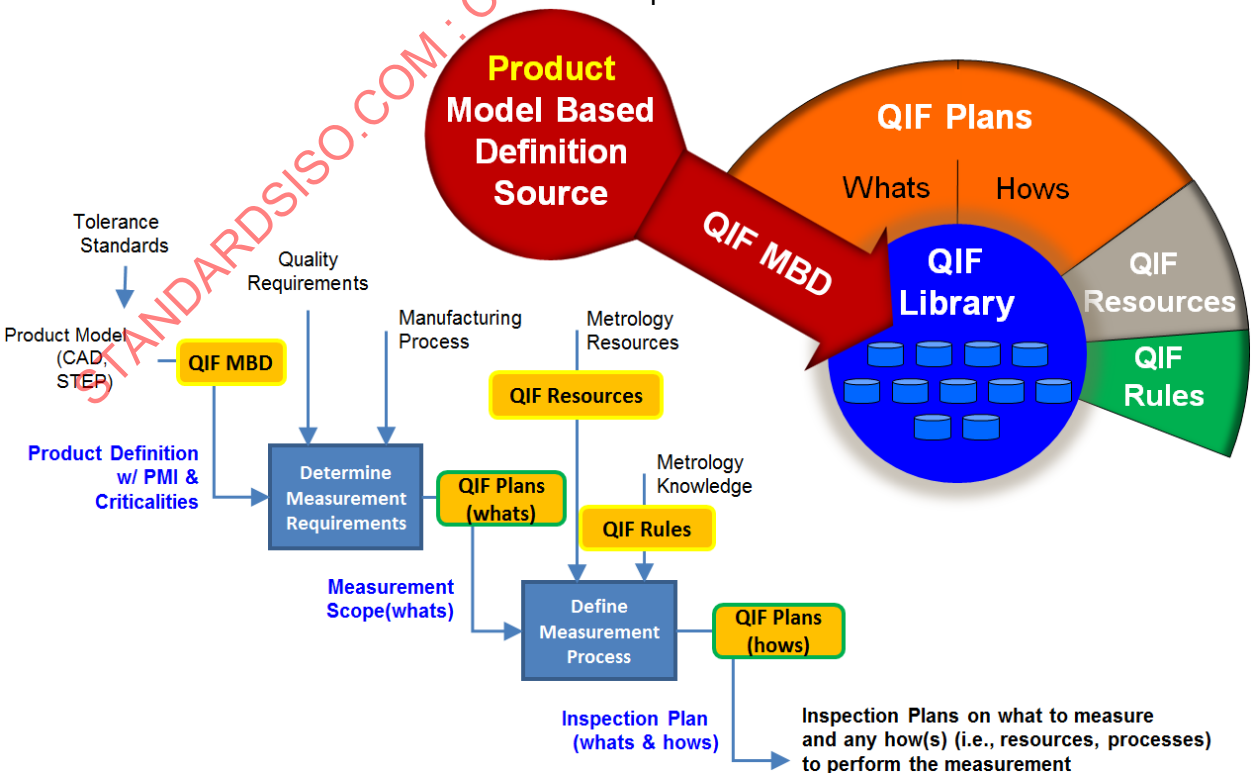


Figure 214 – Inspection Process Planning with QIF Plans

8.3.5.4 Routing Plan

The routing plan use case, includes the “what” to measure with “how” to measure, but also adds the “where” to measure such as the work cell for performing the measurement.

8.3.5.5 Application Use-Cases

A QIF Plan can be used for the following activities:

- Plan for First Article Inspection (FAI)
- Plan for Sample (Partial) Inspection
- Plan for 100% Inspection
- Plan within a Technical Data Package (MIL-STD-31000A)
- Plan using supplied Measurement Rules Definition
- Plan from a Model Based Product Definition
- Plan from a Document/Drawing-Based Product Definition
- Plan for complete Plant-wide Product Inspection
- Plan for Enterprise-wide Product Inspection
- Plan for QIF Plans items Persistence with QIF Results items

8.3.6 QIF Plans product definition support

The QIF information models support a digital engineering environment. The QIF is designed to support a workflow of quality data within a product definition environment that is either:

- Model-Centric (that is product shape only)
- Model-Based Definition (that is product shape with PMI)
- Drawing-based

8.4 Data types and *element*s of the QIF Plans information model

The QIF Plans information model employs the following high level data types and *element*s. Data *element*s are shown in Figure 216.

8.4.1 Plan

A QIF instance file that is a **Plan** has a unique persistent identifier. The file may make an association with a product definition (that is the model-centric, model-based definition, or drawing based), may have traceability information that describes the circumstances of the plan, has one **PlanElement**, and may reference one or many rules to be used or that were used.

8.4.2 PlanElement

A **PlanElement** can be an action or an action group.

8.4.3 Action

An action defines “what” needs to be measured/validated. An action may:

- include instructions defining what to do,
- specify measurement resources,
- have a preferred action method,
- have a list of alternative action methods,
- have preferred measurement resources,
- have a list of work instructions.

Descriptions of the specific types of action defined in QIF follow.

8.4.3.1 *MeasureEvaluateAllActionType*

The ***MeasureEvaluateAllActionType*** defines an action that means: measure whatever is necessary in order to find **CharacteristicMeasurement**s for all **CharacteristicItem**s, and find those measurements. For any **FeatureItem**s that are measured, also populate the corresponding **FeatureMeasurement**s with values.

8.4.3.2 *MeasureEvaluateSpecifiedActionType*

The ***MeasureEvaluateSpecifiedActionType*** defines an action that means: measure whatever is necessary in order to find **CharacteristicMeasurement**s for the **CharacteristicItem**s whose QIF ids are specified, and find those measurements. For any **FeatureItem**s that are measured, also populate the corresponding **FeatureMeasurement**s with values.

8.4.3.3 *EvaluateSpecifiedCharacteristicsActionType*

The ***EvaluateSpecifiedCharacteristicsActionType*** defines an action that means: find **CharacteristicMeasurement**s for the **CharacteristicItem**s whose QIF ids are specified. Do not measure anything while performing this action. All features that need to be measured in order to evaluate the **CharacteristicMeasurement**s must already be measured.

8.4.3.4 *MeasureSpecifiedFeaturesActionType*

The ***MeasureSpecifiedFeaturesActionType*** defines an action that means: measure the **FeatureItem**s whose QIF ids are specified, and populate the corresponding **FeatureMeasurement**s with values.

8.4.3.5 *MeasureSpecifiedMeasurandsActionType*

The ***MeasureSpecifiedMeasurandsActionType*** defines an action that means: measure the **Measurand**s whose QIF ids are given, and populate the corresponding **FeatureMeasurement**s with values.

8.4.3.6 *HaltActionType*

The ***HaltActionType*** defines an action that means stop executing the plan. A plan is not required to include an instance of ***HaltActionType***.

8.4.3.7 *VariableSetType*

The ***VariableSetType*** represents the setting of a variable.

8.4.4 Action Groups

Actions may be combined together into the following types of action group: ordered group, unordered group, one of group, partially ordered group, pick some group, “if” group, or “while” group.

8.4.4.1 OrderedActionGroup

The **OrderedActionGroup** defines an action group for which all of the steps it contains must be executed in order of increasing sequence number. The **SequenceNumber**s of the **Step**s in an instance of **OrderedActionGroup** must be present, must be assigned 1, 2, 3, ..., and must be executed in that order.

8.4.4.2 UnorderedActionGroup

The **UnorderedActionGroup** defines an action group for which all of the **Step**s it contains should be executed, however the **Step**s may be executed in any order and no particular order of execution is implied.

8.4.4.3 PartiallyOrderedActionGroup

The **PartiallyOrderedActionGroup** defines an action group for which all of the **Step**s it contains should be executed, but each **Step** may be executed only after all of the **Predecessors** of that **Step** have been executed. If more than one **Step** meets that condition, any order of executing those **Step**s will work, and no particular order is required by the plan. The **SequenceNumber**s of the **Step**s in the **StepsWithPredecessors** in an instance of **PartiallyOrderedActionGroup** must be assigned 1, 2, 3, ... but usually will not be executed in that order.

8.4.4.4 OneOfActionGroup

The **OneOfActionGroup** defines an action group for which exactly one of the **Step**s it contains must be executed. Any **Step** in the list will do. The **SequenceNumber**s of the **Step**s in an instance of **OneOfActionGroup** are not required to be distinct. The **SequenceNumber**s indicate a preference for which **Step** is executed, with 1 the most preferred, 2 the second most preferred, and so on. **Step**s with the same **SequenceNumber** are equally preferred.

8.4.4.5 IfActionGroup

The **IfActionGroup** is a conditional action group that represents an If-ElseIf-Else construct with zero to many Else-Ifs and an optional Else for the purpose of conditionally executing one or more specific actions. Details of the **IfActionGroup** are given in subclause 8.9.4.2.

8.4.4.6 WhileActionGroup

The **WhileActionGroup** is a conditional action group that represents a while loop that controls a flow of one or more actions to be executed repeatedly based on a given Boolean expression. Details of the **WhileActionGroup** are given in subclause 8.9.4.3.

8.4.5 Nesting of Action Groups

Inside any action group (starting with the **PlanRoot**, which is an action group), the plan *element*s to be executed may be either actions or action groups. In this way, a nest of action groups, possibly interspersed with actions, may be formed. For most practical purposes of dimensional metrology, however, a single root action group with no nested action groups will suffice as a plan.

8.4.6 Action Group Functions

The ***MeasureActionGroupFunctionType*** defines the function of an action group. The data *element* may contain one of the enumerated functions in the following subclauses or it may contain a string describing some other function.

8.4.6.1 RoutingPlan

A **RoutingPlan** is a group of actions that may route the product from one measurement work cell to another. It's enumeration value is ROUTING_PLAN.

8.4.6.2 OperationSequenceGroup

An **OperationSequenceGroup** is a group of actions that are performed at one work cell, typically referencing a measurement device resource. It's enumeration value is OPERATION_SEQUENCE.

8.4.6.3 SetupUsageGroup

A **SetupUsageGroup** is a group of actions that are performed at one part setup, typically referencing a fixture resource. It's enumeration value is SETUP_USAGE_GROUP.

8.4.6.4 SensorUsageGroup

A **SensorUsageGroup** is a group of actions that are performed at one sensor tool change, typically referencing a sensor resource. It's enumeration value is SENSOR_USAGE_GROUP.

8.4.6.5 CarriageUsageGroup

A **CarriageUsageGroup** is a group of actions that are performed at one carriage, typically referencing a carriage of a measurement device resource. It's enumeration value is CARRIAGE_USAGE_GROUP.

8.4.6.6 PartCoordinateSystemUsageGroup

A **PartCoordinateSystemUsageGroup** is a group of actions that are performed at one active part coordinate system which usually corresponds with a datum reference frame. It's enumeration value is PCS_USAGE_GROUP, where "PCS_" means "part coordinate system".

8.4.6.7 EvaluateCharacteristicActions

An **EvaluateCharacteristicActions** is a group of actions that are performed to evaluate a product characteristic. It's enumeration value is EVALUATE_CHAR_ACTIONS, where "CHAR_" means "characteristic".

8.4.6.8 EstablishDatumActions

An **EstablishDatumActions** is a group of actions that are performed to establish a datum with a datum reference frame. It's enumeration value is ESTABLISH_DATUM_ACTIONS.

8.4.7 Measurand

A measurand is the object, quantity, property or condition to be measured for a specific purpose. It can be used to evaluate a product characteristic item of a feature, can be used to establish a datum instance within a datum reference frame, and can be used to determine a particle/grain size within granular materials.

8.4.8 Action Method

An action method describes “how” an action is to be performed. It can be used by multiple actions, may have chosen measurement resources, may have work instructions, and may be a measure feature method derived type.

8.4.9 Measure Feature Method

A measure feature method will be subtyped as one of the following:

- coordinate measure feature method (e.g., measure a feature with a CMM)
- gage measure feature method (e.g., measure a feature with a gage)
- manual measure feature method (e.g., measure a feature by hand using a caliper, micrometer, or open set-up)
- microscope measure feature method (e.g., measure a feature by microscope)
- autocollimator measure feature method (e.g., measure a feature by an optical instrument for non-contact measurement of angles)
- profile projector measure feature method
- universal length measuring machine measure feature method
- laser radar measure feature method
- laser tracker measure feature method
- computed tomography measure feature method
- theodolite measure feature method
- calibrated comparator measure feature method

- external reference measure feature method
- other measure feature method

8.4.10 Work Instruction

A work instruction can be textual, a document file, an image file, or a video file.

8.5 Tracking information through the product lifecycle

QIF is constructed to enable a seamless flow of information from upstream applications such as a QIF Plan to downstream applications such as a QIF Results and to enable tracking information through a product's lifecycle. This can be enabled through separate or a common QIF documents and with QPIds. The **QIFDocument** is used as a unique container that holds applications such as the measurement plan and measurement results as well as other key *element*s.

8.6 QIF Plans data flow to results

QIF is constructed to enable a seamless flow of information from upstream applications such as QIF Products and/or QIF Plans to downstream applications such as QIF Results and to enable tracking information through a product's lifecycle. QIF Results is downstream from QIF Plans. A QIF Document can contain a QIF Plan with multiple QIF Results. QIF Results data files can be generated in the absence of a previously built QIF Plans file. However, if a QIF Plans file exists prior to the measurement execution, then a significant portion of the contents of the corresponding QIF Results file can be obtained from the shared QIF Documents or from the QIF Plans' QIF Document file.

The following items may normally be transcribed directly from a QIF Plan's QIF Document file to a corresponding QIF Results file:

- file units
- datum feature definitions
- datum definitions
- datum target definitions
- datum reference frames
- measurement resources
- product's part and assembly definitions
- feature definitions, nominals, and items
- characteristic definitions, nominals, and items

8.7 QIF Results reference to QIF Plans

Within a Results file there are two ways to express an association with the Plans file that was used in producing the Results file. They both use the *choice*-type **InspectionTraceability** *element* within the **MeasurementResults** *element*. The first choice records the QPId of the QIF Plans file through a **ReferencedQIFPlanInstance** *element.* The second choice indicates that the QIF Plans file is contained in the QIF Document file by setting the **ReferencedQIFPlan** *element* to "ThisFile".

8.8 Item tracking and persistence between QIF Plans and QIF Results

An application can associate multiple results items from Results files with plan items from a Plans file by using QIF persistent identifiers (QPId)s, as described in Clause 5.

8.9 High level description of QIF Plans.xsd

8.9.1 High level structure of the QIF Plans schema

This subclause describes the highest level *elements* of the QIF Plans information model. The QIF Plans schema model includes the information items from the QIFPlan.xsd schema file and several of the schema files in the QIF Library. The QIF Library files are incorporated into the schema by a chain of "*include*" directives starting in the QIFPlan.xsd schema file. *Element*s in the figure with + signs at the right have substructure that is not shown in the figure. All substructures are defined in the QIF Library files. A complete description of all QIF Plans data definitions and *element*s is available in text in the XML schema files and graphically in the QIF HTML files. QIFPlan.xsd also includes several *key*, and *keyref constraints* on *element*s and relationships between *element*s.

The **Plan** *element* is of type ***PlanType*** and is the highest level *element* of the QIF Plans model. An XML instance file written in conformance with the *QIFPlan.xsd* schema file may contain one **Plan** *element*. A ***PlanType***'s top level components are described in Figure 215. They include:

- **Version** – this optional *element* gives version information about the measurement plan. The ***VersionType*** includes the **VersionQPId** *element* which uniquely identifies this measurement plan.
- **RulesUsedQPId –** this optional *element* gives the QPId of the rules that were used in creating the plan. The rules may be in the same **QIFDocument** as the plan, or in a separate QIFDocument, or both (in which case the two sets of rules must be identical).
- **RulesToUseQPId –** this optional *element* gives the QPId of the rules to be used in making a more detailed plan from the plan or in generating a program from the plan (a DMIS program, for example). The rules may be in the same **QIFDocument** as the plan, or in a separate **QIFDocument**, or both (in which case the two sets of rules must be identical).
- **WorkInstructions –** this optional *element* captures any instructions necessary for executing the **Plan**.
- **ActionMethods** - this optional *element* captures any action methods necessary for executing the **Plan**.
- **Measurands** - this optional *element* captures any measurands used in the **Plan**.
- **LocalVariables** – this optional *element* captures any local variables used in the **Plan**.
- **PlanRoot** - this *element* gives information about the activities to be carried out in executing the measurement plan and how execution of the measurement plan is to be controlled. Structurally, the **PlanRoot** is at the top of a plan tree of **PlanElements**. All **PlanElement** instances occur within this plan tree.

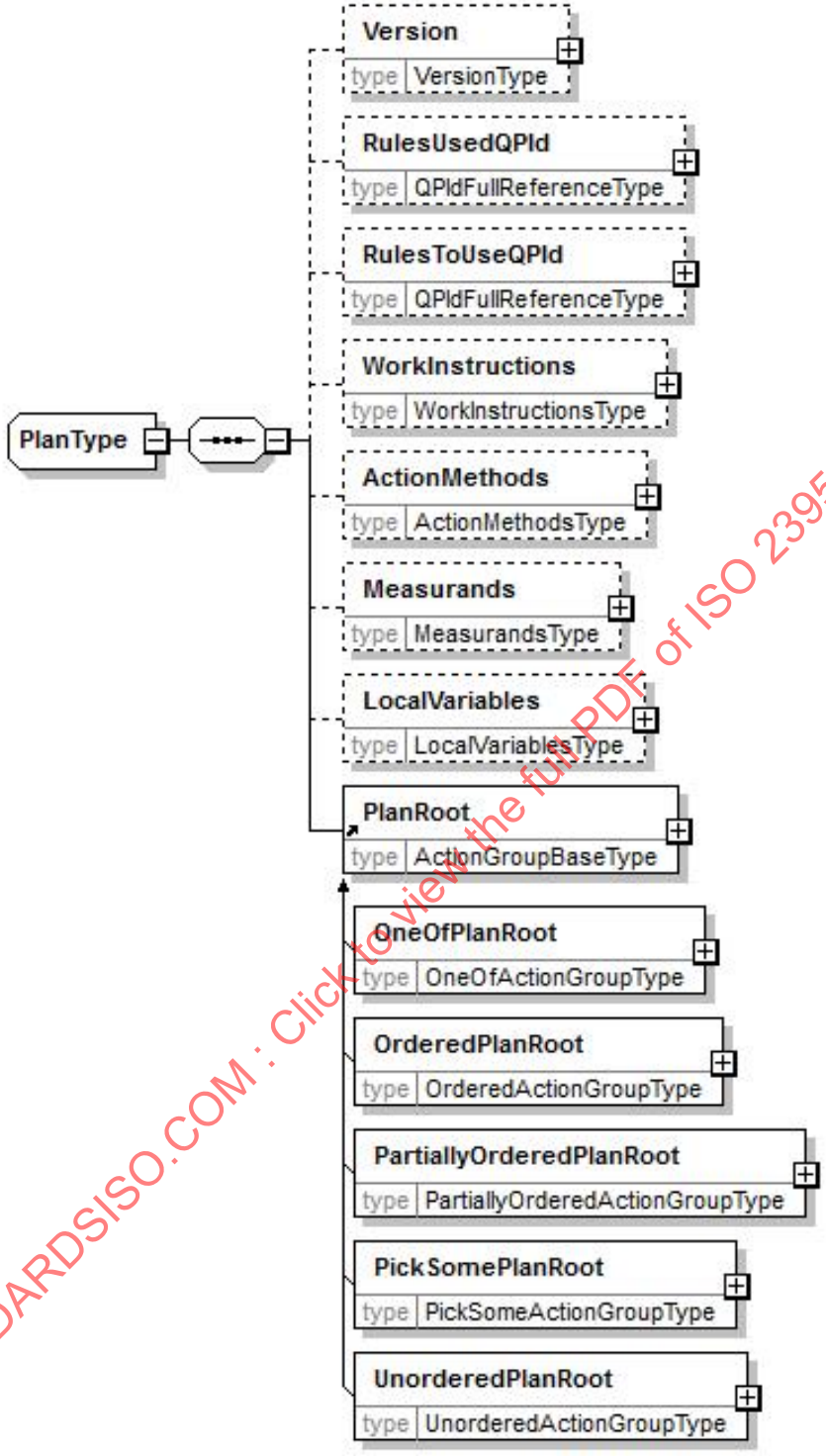


Figure 215 – Sub-*element*s of the Plan data type

The global **PlanRoot** *element*, introduced in Figure 215, is of ***ActionGroupBaseType*** that may be replaced (as provided by substitution group declarations) by any of the following *element*s representing action groups:

- **OneOfPlanRoot** – A **PlanRoot** which is an **OneOfActionGroup** that defines an action group for which exactly one action must be executed.
- **OrderedPlanRoot** – A **PlanRoot** which is an **OrderedActionGroup** that defines an action group for which all of the actions it contains must be executed in order.

- **PartiallyOrderedPlanRoot** – A **PlanRoot** which is a **PartiallyOrderedActionGroup** that defines an action group for which all of the actions it contains should be executed only after all of the predecessors of that action have been executed.
- **PickSomePlanRoot** - A **PlanRoot** which is a **PickSomeActionGroup** that defines an action group for which a specified number of actions it contains must be executed in any order.
- **UnorderedPlanRoot** - A **PlanRoot** which is an **UnorderedActionGroup** that defines an action group for which all of the actions it contains must be executed in any order of execution.

Figure 216 shows the major data *element*s introduced in the QIF Plan.xsd schema file with simplified relations between QIF Plans *element*s as well as with *element*s from the QIF library.

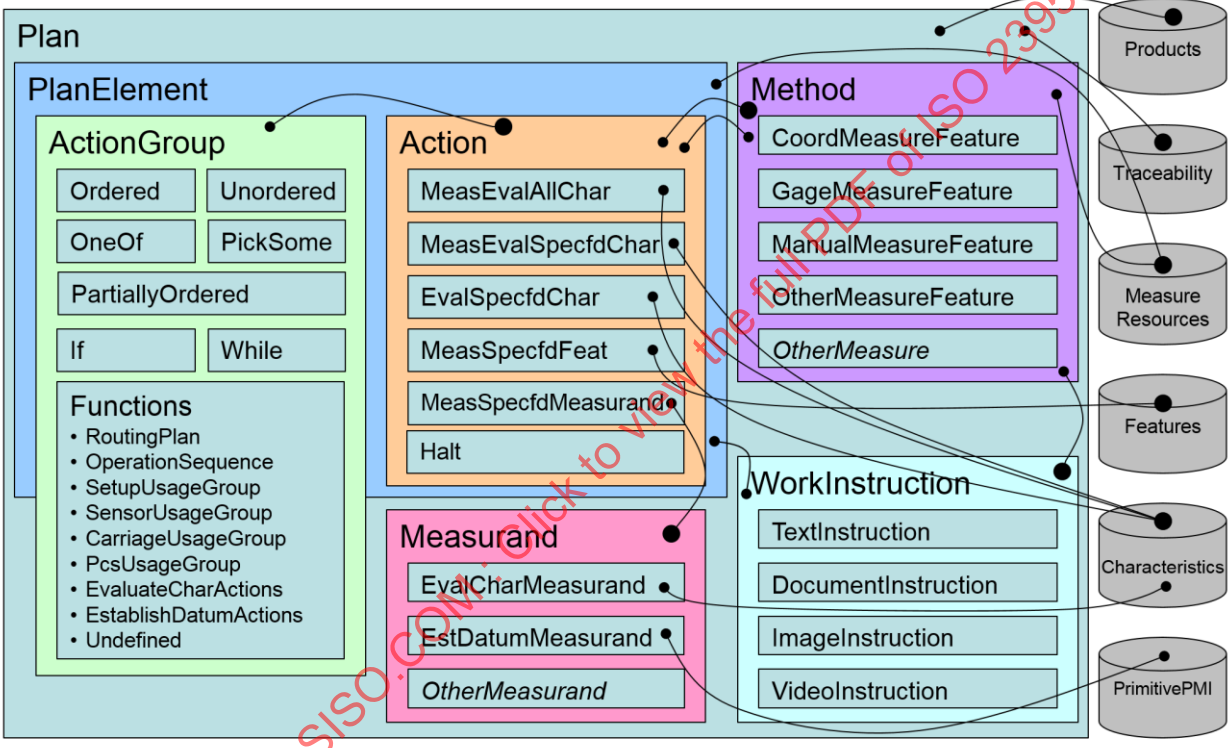


Figure 216 – QIF Plans Major *Element*s with Simplified Relations

8.9.2 Major *elements*

The major data *elements* of the QIF Plans information model are listed below and explained in both subclause 3.0 (Terms and Definitions) and subclause 8.4

- **Plan**
  - **Plan*Element***
    - **Action**
    - **ActionGroup**
- **Measurand**
- **Method**
- **WorkInstruction**

8.9.3 Simplified relationships *element*s

The lines from a small dot to a larger dot signify a one-to-many relationship whereas the lines from a small dot to a small dot signify a one-to-one relationship. High level relationships include:

- A measurement plan may have product information
- A measurement plan has traceability Information
- A measurement plan has a top level **Plan** *element*
- A **PlanElement** is an action or an action group
- An action group references zero to many work instructions
- An action group contains one to many actions
- An action references zero or one action method
- An action references zero to many preferred measurement resources
- An action references zero to many work instructions
- Some actions reference features
- Some actions reference characteristics
- Some actions reference measurands
- An action method references zero to many chosen measurement resources
- An action method references zero to many work instructions
- Some measurands evaluate a characteristic
- Some measurands establish a datum within a datum reference frame

8.9.4 Conditional Action Groups

Most of the action groups can be used as a plan root. The way they work is described following Figure 215. The ***IfActionGroupType*** and the ***WhileActionGroupType*** cannot be used as a plan root but are conditional action groups that can be used anywhere else a **PlanElement** may be used.

8.9.4.1 Expressions

The **IfActionGroup** and **WhileActionGroup** plan *elements* test Boolean expressions as a conditional action group. The Boolean expressions include comparing arithmetic expressions as well as Boolean constants *true* and *false* and the Boolean operations *and*, *or*, and *not*. The arithmetic expressions include the usual arithmetic operations (plus, minus, etc.), constants, and several types of arithmetic values extracted from features, characteristics, and other types of object. They also include using the values of plan variables. Plan variables are described in subclause 8.9.5.

The formal models of Boolean expressions and arithmetic expressions other than plan variables are given in the Expressions.xsd and GenericExpressions.xsd schema files and are described in Clause 6.

8.9.4.2 IfActionGroupType

The ***IfActionGroupType*** is derived from ***ActionGroupBaseType*** and represents an If-ElseIf-Else construct with zero to many Else-Ifs and an optional Else.

The **If** *element* is a test to evaluate and the **PlanElement** to execute if the test evaluates to true.

Each optional **ElseIf** is a test to evaluate and the **PlanElement** to execute if the test evaluates to true.

The optional **ElseDo** is a **PlanElement** to execute if none of the **If** or **ElseIf** tests evaluates to true.

To execute an ***IfActionGroupType***:

- Evaluate the test in the **If**. If it evaluates to true, execute the **PlanElement** in the **If** and return.
- Otherwise, evaluate the test in the next **ElseIf** if there is one. If it evaluates to true, execute the **PlanElement** in the **ElseIf** and return. Repeat until either return occurs or there are no more **ElseIf**s.
- Otherwise, if there is an **ElseDo**, execute the **PlanElement** in the **ElseDo** and return.
- Otherwise, return.

It is possible that no **PlanElement** will be executed when an ***IfActionGroupType*** is executed.

8.9.4.3 WhileActionGroupType

The **WhileActionGroup** plan *element* (of ***WhileActionGroupType***) represents a loop. It consists of a **BooleanExpression** and a **PlanElement**.

To execute a **WhileActionGroup**:

- evaluate the **BooleanExpression**. If the **BooleanExpression** evaluates to true, execute the **PlanElement**.
- Repeat evaluating and executing as long as the **BooleanExpression** evaluates to *true*.
- The first time the **BooleanExpression** evaluates to *false*, return.

If the **BooleanExpression** evaluates to *false* the first time it is evaluated, the **PlanElement** is never executed.

In order that a **While** will eventually stop looping, it is necessary that executing the **PlanElement** will eventually change something, and that the change will eventually cause the **BooleanExpression** to evaluate to *false*.

The **WhileActionGroup** has limited usefulness. Additional methods of referencing variables and modifying strings will be needed in future versions of QIF to make it more useful.

8.9.5 Plan Variables

As shown in Figure 215, a plan may include a **LocalVariables** *element* that is a set of variable declarations. Each declaration gives the name of the variable and its initial value.

The value of a variable may be changed by using a **VariableSet** element (which is of type ***VariableSetType***).

The value of a variable may be used in an expression by using a **VariableValue** *element* (which is of type ***VariableValueType***).

9 QIF Resources information model

9.1 Foreword

The Quality Information Framework (QIF) was developed by computer scientists and domain experts from the manufacturing quality community representing a wide variety of industries and quality measurement needs. Specifically for the QIF Resources work, past and current contributors include:

- Honeywell Federal Manufacturing & Technology
- Lockheed Martin
- Manufacturing Technology Centre
- Metrosage LLC
- Mitutoyo America Corporation
- National Institute of Standards and Technology
- Nikon Metrology
- Origin International, Inc.
- Rolls-Royce plc
- University of North Carolina, Charlotte

This clause was written by the QIF Resources Working Group, and given final approval for ANSI review by the DMSC's Quality Measurement Standards (QMS) Committee.

Version 3.0 of QIFResources has minor changes from version 2.1. The most notable changes are the addition of the LaserRadarType of instrument, the additional derived types CaliperDialType and CaliperDigitalType from the CaliperType, and the additional derived types MicrometerAnalogType and MicrometerDigitalType from the MicrometerType. These derived types are added to support QIFRules for DME selection.

9.2 Introduction

The Quality Information Framework (QIF) consists of a suite of information models that fit into one of two functional categories, QIF Library or QIF Applications. The QIF achieves system wide interoperability by designating a set of information models as part of a QIF Library. The QIF Library information models are common, reusable components. A QIF application area information model represents a unique application area, one of QIF model based definition (i.e., QIF Product), measurement resources (i.e., QIF Resources), measurement plans (i.e., QIF Plans), measurement rules (i.e., QIF Rules), measurement results (i.e., QIF Results), and measurement statistics (i.e., QIF Statistics). Because the QIF library components are referenced throughout the comprehensive QIF information model, it ensures interoperability and extensibility between any data producer and consumer that implements the QIF formats within their software.

An essential part of any dimensional measurement planning process is an adequate description of the measurement devices, tools, and auxiliary equipment available for possible application to the measurement task at hand. The measurement planner will then have the information necessary to make appropriate selections of measurement facilities for the inspection task at hand.

The QIF Resources clause is intended to supply a uniformly detailed set of this information. In addition to obviously required information, such as dimensional measuring equipment (DME) nomenclature, the clause describes everything sufficient to support high-level decisions about DME

capabilities and applicability, including description, location, and characterization of the DME capabilities. Examples of pertinent information are: achievable accuracy, measurement speed, workpiece size and mass capacity, etc. Additionally, historical information, such as calibration history and maintenance records, may be useful to downstream processes to establish measurement traceability and validity. Finally, information about associated auxiliary equipment (e.g. sensors, fixtures, etc.) may be required to determine fitness of the DME system for a particular measurement task.

9.3 Scope

This clause specifies the description and documentation of dimensional measurement resources, sufficient for use in generating a high level measurement plan for product certification, acceptance, or any other common application of dimensional measurement data.

9.3.1 Contents of this clause

This clause defines an information model for a portion of the QIF manufacturing quality information model designated QIF Resources. As described in Clause 5, an XML data file conforming to an XML schema model is called an instance file. The root of every QIF instance file is a QIFDocument, but different QIF instance files may focus on different QIF applications. A QIFDocument containing a Resources *element* may be regarded as a QIF Resources instance file.

As applicable, this clause covers the following topics:

- general descriptive information,
- sensors and sensor hardware,
- system calibration,
- system performance tests,
- workpiece capacity,
- geometric characteristics, and
- mechanical characteristics.

The information model consists of definitions for data types, *element*s, the logical relationships between them, and the semantics of the quality information. The information model, defined using the XML Schema definition language (XSDL), is scoped to be a digital data exchange mechanism that can be easily incorporated in application software developed by commercial solution vendors that implement manufacturing quality systems.

XSDL also supports the definition of rules and checks for validation of QIF instance files.

The QIFMeasurementResources.xsd schema file defines the top level information model for resources. QIF Resources also draws from *element*s in the QIFLibrary. The types and relationships of information contained in a QIF Resources instance file are governed by the schema. The way in which data is formatted for QIF Resources instance files is prescribed by the rules of XML and the rules for how instance files conform to schema files. All QIF XML schema files are a normative part of the QIF standard and can be downloaded at www.qifstandards.org.

9.4 QIF Resources Requirements

The specific technical requirements of QIF Resources are twofold:

- Describe various measurement resource entities and systems
- Provide the necessary data structures sufficient to support measurement planning and measurement results (particularly QIF Plans and QIF Results)

9.5 The QIF Resources data model

9.5.1 QIF Resources Instance Data

QIF Resources instance data is contained in a **MeasurementResources** *element*. This data will be found within a **QIFDocument** node, and may be the only contents of the QIF Document (making the QIF instance file a purely QIF Resources instance file), or it may sibling to QIF MBD, QIF Plans, QIF Results, etc. *element*s.

MeasurementResources is of type ***MeasurementResourcesType*** as shown in Figure 217.

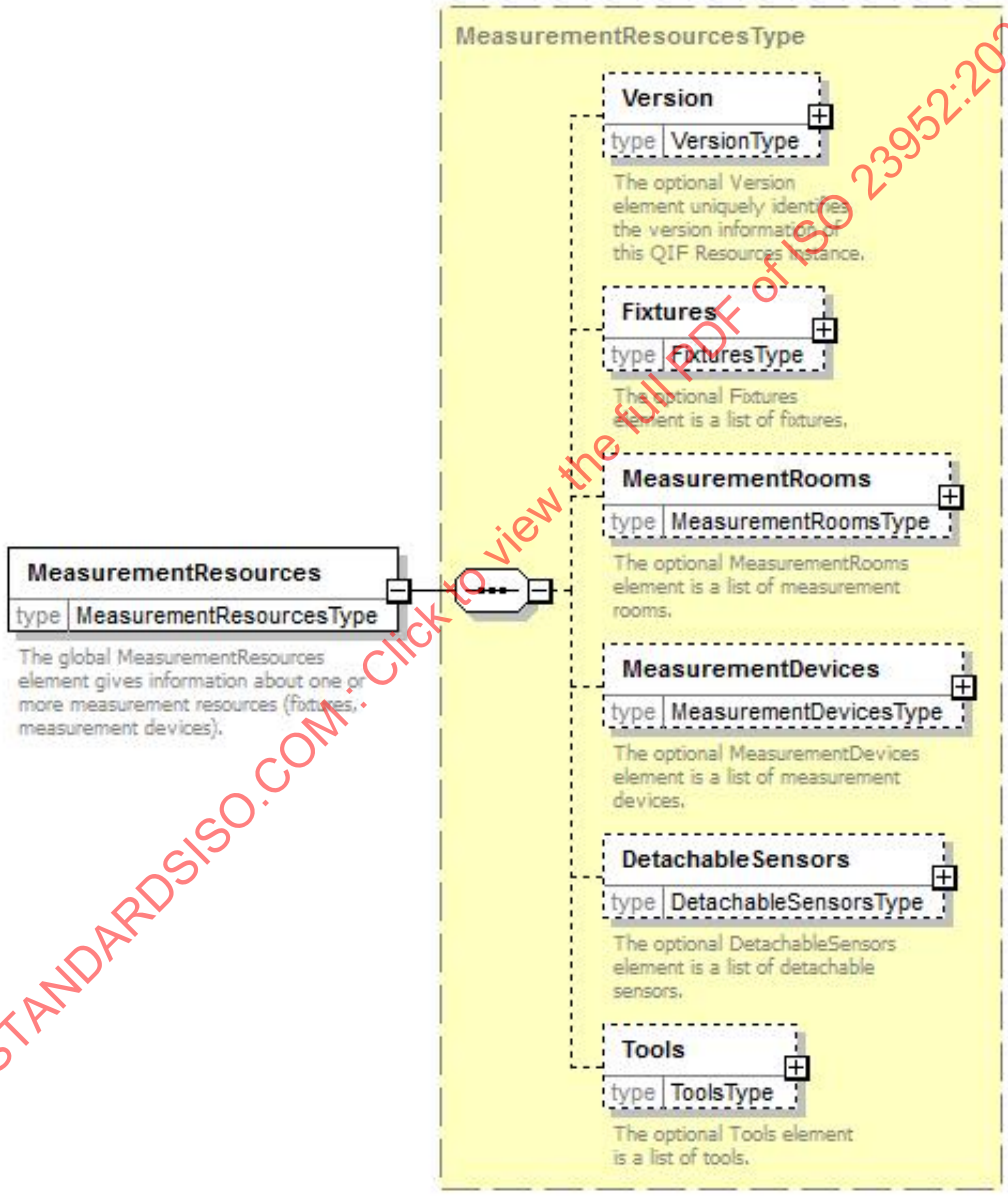


Figure 217 – QIF Resources instance data high level view

This type contains **Version** information (identification and traceability information specific to QIF Resources QIF instance file data) and **Fixture** information.

MeasurementRooms is an optional list of rooms in which measurements may be made.

MeasurementDevices will contain all *element*s that are derived from ***MeasurementDevicesType***. These include manual devices, CMMs, CT, laser trackers, etc. and is discussed in subclause 9.5.3: Measurement Devices.

The next 2 *element*s, **DetachableSensors** and **Tools**, provide the mechanisms by which all ***SensorType*** derived items will be ultimately be attached to a Measurement Device. ***SensorType*** types are measurement resources which are mounted on ***MeasurementDeviceType*** types in order to carry out a measurement. So for example, a tactile probe mounted on a CMM is an example of a ***SensorType*** mounted on a ***MeasurementDeviceType***.

DetachableSensors are where one can place ***DetachableSensorBaseType*** derived items, which are Sensors that are physically independent units that can be added and removed from a given measurement device.

Tools are where one can place ***ToolBaseType*** derived items, all of which fall either into the category ***ToolWithDetachableSensorsType*** or ***ToolWithIntegratedSensorType***. **ToolWithDetachableSensors** has an *element* where one can specify the detachable sensor(s) that are mounted onto the tool. An instance of ***ToolWithIntegratedSensorType*** is a special type of tool that has a ***SensorType***-derived item physically integrated into it.

For an example of how ***MeasurementDeviceType*** items, ***ToolBaseType*** items, and ***SensorType*** items all interact, see subclause 9.5.4: CMM.

9.5.2 *MeasurementResourceBaseType*

This is the base class for many of the types in QIFMeasurementResources.xsd. All types which represent measurement equipment of any type will be derived from this.

The ***MeasurementResourceBaseType*** is where many of the standard characteristics of any measurement device can be found, including a Name, Description, Manufacturer, Model Number, Mass, Size, and Location.

There is also an **Attributes** *element* which allows the user to specify various types of user defined data within this type, while still validating to the schema. (It is important to note, however, that use of these user-defined Attributes presents serious issues in terms of interoperability). The **Attributes** may include a persistent ID, which can either be in the form of a QIF-recommended QPId (ensuring uniqueness), or in the form of a non-standard ID string (the uniqueness of which must be enforced by the implementer).

Below, in Figure 218, Figure 219, and Figure 220 is an inheritance diagram for the types derived from ***MeasurementResourceBaseType***.

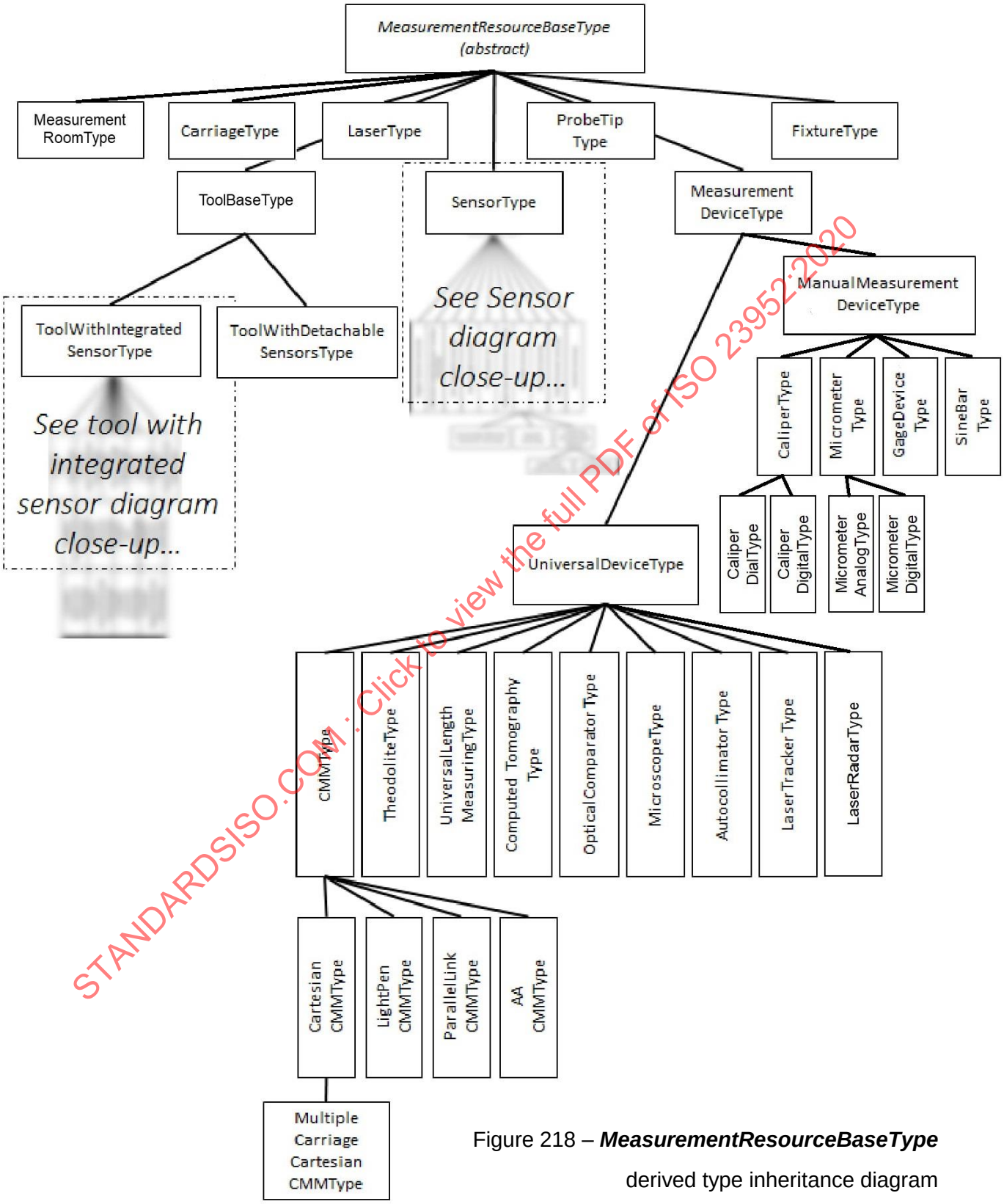


Figure 218 – ***MeasurementResourceBaseType***
derived type inheritance diagram

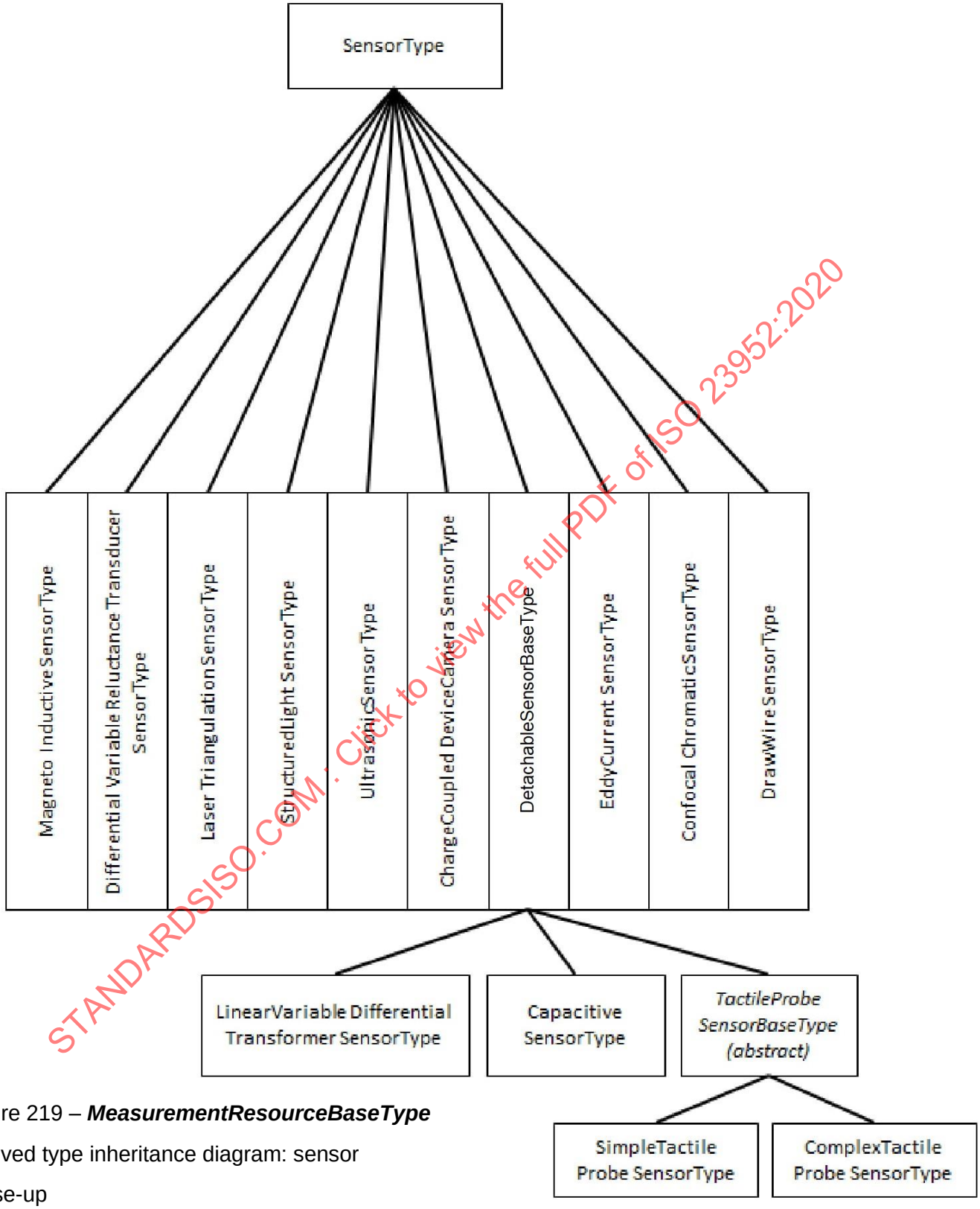


Figure 219 – ***MeasurementResourceBaseType*** derived type inheritance diagram: sensor close-up

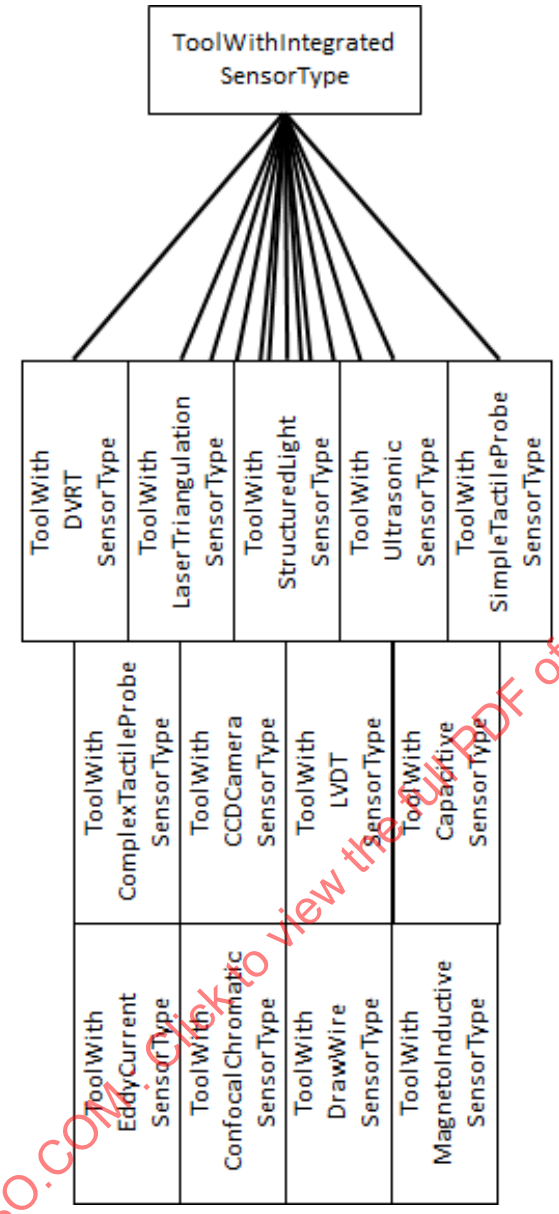


Figure 220 – ***MeasurementResourceBaseType*** derived type inheritance diagram: tool with integrated sensor close-up

9.5.3 **Measurement Devices**

A Measurement Device in QIF Resources is defined as a piece of measurement equipment that is directly used by the end user. As an example, a CMM is considered a Measurement Device, since it is directly manipulated by an end user, but the sensor which is attached to it is not considered a Measurement Device, since that is an instrument used by the CMM, and only indirectly used by the end user.

Measurement Devices are described by using ***MeasurementDeviceType***.

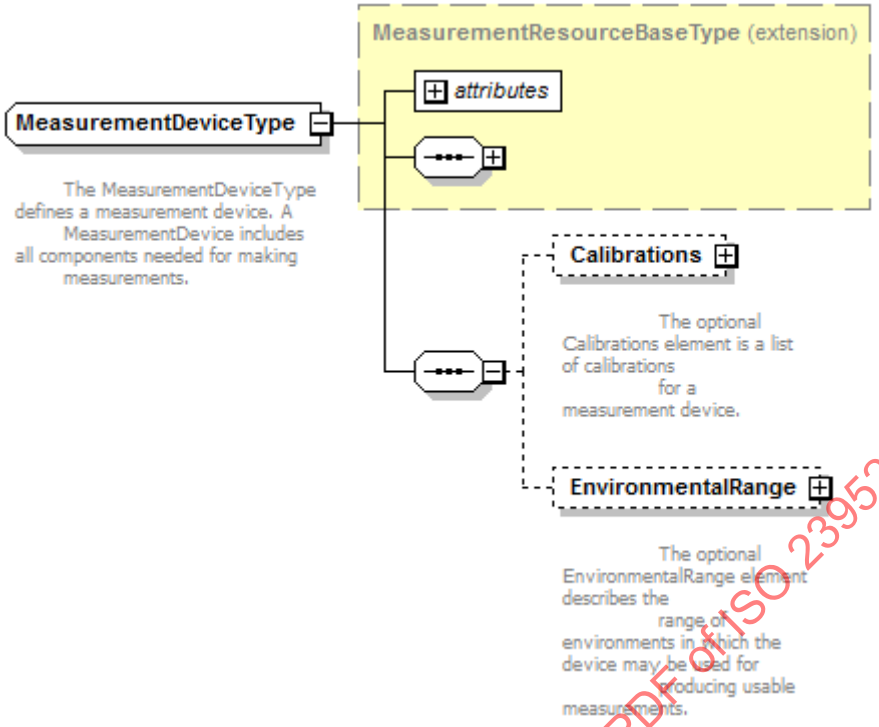


Figure 221 – ***MeasurementDeviceType*** overview

In addition to the *element*s inherited from ***MeasurementResourceBaseType***, the ***MeasurementDeviceType*** provides a **Calibration** *element* and an **EnvironmentalRange** *element*.

Generally speaking, while this type is not declared as abstract, it is preferred that derived types be used (see the remainder of this subclause).

9.5.3.1 Manual Devices

The ***ManualMeasurementDeviceType*** is used mainly for the purposes of derivation of the following types:

- ***CaliperType***
- ***MicrometerType***
- ***GageDeviceType***
- ***SineBarType***

Although the ***ManualMeasurementDeviceType*** is not declared abstract, it is recommended that this type be used only if none of its derived types adequately describe the intended device.

9.5.3.1.1 *CaliperType*

To describe a caliper, this type should be used. Aside from the inherited *element*s from its base classes, all *element*s of this type are optional.

9.5.3.1.2 *MicrometerType*

To describe a micrometer, this type should be used. Aside from the inherited *element*s from its base classes, all *element*s of this type are optional.

9.5.3.1.3 *GageDeviceType*

To describe a gage, this type should be used. Although this type provides no additional *element*s beyond the *element*s of its base classes, it does convey the fact that a gage is being described, as opposed to a generic “measurement device”.

9.5.3.1.4 *SineBarType*

To describe a sine bar, this type should be used. Aside from the inherited *element*s from its base classes, all *element*s of this type are optional.

9.5.3.2 Universal Devices

A Universal Device is a measurement device which typically satisfies the following conditions:

- not a manual device
- has a working volume
- has an effective working volume
- has a resolution.

Although the ***UniversalDeviceType*** is not declared abstract, it is recommended that this type be used only if none of its derived types adequately describe the intended device.

The **Resolution** *element* is optional and can be one of many different resolution types. See subclause 9.5.7 for more information about the available resolution types.

The **WorkingVolume** *element* is optional and can be one of many different working volume types. See subclause 9.5.8 for more information about the available working volume types.

The **EffectiveWorkingVolume** *element* is optional and can be one of many different effective working volume types. See subclause 9.5.8 for more information about the available effective working volume types.

The **TemperatureCompensation** *element* is an optional *element* that is used to specify the type of temperature compensation that the universal device uses. This can be a ***TemperatureCompensationEnumType***, or a custom string by using **OtherTemperatureCompensation**. The **OtherTemperatureCompensation** *element* should only be used if none of the enum values in ***TemperatureCompensationEnumType*** adequately express the temperature compensation of the device.

9.5.3.2.1 *CMMType*

This type is addressed in subclause 9.5.4.

9.5.3.2.2 *TheodoliteType*

The ***TheodoliteType*** is derived from ***UniversalDeviceType*** and represents a theodolite. In addition to the *element*s from its base classes, it provides a number of optional *element*s for describing the theodolite.

9.5.3.2.3 *UniversalLengthMeasuringType*

The ***UniversalLengthMeasuringMachineType*** is derived from ***UniversalDeviceType*** and represents a universal length measuring machine. In addition to the *element*s from its base classes, it provides a number of optional *element*s for describing the universal length measuring machine.

9.5.3.2.4 *ComputedTomographyType*

The ***ComputedTomographyType*** is derived from ***UniversalDeviceType*** and represents a computed tomography measuring machine. In addition to the *element*s from its base classes, it provides a number of optional *element*s for describing the computed tomography measuring machine.

9.5.3.2.5 *OpticalComparatorType*

The ***OpticalComparatorType*** is derived from ***UniversalDeviceType*** and represents a profile projector. In addition to the *element*s from its base classes, it provides a number of optional *element*s for describing the profile projector.

9.5.3.2.6 *MicroscopeType*

The ***MicroscopeType*** is derived from ***UniversalDeviceType*** and represents a microscope. In addition to the *element*s from its base classes, it provides a number of optional *element*s for describing the microscope.

9.5.3.2.7 *AutocollimatorType*

The ***AutocollimatorType*** is derived from ***UniversalDeviceType*** and represents an autocollimator. In addition to the *element*s from its base classes, it provides a number of optional *element*s for describing the autocollimator.

9.5.3.2.8 *LaserTrackerType*

The ***LaserTrackerType*** is derived from ***UniversalDeviceType*** and represents a laser tracker. In addition to the *element*s from its base classes, it provides a number of optional *element*s for describing the laser tracker.

9.5.3.2.9 *LaserRadarType*

The ***LaserRadarType*** is derived from ***UniversalDeviceType*** and represents a laser radar. In addition to the elements from its base classes, it provides a number of optional elements for describing the laser radar.

9.5.4 Coordinate Measuring Machine (CMM)

There are various types of CMM designs that are supported by QIF Resources. These are all derived from ***CMMType***. An inheritance diagram for the ***CMMType*** types can be seen in Figure 222.

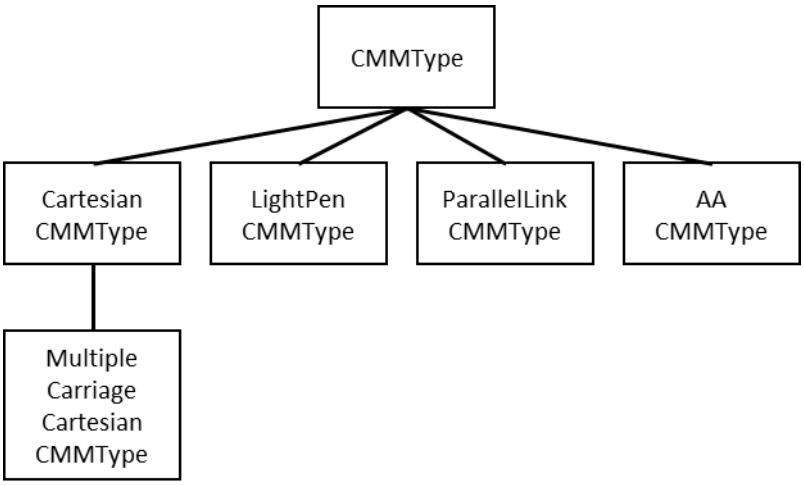


Figure 222 – CMM type inheritance diagram

Although ***CMMType*** is not abstract, it not recommended that it be used unless *none* of its derived types describe the particular CMM in question.

For single carriage Cartesian CMMs, the ***CartesianCmmType*** should be used. The ***MultipleCarriageCartesianCMMType*** is meant for more complex CMMs with multiple carriages.

Cartesian CMM types make use of the ***MeasurementDeviceScaleType*** to describe the scales of the CMM.

9.5.4.1 CMM Accuracy Tests

Each CMM accuracy test, addressed in the two subclauses below, is accompanied by an ***EnvironmentalRangeType*** which indicates the range of conditions under which the accuracy test is valid.

There is also an **AccuracySource** *element* which is represented by an ***AccuracySourceType*** type. This *element* is meant to indicate whether the accuracy test values originated from the equipment manufacturer, or from a custom test of some type.

9.5.4.1.1 *Cartesian CMM Accuracy Tests*

There are various methods for specifying Cartesian CMM accuracy tests. These types are applied via the substitution group ***CartesianCMMAccuracyTest***. The available types are:

- ***CartesianCMMFPSTest***
- ***CartesianCMMB89Test***
- ***CartesianCMMISO10360Test***
- ***CartesianCMMPointAccuracyTest***

9.5.4.1.2 *Articulating Arm CMM Accuracy Tests*

There are various methods for specifying AA CMM accuracy tests. These types are applied via the substitution group ***AACMMAccuracyTest***. The available types are:

- ***AACMMB89Test***
- ***AACMMISO10360Test***
- ***AACMMPointAccuracyTest***

9.5.4.2 CMM Speeds

The ***CMMSpeedsType*** substitution group is used to specify CMM speeds for Cartesian CMMs and Parallel Link CMMs. There are two types of CMM speed types:

- ***CartesianCMMSpeedsType***
- ***ParallelLinkCMMSpeedsType***

9.5.4.3 CMM Geometry Types

There are various types of CMM geometries that can be described using QIF Resources. Images of these CMM geometries are depicted in Figure 223.

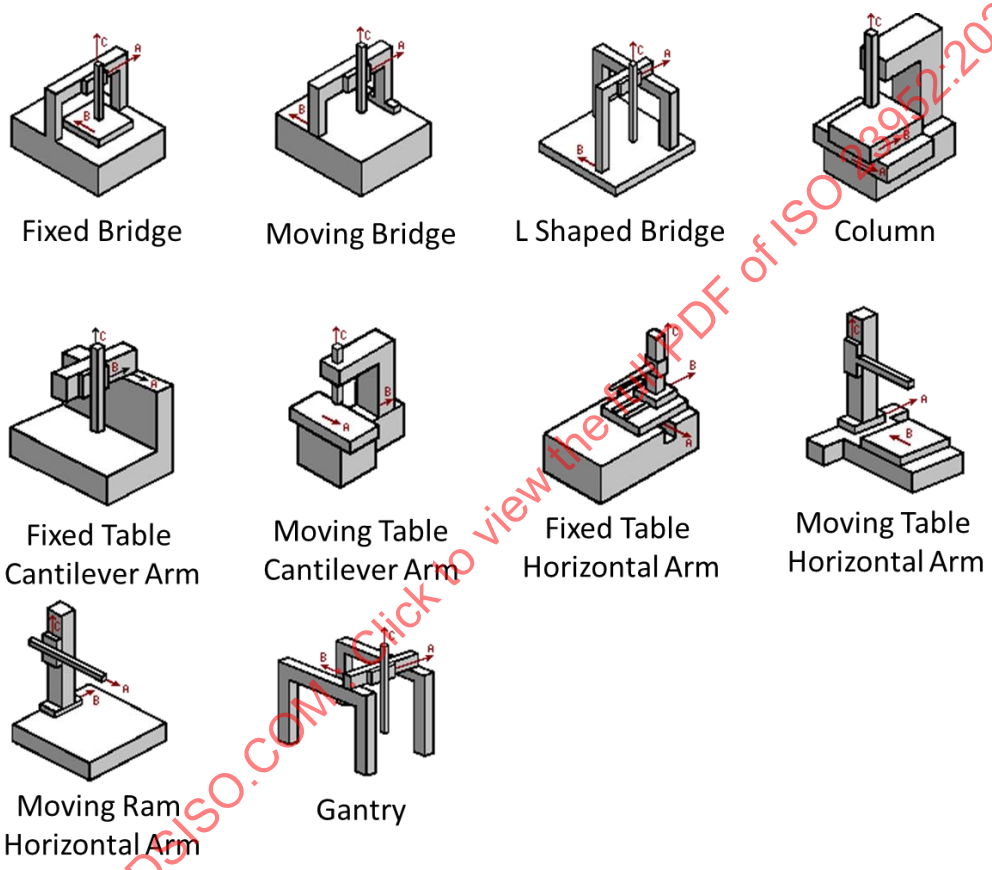


Figure 223 – Cartesian CMM geometry types

The CMM geometries depicted in Figure 223 correspond to the following enumeration values defined in the type ***CartesianCMMGeometryEnumType***:

- FIXED_BRIDGE
- MOVING_BRIDGE
- L_SHAPED_BRIDGE
- COLUMN
- FIXED_TABLE_CANTILEVER_ARM
- MOVING_TABLE_CANTILEVER_ARM
- FIXED_TABLE_HORIZONTAL_ARM
- MOVING_TABLE_HORIZONTAL_ARM
- MOVING_RAM_HORIZONTAL_ARM
- GANTRY

If none of the values listed above correctly describe the CMM in question, then the **OtherCartesianCMMGeometry** string value *element* can be used.

9.5.5 Sensors and Tools

Sensors and tools are ***MeasurementResourceBaseType***-derived types which exist in order to be mounted on a ***UniversalDeviceType***-derived resource. For example, a tactile probe (a sensor) can be mounted on a CMM (a universal device).

9.5.5.1 Detachable vs Integrated Sensors

There are two ways in which sensors and tools can interact with a ***UniversalDeviceType***-derived resource.

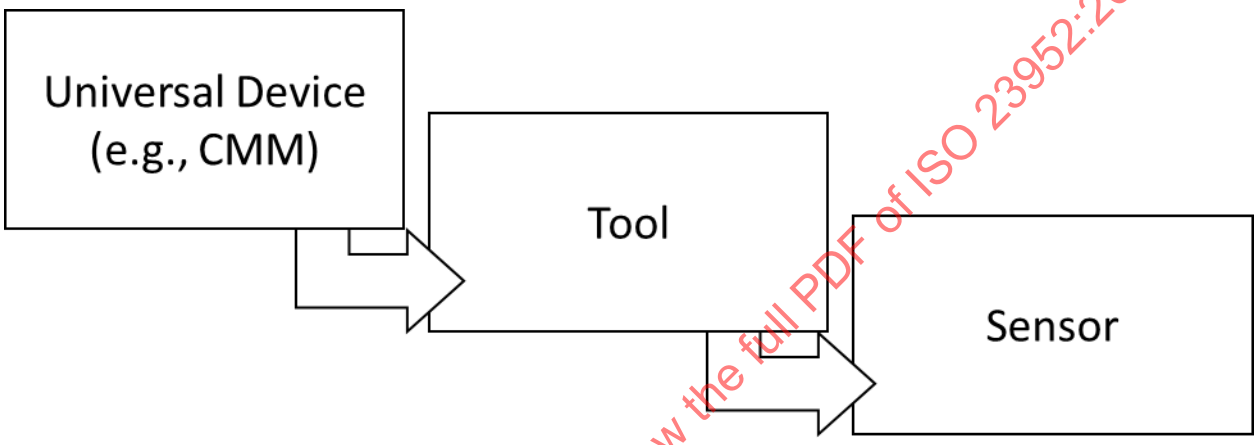


Figure 224 – Method 1 for mounting a sensor on a universal device

The first method is depicted in Figure 224. A ***ToolBaseType***, specifically, a type derived from ***ToolWithDetachableSensorsType***, is mounted onto a ***UniversalDeviceType***-derived resource. Then, a ***SensorType***-derived resource is mounted onto the Tool. This method should be used when the tool is a separate physical entity from the sensor device.

In cases when the sensor is physically integrated with the tool, then the second method should be used. This can be used, as depicted in Figure 225, below.

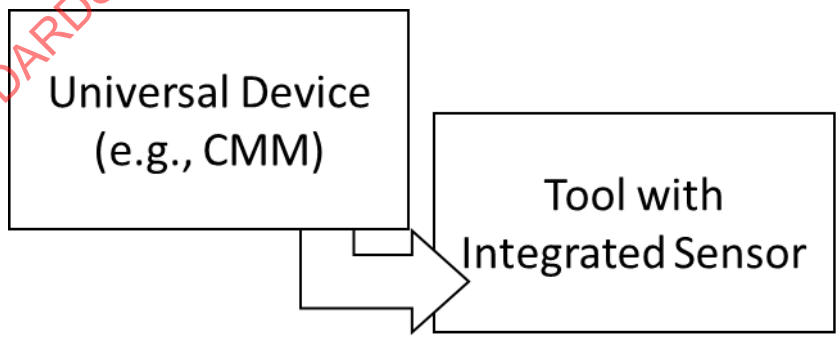


Figure 225 – Method 2 for mounting a sensor on a universal device

For this situation, QIF Resources provides a special set of tools with integrated resources. These are the types that are derived from the abstract type ***ToolWithIntegratedSensorType***. The sensor *element*s which are found on the ***ToolWithIntegratedSensorType***-derived types are the

SensorType-derived measurement resources, which means that this method and the first method described above are both capable of carrying the same sensor data.

9.5.5.2 Sensor Types

This subclause will address the types of sensors that are available in QIF Resources. Each sensor and its corresponding tool-integrated sensor will be addressed simultaneously, since they are so similar. (See subclause 9.5.5.1 for more information on tools and sensors).

There are various types of sensors which are supported by QIF Resources.

Below is a listing of the sensors:

- Linear Variable Differential Transformer Sensor: ***LinearVariableDifferentialTransformerSensorType*** and ***ToolWithLVDTSensorType***
- Capacitive Sensor: ***CapacitiveSensorType*** and ***ToolWithCapacitiveSensorType***
- Eddy Current Sensor: ***EddyCurrentSensorType*** and ***ToolWithEddyCurrentSensorType***
- Confocal Chromatic Sensor: ***ConfocalChromaticSensorType*** and ***ToolWithConfocalChromaticSensorType***
- Draw Wire Sensor: ***DrawWireSensorType*** and ***ToolWithDrawWireSensorType***
- Magneto Inductive Sensor: ***MagnetoInductiveSensorType*** and ***ToolWithMagnetoInductiveSensorType***
- Differential Variable Reluctance Transducer Sensor: ***DifferentialVariableReluctanceTransducerSensorType*** and ***ToolWithDVRTSensorType***
- Laser Triangulation Sensor: ***LaserTriangulationSensorType*** and ***ToolWithLaserTriangulationSensorType***
- Structured Light Sensor: ***StructuredLightSensorType*** and ***ToolWithStructuredLightSensorType***
- Ultrasonic Sensor: ***UltrasonicSensorType*** and ***ToolWithUltrasonicSensorType***
- Simple Tactile Probe Sensor: ***SimpleTactileProbeSensorType*** and ***ToolWithSimpleTactileProbeSensorType***
- Complex Tactile Probe Sensor: ***ComplexTactileProbeSensorType*** and ***ToolWithComplexTactileProbeSensorType***
- Charge Coupled Device Camera Sensor: ***ChargeCoupledDeviceCameraSensorType*** and ***ToolWithCCDCameraSensorType***

9.5.5.2.1 *Touch Trigger sensors*

There are 2 touch trigger sensor types available in QIF Resources: the Simple Tactile Probe Sensor and the Complex Tactile Probe Sensor.

The Simple Tactile Probe Sensor is meant to represent a single tip with no articulation. The types for this simple probe are ***SimpleTactileProbeSensorType*** and ***ToolWithSimpleTactileProbeSensorType***.

The Complex Tactile Probe Sensor is meant to represent a tactile probing system that does not fall into the category of Simple Tactile Probe Sensor. This could include a sensor tip that is capable of articulation in some way, or a multiple stylus tipped probing system.

When a Complex Tactile Probe System is used, a measurement point item or actual can reference a specific tip on that probe. This is done by using referencing the ***ProbeTipType***. This type is found under the **LocatedTips** *element* of the complex probe.

9.5.6 Rotary Table

The ***RotaryTableType*** represents a rotary table. It can be attached to a ***CartesianCMMType*** or to a ***ComputedTomographyType*** via their **RotaryTable** *element*.

9.5.7 Resolution Types

There are various methods for characterizing the resolution of a measurement device. These types are used for ***UniversalDeviceType***, and are applied via a substitution group. The available types are:

- Cartesian Resolution: ***CartesianResolutionType***
- Linear Resolution: ***LinearResolutionType***
- Spherical Resolution: ***SphericalResolutionType***
- User Defined Resolution: ***UserDefinedResolutionType***

9.5.8 Working Volumes

There are various methods for characterizing the working volume and effective working volume of a measurement device.

For any given machine geometry, there is the ***WorkingVolumeBaseType*** and the ***EffectiveWorkingVolumeBaseType***, each of which is characterized by different XML types.

WorkingVolumeBaseType and ***EffectiveWorkingVolumeBaseType*** are used for all ***UniversalDeviceType***-derived types, and are applied via a substitution group. The working volume types are used for some sensors as well.

- Cartesian Working Volume: ***CartesianWorkingVolumeType*** and ***EffectiveCartesianWorkingVolumeType***
- Closed Shell Set Working Volume: ***ClosedShellSetWorkingVolumeType*** and ***EffectiveClosedShellSetWorkingVolumeType***
- Cylindrical Working Volume: ***CylindricalWorkingVolumeType*** and ***EffectiveCylindricalWorkingVolumeType***
- Spherical Working Volume: ***SphericalWorkingVolumeType*** and ***EffectiveSphericalWorkingVolumeType***
- User Defined Working Volume: ***UserDefinedWorkingVolumeType*** and ***EffectiveUserDefinedWorkingVolumeType***

9.5.9 Axis Types

The axis types are used for characterizing the type of axis that is referenced by the ***UserDefinedResolutionType*** and ***UserDefinedWorkingVolumeType***. It is applied via a substitution group.

There are two types of axis types:

- Linear Axis: ***LinearAxisType***
- Rotary Axis: ***RotaryAxisType***

9.5.10 Environmental Data

The ***EnvironmentalRangeType*** is used by ***MeasurementDeviceType***, ***SensorType***, and ***MeasurementDeviceAccuracyBaseType*** to describe a range of environmental parameters (temperature, air pressure, humidity, vibration, etc.).

In the case of the ***MeasurementDeviceType*** and ***SensorType***, the ***EnvironmentalRangeType*** is used to describe the acceptable range of environmental conditions for measurement.

In the case of the ***MeasurementDeviceAccuracyBaseType***, the ***EnvironmentalRangeType*** is used to describe the acceptable range of environmental conditions for the given accuracy statement.

9.5.11 Calibrations

The ***CalibrationType*** is used by ***MeasurementDeviceType*** to characterize calibrations that have taken place on the given measurement device.

9.5.12 *MeasurementRoomType*

The ***MeasurementRoomType*** describes an environmentally controlled room in which measurements may be done. It provides for controls on temperature and humidity.

10 QIF Rules information model

10.1 Introduction

10.1.1 Why

In an attempt to encourage uniformity in measurement strategies across a large corporation or supply chain, a common approach is to distribute a set of measurement rules (sometimes called measurement templates). This will give the Coordinate Measuring Machine (CMM) (or any other Dimensional Measurement Equipment (DME)) inspection program creator a guideline for implementing programs that conform to best organizational practices.

Using the QIF Rules model, any organization can build a file conforming to the model and containing a set of rules meant to give a dimensional measurement process planner, CMM programmer, or CMM software package a set of standard practices regarding:

- how individual features of a product should be measured based on selected criteria (such as applicable characteristics, tolerances, feature geometry type, feature size, datum, etc.) and/or measurement requirements (such as uncertainty, setup, and environment)
- how measurement resources should be selected, based on similar criteria plus part and DME parameters.

Benefits of using rules are reduced uncertainty, consistent planning, and planning automation. The QIF Rules model does not propose any specific set of rules. It only provides a format for rules files in order that sets of rules may be exchanged easily and unambiguously.

10.1.2 What

Rules may be made for the following items on a per feature basis:

- the number or density of CMM hit points to use
- the pattern in which the CMM hit points should be distributed (point sampling strategy)
- the feature fitting algorithm to use.

Rules may be made for DME selection on a per feature basis or more broadly.

A QIF Rules instance file will consist of a set of rules, each of which normally contains an “if” part that is a “Boolean Condition” and a corresponding “then” part that specifies what action is allowed or required if the “if” part is true. A Boolean Condition is a statement which can be unambiguously evaluated as true or false. For example: “Joe’s car is red” would evaluate to “true” if and only if the color of Joe’s car is red. Any given Boolean Condition may have one or more corresponding actions that should be taken depending on the value of the Boolean Condition. For example, if “Joe’s car is red” evaluates to true, the corresponding action may be requested: “Borrow Joe’s car.” QIF Rules simply consists of a set of Boolean Conditions pertaining to the feature, characteristic, or part in question, and a set of measurement-related actions that should be carried out depending on these conditions.

The QIFRules.xsd schema file defines the top level information model for rules. QIF Rules also draws from *element*s in the QIF Library, particularly those in the GenericExpressions.xsd and Expressions.xsd schema files. For DME selection, QIF Rules references types defined in QIFMeasurementResources.xsd.

A QIFDocument containing a **Rules** *element* may be regarded as a QIF Rules instance file. It is anticipated that typically, a QIF Rules file will contain no other application and will not contain items such as features, characteristics, and traceability that are not required for stating rules.

Using a QIF Rules file during planning, however, will require using other information items such as features and characteristics.

The measurement rules described in this clause are *not* an attempt to give DME programmers a precise set of instructions regarding how to carry out any given measurement on a given part. Such a detailed set of instructions would be transmitted using QIF Plans.

10.1.3 How

QIF rules files are intended to be used during measurement planning. QIF specifies the model of a rules file. The rules files themselves are written by any QIF user who wants to put a set of rules into a well-defined, formal, standard format. Although examples of rules files have been built during QIF development, QIF does not require or recommend any particular set of rules. Using the QIF rules model does not guarantee or even imply that a given set of rules is usable or even makes sense; it is up to the organization writing the rules to see to that. It is intended that there be three main parties involved in the use of rules:

- a person or group that writes the rules
- a person or group that decides how the rules are to be applied
- a person or software system that uses the rules during process planning or DME programming.

The party writing the QIF Rules instance file should decide at the outset in what way the rules are intended to be used. If a QIF Rules instance file is to be used in an automatic process planning application or an automatic DME programming system, it will be necessary to write enough rules to cover all the decisions that the software may be required to make. This is expected to be difficult. The model does not attempt to ensure that a set of rules conforming to the model will even be consistent, let alone complete. If the intent of writing the QIF Rules instance file is to encode a (possibly existing) set of natural language rules in a formal way, the job of the rules writers is expected to be more straightforward. However, since natural language may be used to say almost anything, it is possible that the QIF Rules model will not be sufficiently expressive to state some desired rules. It is the intent of the DMSC that future versions of the rules model be improved by adding functionality that is found by users to be needed.

The party deciding how the rules will be used may be the same one that wrote the rules or it may be a manager, process planner, or DME programmer. The most lax decision would be to use the rules merely as suggestions. Another possibility is to require that the rules be followed, but where there are alternatives that are all allowed by the rules, use the desirability values in the rules only as suggestions, and let the person or software using the rules decide among the alternatives. A third possibility is to require that the rule user use the given desirability values to find and select the most desirable alternative.

The party using the rules during process planning or DME programming will presumably follow the organization's rules for how a set of rules modeled using QIF will be applied. Applying the rules will be a non-trivial undertaking. If this party is a software application, a great deal of functionality will

need to be programmed into it. If this party is a person, some form of translation from the QIF rules instance file into an easily understandable form will be needed.

10.1.4 Changes from QIF 2.1

As compared with version 2.1 of QIFRules published as ANSI/QIF Part 6–2015, this version of QIF Rules contains a major addition. That is a model of rules for selecting dimensional metrology equipment (DME). There was no such model in previous versions of QIF. Most of the formal DME selection rules model is contained in the QIFRules.xsd schema file. Minor additions and improvements needed by the DME selection rules model were made in other schema files (Expressions.xsd, GenericExpressions, and QIFMeasurementResources.xsd).

10.2 Design principles of QIF Rules

This subclause gives an overview of the QIF Rules information model.

10.2.1 Structure of a Rule

A QIF rules file consists of a set of rules, each of which (usually) has a “Boolean Condition” (a statement which can be unambiguously evaluated as true or false) and always has a corresponding action, as in the following pseudocode model:

```
If (“Boolean Condition”) then
  {
     Corresponding “action”
  }
```

For example, if “the feature is a cylinder” evaluates to true, the corresponding action may be requested: “measure 13 points”.

If the Boolean Condition is omitted, the action always applies.

The following Boolean Conditions are supported by QIF Rules:

- Characteristic Type (is equal to a given characteristic type)
- Feature Type (is equal to a given feature type)
- Shape class (part shape class is equal to one of prismatic, thin walled, etc.)
- Is feature a datum?
- Is feature internal?
- User defined Sampling Category (compared to a given value)
- Boolean expressions using the Boolean constants `True` and `False` and the Boolean operators `BooleanEqual`, `And`, `Or`, and `Not`.
- Arithmetic comparisons that yield a Boolean result: `ArithmeticEqual`, `GreaterThan`, `LessThan`, `GreaterOrEqual`, and `LessOrEqual`.

The arithmetic comparisons act on arithmetic expressions, which include:

- arithmetic constants
- arithmetic operations `Negate`, `Plus`, `Minus`, `Times`, `DividedBy`, `Max`, `Min`
- parameters (numerical *element*s or *attribute*s) of characteristics, features, parts, and DMEs
- characteristic tolerances
- feature size, area, or length

The actions available for rules differ between feature rules and DME selection rules. Also, the two types of rule have differing structure, as described in the following subclauses.

The rules are to be evaluated in an environment in which the **SamplingCategory** has been set to some positive integer so that the ***SamplingCategoryIsType*** Boolean expression can be evaluated. For example, the values 1 and 2 might represent `first article inspection` and `process control`, or the values 1 through 4 might represent `LOW`, `MEDIUM`, `HIGH`, and `SUPER`, respectively.

10.2.2 Feature Rules

For features, the following actions can be triggered by the above Boolean Condition:

- Number of measurement points
- Measurement point density
- Point sampling strategy (taken from ISO-14406:2010)
- Feature fitting algorithm to use

Here is an example in pseudocode:

```
If (FeatureType is Cylinder) then
  {
     Number of Points = 64
  }
```

For features, this if/then clause could optionally be followed by a series of “else if” and finally an “else” statement. These “else if” and “else” statements are modeled after the commonly seen “if/else if/else” statements found in applications of computer science.

Here is a simple example in XML. In this example, we test whether the surface area of the feature is greater than 10 units of area. If it is, then a minimum of 25 points is required. Otherwise (“else”), a minimum of 10 points is required.

```
<MaxFeatureRules n="2">
  <IfThenSurfaceRule>
    <GreaterThan>
      <FeatureArea/>
      <ArithmeticConstant val="10"/>
    </GreaterThan>
    <ThenPoints>
      <MinPoints>25</MinPoints>
    </ThenPoints>
  </IfThenSurfaceRule>
  <Else>
    <ThenPoints>
      <MinPoints>10</MinPoints>
    </ThenPoints>
  </Else>
</MaxFeatureRules>
```

When a rule is applied to a feature, if more than one characteristic applies to the feature, and a condition uses a characteristic type test, the condition should be tested for each characteristic applied to the feature.

Since measurement points are taken on Features (not Characteristics), there may often arise a situation where two or more conflicting Rules may apply to a given Feature. When this is the case, the Rule with the higher number of points should be used. If this happens when two sampling strategies are specified, then it is up to the evaluating system to make the appropriate decision. (Exception: the ***IfThenElseFeatureRulesType***; see subclause 10.4.2.1.)

10.2.3 DME Selection Rules

The DME selection rules model is designed to allow the rule writer to write a wide range of types of rule. A DME selection rule may be very general, such as, “A caliper must not be used”, or it may be very specific, such as, “If the feature is internal, and the feature is a cylinder, and the diameter is greater than 3.2, and the characteristic is diameter, then the cartesian coordinate measuring machine whose QIF ID is 1234 must be used”.

DME selection rules can be used to:

- require using a particular DME or any of a set of DMEs
- authorize (but not require) using a particular DME or any of a set of DMEs
- prohibit using a particular DME or any of a set of DMEs.

DME selection rules are intended to be used during measurement planning. A set of rules conforming to the model may be written so as to leave the planner with no choice (useful if the planner is an automated system). A rule set may also be written so as to provide the planner with a set of acceptable alternatives from which the planner may choose. Where a rule set leaves alternatives, the desirability of each alternative may be specified as a number, so that the choice can be made algorithmically if that is what the planner (or the planner’s organization) wishes to do.

Specifying the set of DMEs to which the rules apply during a measurement planning session is the responsibility of the organization in which measurement planning is taking place. For example, it might be any of (but is not limited to):

- a list of DMEs on hand
- commercially available measurement equipment, e.g., equipment found in a catalog
- a QIF measurement resources file.

In a QIF instance file, the rule above that has no “If” part and prohibits the use of a caliper would be:

```
<DMEDecisionRule>
  <DMEThen n="1">
    <DMEDecisionClass>
      <MustNot/>
      <DMEClassName>CALIPER</DMEClassName>
    </DMEDecisionClass>
  </DMEThen>
 </DMEDecisionRule>
```

And the complex rule requiring the coordinate measuring machine with QIF id 1234 would be:

```
<DMEDecisionRule>
  <And n="4">
    <FeatureTypeIs val="CYLINDER"/>
    <FeatureIsInternal/>
    <CharacteristicIs val="DIAMETER"/>
```

```
        <GreaterThan>
          <ArithmeticFeatureParameter>
            <Parameter>Diameter</Parameter>
          </ArithmeticFeatureParameter>
          <ArithmeticConstant val="3.2"/>
        </GreaterThan>
      </And>
      <DMEThen n="1">
        <DMEDecisionId>
          <Must/>
          <DMEId>1234</DMEId>
        </DMEDecisionId>
      </DMEThen>
    </DMEDecisionRule>
```

10.3 QIF Rules schema files

The QIF Rules schema model includes the information items from the QIFRules.xsd schema file and several of the schema files from the QIF Library. As with other QIF components, the QIF Library schema files are incorporated into the model by a chain of "*include*" directives starting in the QIFRules.xsd schema file. The following QIF Library schema files are required:

- Expressions.xsd
- IntermediatesPMI.xsd
- PrimitivesPMI.xsd
- Units.xsd
- GenericExpressions.xsd
- Primitives.xsd.

10.4 QIF Rules *element*s and data types

This subclause describes the major *element*s and types in the QIF Rules model.

10.4.1 QIF Rules top level

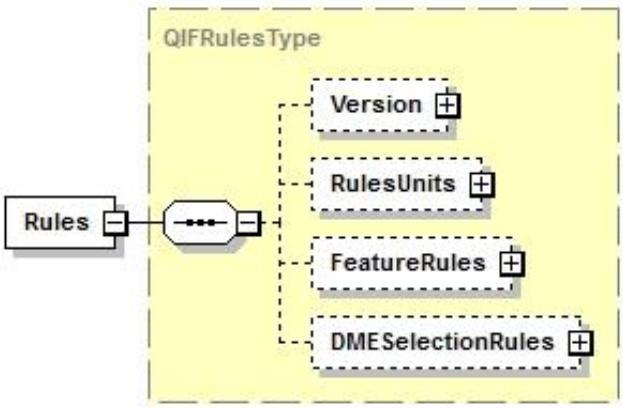


Figure 226 – **Rules** *element* and ***QIFRulesType***

The top-level data object under **QIFDocument** in a QIF Rules instance file is a **Rules** *element* of type ***QIFRulesType***, as shown in Figure 226. The **Rules** *element* contains an optional set of rules

for features in its **FeatureRules** *element* and an optional set of rules for selecting dimensional measurement equipment (DMEs) in the **DMESelectionRules** *element*.

10.4.2 Feature Rules

The **FeatureRules** *element* shown in Figure 226 is of type ***FeatureRulesType***, as shown in Figure 227.

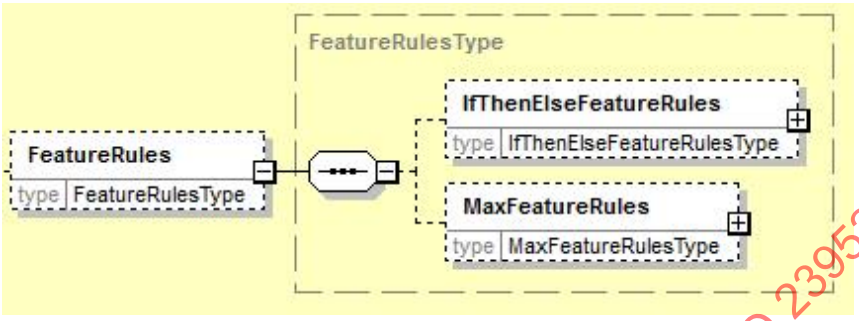


Figure 227 – **FeatureRules** *element* and ***FeatureRulesType***

The ***FeatureRulesType*** models a collection of sets of rules for selecting quality measurement items. Currently it contains a complete model of rules for selecting any or all of the following for a given feature:

- the quantity (number or density) of hit points
- the strategy for selecting hit points
- a feature fitting algorithm.

As shown in Figure 227, the ***FeatureRulesType*** provides two methods of making selections, **IfThenElseFeatureRules** and **MaxFeatureRules**. Either method or both methods may be used. If the **IfThenElseFeatureRules** and **MaxFeatureRules** are both used for quantity of points, both sets of rules should be applied and the greatest value should be used. Also, if both are used and they give different point sampling strategies or feature fitting algorithms, the evaluating system may use any of those strategies or algorithms.

Since a given feature F may be associated with zero to many characteristics C1 ... Cn, if there is any characteristic associated with F, the rules should be evaluated for each FCi pair, and the number or density of hit points to use for F should be set to the maximum value.

If a number of hit points must be compared with a density of hit points, the evaluating system should find and use the area of the feature to convert density values to numbers of points.

Numerical quantities that are constants, feature parameters, or characteristic parameters may be used in conditions.

The numerical quantities, **FeatureSize**, **FeatureLength** and **FeatureArea**, may also be used in conditions. The evaluating system must set the value of **FeatureLength** for the feature being processed if **FeatureLength** is used in any condition. The evaluating system must set the value of **FeatureArea** for the feature being processed if **FeatureArea** is used in any condition.

The rules are not intended for picking the quantity of points or the point sampling strategy to use for measuring a characteristic that is not associated with any feature.

10.4.2.1 IfThenElseFeatureRulesType

The ***IfThenElseFeatureRulesType*** is a set of rules for selecting (1) the number or density of hit points for a single feature, possibly with an associated characteristic and/or (2) a strategy for selecting the points. To evaluate an ***IfThenElseFeatureRulesType***, each **IfThenFeatureRule** should be considered in order until the 'if' part of one of them evaluates to true, at which point the 'then' part of the rule should be evaluated and the value of the point quantity and/or point sampling strategy returned. In this case, the remainder of the **IfThenFeatureRule**s should be ignored. If the 'if' part of no **IfThenFeatureRule** evaluates to true, then the 'then' part of the Else should be evaluated and returned.

Below is an example of an ***IfThenElseFeatureRulesType*** rule.

```
<IfThenElseFeatureRules n="3">
  <IfThenCylinderRule>
    <ThenPoints>
      <NumberOfPoints>23</NumberOfPoints>
    </ThenPoints>
  </IfThenCylinderRule>
  <IfThenCircularArcRule>
    <ThenPoints>
      <PointDensity>0.8</PointDensity>
    </ThenPoints>
  </IfThenCircularArcRule>
  <Else>
    <ThenPoints>
      <MinPoints>13</MinPoints>
    </ThenPoints>
  </Else>
</IfThenElseFeatureRules>
```

In this example, we first test if the Feature is a cylinder. If so, then we request 23 points and have completed the ***IfThenElseFeatureRulesType*** rule. Otherwise, we test if the Feature is an arc. If it is, then we request a point density of 0.8 (with the exact placement of points to be calculated by the evaluating system). If the Feature was a circular arc, then we are finished evaluating. Otherwise, we fall into the “Else” statement: a minimum of 13 points.

10.4.2.2 MaxFeatureRulesType

The ***MaxFeatureRulesType*** is a set of rules for selecting (1) the number or density of hit points for a single feature, possibly with an associated characteristic and/or (2) a point sampling strategy for the points. To evaluate a ***MaxFeatureRulesType***, keep track of a current answer for point quantity, which is initially set to zero. Each **IfThenFeatureRule** should be considered in order. If the 'if' part of a rule evaluates to true, the 'then' part of the rule should be evaluated and if that value is greater than the current answer, the current answer should be set to that value. If the 'if' part of no **IfThenFeatureRule** evaluates to true, so that the current answer is still zero after all **IfThenFeatureRule**s have been processed, then the 'then' part of the Else should be evaluated and the current answer set to that value. The returned value for point quantity is the final value of the current answer. In addition to keeping track of the current answer for point quantity, the evaluating system should collect all the strategies and feature fitting algorithms from the 'then' parts that were evaluated.

Below is an example of the ***MaxFeatureRulesType*** rule.

```
<MaxFeatureRules n=”3”>
  <IfThenSurfaceRule name=”FirstRule”>
    <GreaterThan>
      <FeatureArea/>
      <ArithmeticConstant val="2"/>
    </GreaterThan>
    <ThenPoints>
      <MinPoints>4</MinPoints>
    </ThenPoints>
  </IfThenSurfaceRule>

  <IfThenSurfaceRule name=”SecondRule”>
    <And n=”2”>
      <CharacteristicIs val="SURFACEPROFILE"/>
      <LessThan>
        <ArithmeticCharacteristicParameter>
          <CharacteristicTypeEnum>SURFACEPROFILE</CharacteristicTypeEnum>
          <Parameter>ToleranceValue</Parameter>
        </ArithmeticCharacteristicParameter>
        <ArithmeticConstant val="0.010"/>
      </LessThan>
    </And>
    <ThenPoints>
      <MinPointDensity>0.2</MinPointDensity>
    </ThenPoints>
  </IfThenSurfaceRule>
  <Else>
    <ThenPoints>
      <MinPoints>3</MinPoints>
    </ThenPoints>
  </Else>
</MaxFeatureRules>
```

In this example, we must evaluate all rules to see which evaluate to true. There are 2 rules in this example: the **IfThenSurfaceRule** named “`FirstRule`” and the **IfThenSurfaceRule** named “`SecondRule`”. There is also an “Else” statement, to be executed only in the event that no rules apply to the Feature/Characteristic being evaluated.

In many cases, only the “`FirstRule`” or “`SecondRule`” will evaluate to true. In this case, the corresponding **ThenPoints** statement should be executed.

However, if the Feature/Characteristic happens to be on a surface of greater than 2 units of area AND is a surface profile tolerance with a tolerance value of less than 0.010 units, then both of the rules evaluate true. If this is the case, then the evaluating system must determine which of the rules would apply more points to the surface, and choose that rule.

Additional examples of feature rules are provided in the Rules section of the informative sampleInstanceFiles distributed with the QIF XML schema files.

10.4.3 DME Selection Rules

The **DMESelectionRules** *element* shown in Figure 228 is of type ***DMESelectionRulesType***. That type has a set of **DMEDecisionRule** *element*s, a required *attribute*, **n**, giving the number of rules, and an optional *attribute* **defaultDesirability**. The **defaultDesirability** is used as the value of desirability if no explicit value is provided in a rule.

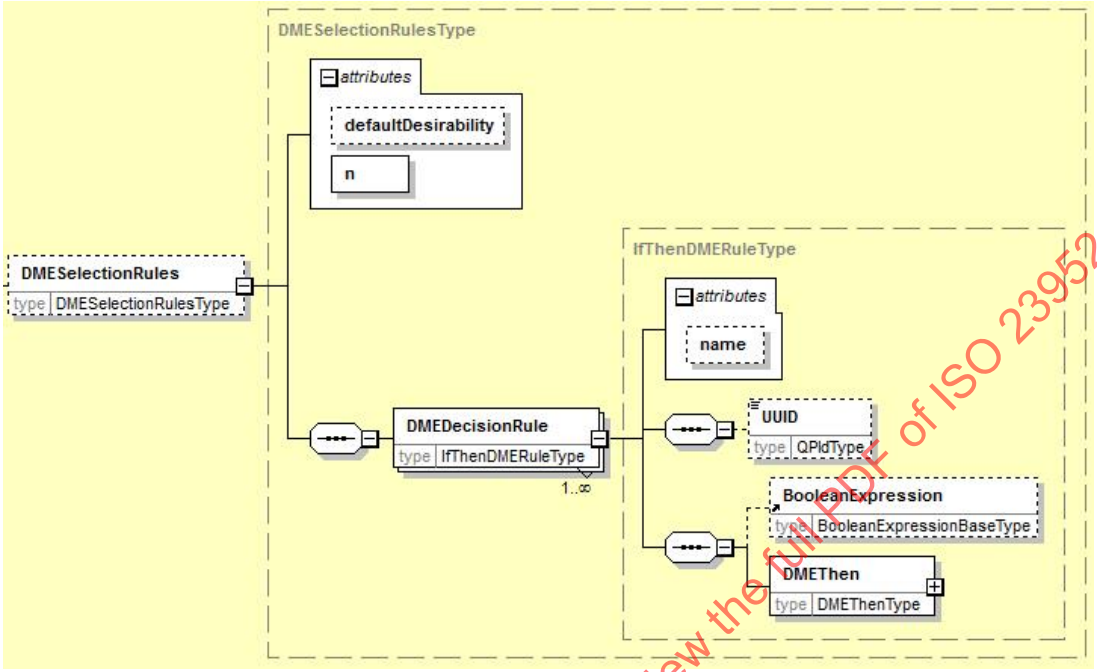


Figure 228 – **DMESelectionRules** *element* and ***DMESelectionRulesType***

As shown in Figure 228, each **DMEDecisionRule** *element* is of type ***IfThenDMERuleType***, which consists of:

- an optional *attribute* that is a **name** for the rule
- an optional **UUID** *element* that is a universally unique identifier for the rule
- an optional **BooleanExpression** *element* and
- a required **DMEThen** *element*.

When rules are being evaluated, the **DMEThen** part of the rule applies if either the **BooleanExpression** evaluates to true or is omitted.The types of Boolean expression supported by the model are described in subclause 10.2.1.

10.4.3.1 DMEThen

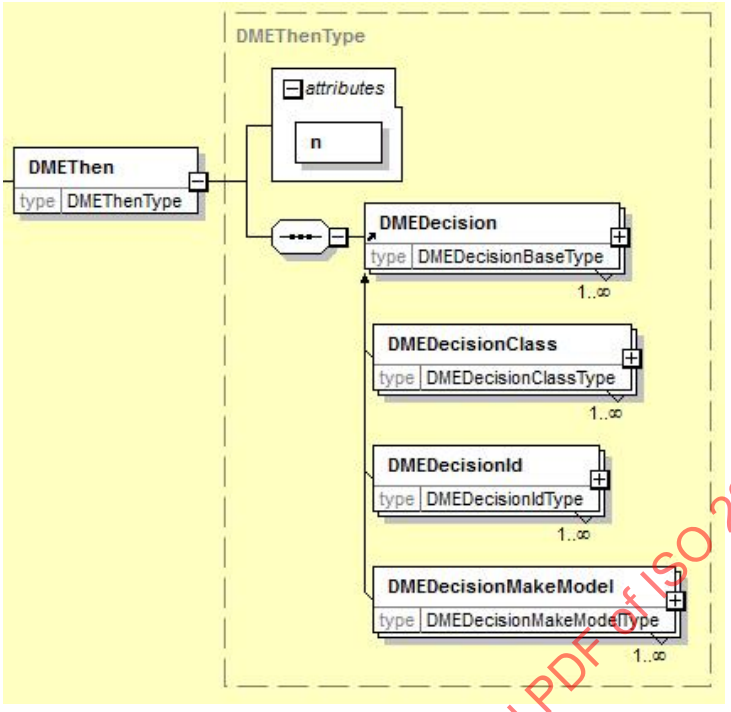


Figure 229 – **DMEThen** *element* and ***DMEThenType***

Figure 229 shows the **DMEThen** *element* and the ***DMEThenType***. The *element* is the “Then” part of a rule and is of type ***DMEThenType***. The type names one *element*, **DMEDecision**, but that *element* is of an *abstract* type, so one of the three *elements*, **DMEDecisionClass**, **DMEDecisionId**, or **DMEDecisionMakeModel** from its substitution group must be used in its place. Note also that multiple *elements* may be used, as given by *attribute* **n**. That enables a single **DMEThen** to specify several DMEs, such as a particular combination of CMM, tool, and probe tip.

10.4.3.2 DMEDecisionClass

The **DMEDecisionClass** *element* and ***DMEDecisionClassType*** are shown in Figure 230.

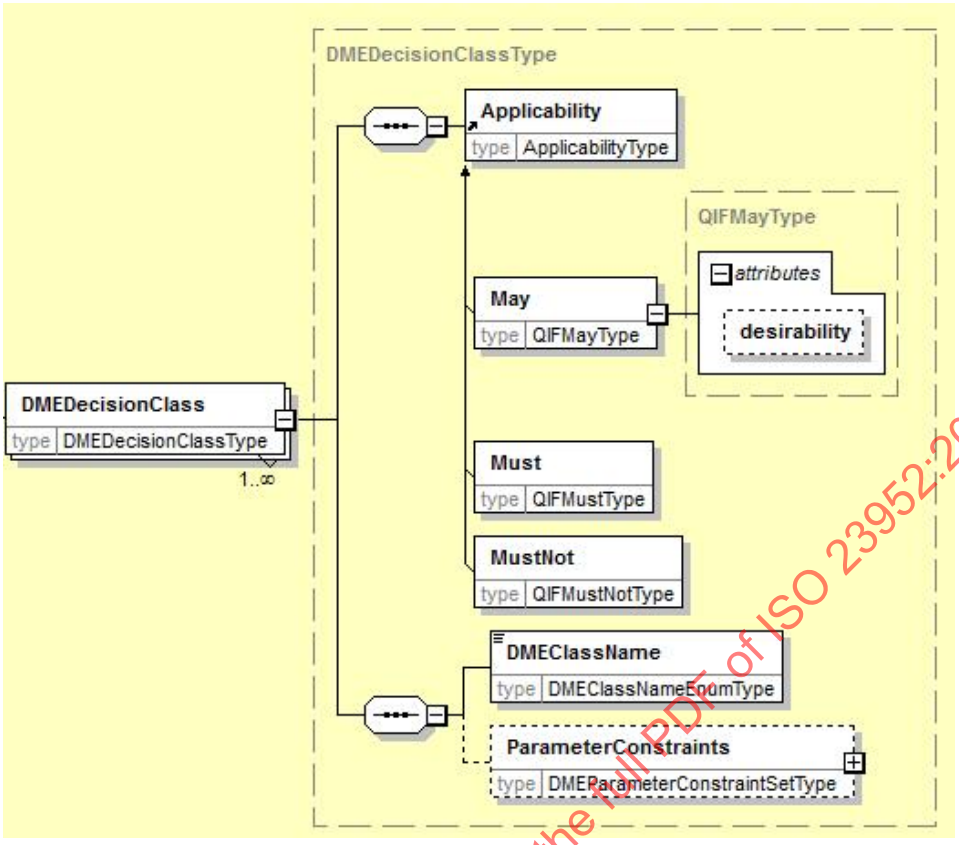


Figure 230 – **DMEDecisionClass** *element* and ***DMEDecisionClassType***

Using a **DMEDecisionClass** *element* indicates that any member of the DME class whose name is given in the **DMEClassName** *element* may be selected or ruled out, provided it meets any constraints given in the optional **ParameterConstraints** *element*. If the **Applicability** is **May** or **Must**, the planning system that is using the rules is expected to select one or more DMEs that are in the class and satisfy the constraints.

The *abstract* ***DMEDecisionBaseType*** (not shown in the figure) has only one *element*, an **Applicability** which is of the *abstract* ***ApplicabilityType***. In an instance file the **Applicability** *element* must be replaced by an *element* in its substitution group, namely one of:

- a **May** *element* of ***QIFMayType***
- a **Must** *element* of ***QIFMustType***
- a **MustNot** *element* of ***QIFMustNotType***

“Must” means that a member of the class must be selected.

“MustNot” means that a member of the class must not be selected.

“May” means that a member of the class may be selected, but does not require that a member of the class be selected.

The ***QIFMustType*** and the ***QIFMustNotType*** do not have any *element*s or *attribute*s, but the ***QIFMayType*** has an optional **desirability** *attribute* of ***ZeroToOneType*** (a double value in range [0..1]). If no value for the **desirability** *attribute* is provided, the value is the **defaultDesirablity** given in the ***DMESelectionRulesType***, if that is provided, or 1 if not. The **desirability** value (which is set by the author of the rules) is intended to help compare choices during DME selection. The way in which the **desirability** value is to be used is not specified by QIF, but is left to the planner and the

planner's organization. It might, for example, be advisory (it would be a good idea to select a more desirable DME), prescriptive (you must select the most desirable choice), or ignored.

The **DMEClassName** *element* of the ***DMEDecisionClassType*** is the name of a DME class. Its value is one of the names given in the ***DMEClassNameEnumType***.

The optional **ParameterConstraints** *element* of the ***DMEDecisionClassType*** is a set of constraints on parameters of the class. A member of the specified class must satisfy the parameter constraints in order to be included in set of identified DMEs. Each constraint is of type ***DMEParameterConstraintType***.

The ***DMEParameterConstraintType*** has **ParameterName**, **Comparison**, and **ArithmeticExpression** *element*s. For a DME to satisfy a constraint, the named parameter of the DME must satisfy the enumerated comparison with the evaluated arithmetic expression. The parameter name must be the name of a parameter of the class of DMEs named in the **DMEDecisionClass** in which the constraint is found. For example if the the DME class name is CALIPER, the parameter is LinearResolution, the comparison is LESS, and the arithmetic expression is 0.01, then the resolution of a caliper being considered must be less than (i.e., better than) 0.01.

It will often be possible to represent a constraint on a DME either as a rule or as a parameter constraint. Using parameter constraints can help avoid a proliferation of rules.

Here is an example of a DME selection rule using the **May** *element*. It is from the Rolls-Royce "Guide" mentioned earlier, page 25, table 5. The English language version of the rule is: "If feature size is less than 5 mm and tolerance is greater than or equal to 0.100 mm, then an analog micrometer may be used". The rule modeled below adds the requirement that the feature being considered must not be an internal feature.

```
<DMEDecisionRule>
  <And n="3">
    <Not><FeatureIsInternal/></Not>
    <LessThan>
      <FeatureSize/>
      <ArithmeticConstant val="5.0"/>
    </LessThan>
    <GreaterOrEqual>
      <ArithmeticCharacteristicParameter>
        <Parameter>Tolerance</Parameter>
      </ArithmeticCharacteristicParameter>
      <ArithmeticConstant val="0.100"/>
    </GreaterOrEqual>
  </And>
  <DMEThen n="1">
    <DMEDecisionClass>
      <May desirability="1"/>
      <DMEClassName>ANALOG_MICROMETER</DMEClassName>
    </DMEDecisionClass>
  </DMEThen>
</DMEDecisionRule>
```


10.4.3.3 DMEDecisionId

The **DMEDecisionId** *element* and ***DMEDecisionIdType*** are shown in Figure 231.

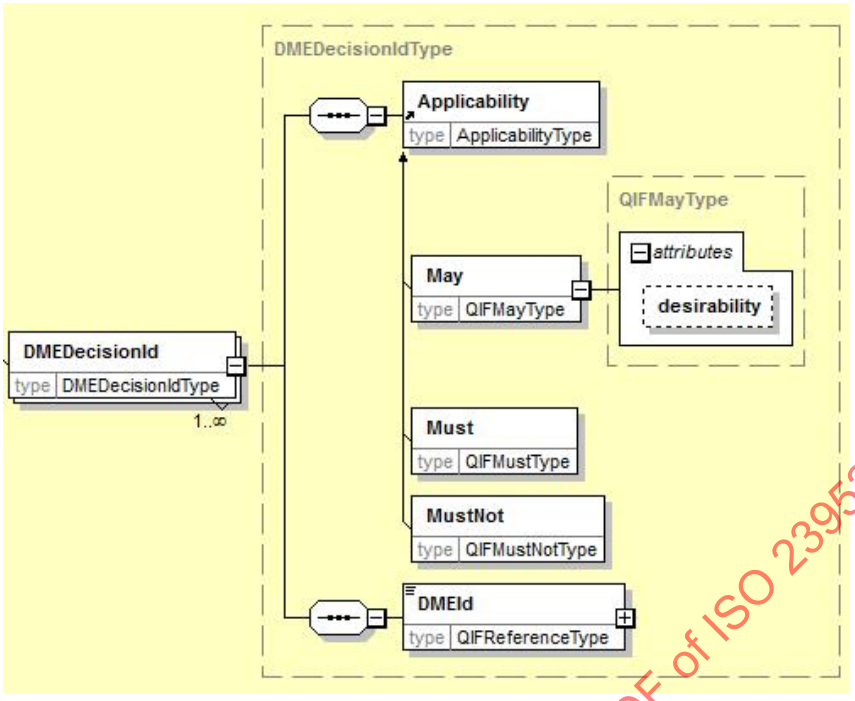


Figure 231 – **DMEDecisionId** *element* and ***DMEDecisionIdType***

The **DMEDecisionId** *element* indicates a specific DME with a QIF Id. The value of the **DMEId** *element* must be the QIF id of a DME found in the **MeasurementResources** section of a **QIFDocument**. The **QIFDocument** with the measurement resource could be in the same file as the one containing the rules, but is expected more usually to be in a separate file.

The meanings of the *element*s that may be substituted for **Applicability** are the same as for the ***DMEDecisionClassType*** described in subclause 10.4.3.2.

10.4.3.4 DMEDecisionMakeModel

The **DMEDecisionMakeModel** *element* is of type ***DMEDecisionMakeModelType*** and is shown in Figure 232. This *element* indicates a DME identified by its make and model. Optionally, a serial number may be given.

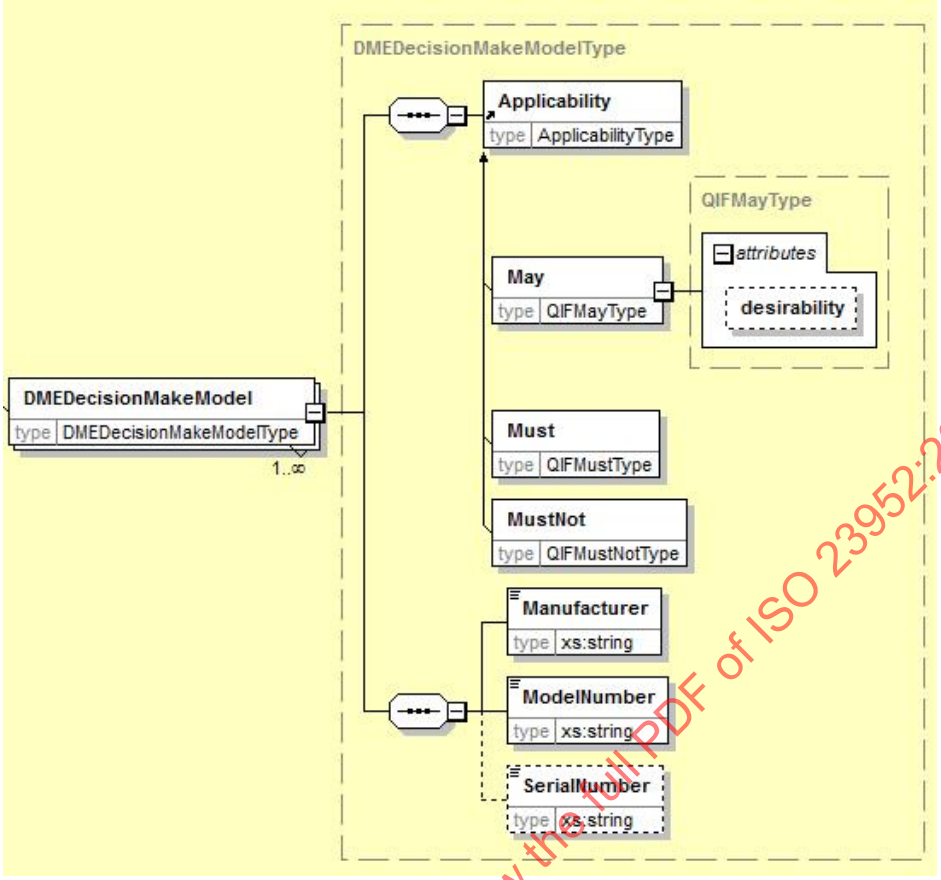


Figure 232 – **DMEDecisionMakeModel** *element*

The ***DMEDecisionMakeModelType*** has required **Manufacturer** and **ModelNumber** *element*s and an optional **SerialNumber** *element* (so that a specific instance of a DME may be chosen even if it is not represented in a QIF resources file).

The meanings of the *element*s that may be substituted for **Applicability** are the same as for the ***DMEDecisionClassType*** described in subclause 10.4.3.2.

11 QIF Results information model

11.1 Foreword

The Quality Information Framework (QIF) information model was developed by domain experts from the manufacturing quality (that is metrology) community representing a wide variety of industries and quality measurement needs. Specifically for the QIF Results work, past and current contributors include:

- Capvidia, Inc.
- Honeywell Federal Manufacturing & Technologies
- Lockheed Martin
- Manufacturing Technology Centre
- Metrosage
- Mitutoyo America
- National Institute of Standards and Technology
- Origin International Inc.
- University of North Carolina, Charlotte

The bulk of the work on this clause was performed by the QIF Results Working Group, revised as needed and approved by the Quality Information Framework (QIF) Working Group, and given final approval for ANSI balloting by the DMSC's Quality Measurement Standards (QMS) Committee. QIF version 3.0 is solely a product of the DMSC and its committees and working groups.

11.2 Introduction

Part inspection is carried out by a measurement execution activity, whose functionality is typically to interpret machine-level measurement plans, give equipment level commands to specific dimensional measuring equipment (DME) control units, collect point data, fit features to the data, and output feature and characteristic data. Measurement execution can also include software solutions that issue instructions to human operators using calipers, go/no-go gages, and specialized inspection equipment to generate results data. Once the machine-level measurement plans are executed, the measurement data, either raw data or pre-processed data, is collected, reported, and analyzed. The QIF Results specification provides a vendor-neutral format for the data. Software solutions that support QIF, by either writing or reading QIF files, can exchange manufacturing quality data efficiently and accurately.

11.3 Scope

11.3.1 Workflow of QIF Results data for manufacturing quality

Clause 5 describes an example of enterprise quality information workflow that employs QIF modeled information at all interfaces. Figure 233 shows an example of quality workflow that uses only QIF Results data, in order to focus on the QIF Results information model.

In this example, upstream processes generate a set of part-specific inspection requirements. A common example is the set of characteristics and requirements data of a First Article Inspection Report (FAIR) conforming to the AS9102B: *Aerospace First Article Inspection Requirement* standard. The Program the Measurement activity takes knowledge of actual inspection resources by

capability and availability, assigns resources to the plan, and generates a program or set of inspection instructions suitable to the resource chosen. The execution program may be in ANSI/DMIS 105.3-2016 format or any other proprietary format, including non-digital information like instructions to human operators using calipers, go/no-go gauges, or other specialized inspection equipment. The QIF information model has been harmonized with ANSI/DMIS 105.3-2016 to support all data that may be defined in a DMIS program.

The Execute Measurement Program activity shown in Figure 233, inspects the part or assembly by executing the program and generates results formatted according to the QIF Results component of QIF. Finally, measurement results are collected, reported, and analyzed, through the Analyze and Report Quality Data activity. Results of a single part inspection, as well as the statistical analysis of groups of parts, can provide feedback to upstream processes such as product manufacturing and quality engineering.

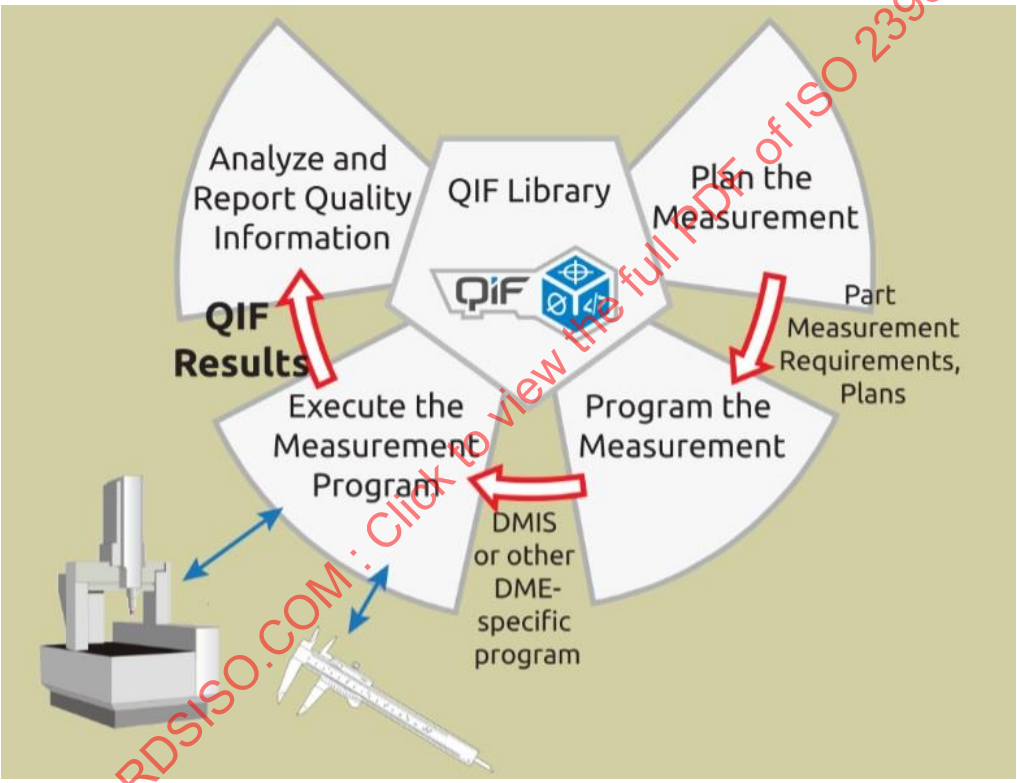


Figure 233 – Example of QIF Results information flow.

11.3.2 Design guidelines for the QIF Results information model

The QIF Results information model is designed to satisfy the following functional requirements:

- express results of dimensional inspection according to the principles of ASME GD&T and ISO GPS, and support data required to fully express the intent and results of inspection programs written in the DMIS language.

- support inspection traceability of quality data, meaning that inspection results data can be identified with the factory resources used to inspect the part, including equipment, software, and people.

- support information needed by downstream processes, e.g., First Article Inspection Report (FAIR) generation, statistical process control, and other computer aided quality solutions.
- support a quality data workflow that is digital model-based or document/drawing-based.

11.4 The QIFResults.xsd schema file

11.4.1 High level structure of the QIF Results schema

This subclause describes the highest level *elements* of the QIF Results information model. The QIF Results schema model includes the information items from the QIFResults.xsd schema file and several of the schema files in the QIF Library. The QIF Library files are incorporated into the schema by a chain of "*include*" directives starting in the QIFResults.xsd schema file.

The **Results** *element,* shown in Figure 234, which is of type ***ResultsType***, is the highest level *element* of the QIF Results model. *Elements* in the figure with + signs at the right (all of them) have substructure that is not shown in the figure. Most substructure is defined in the QIF Library files.

An XML instance file written in conformance with the QIFDocument.xsd and QIFResults.xsd schema files may contain one **Results** *element*.

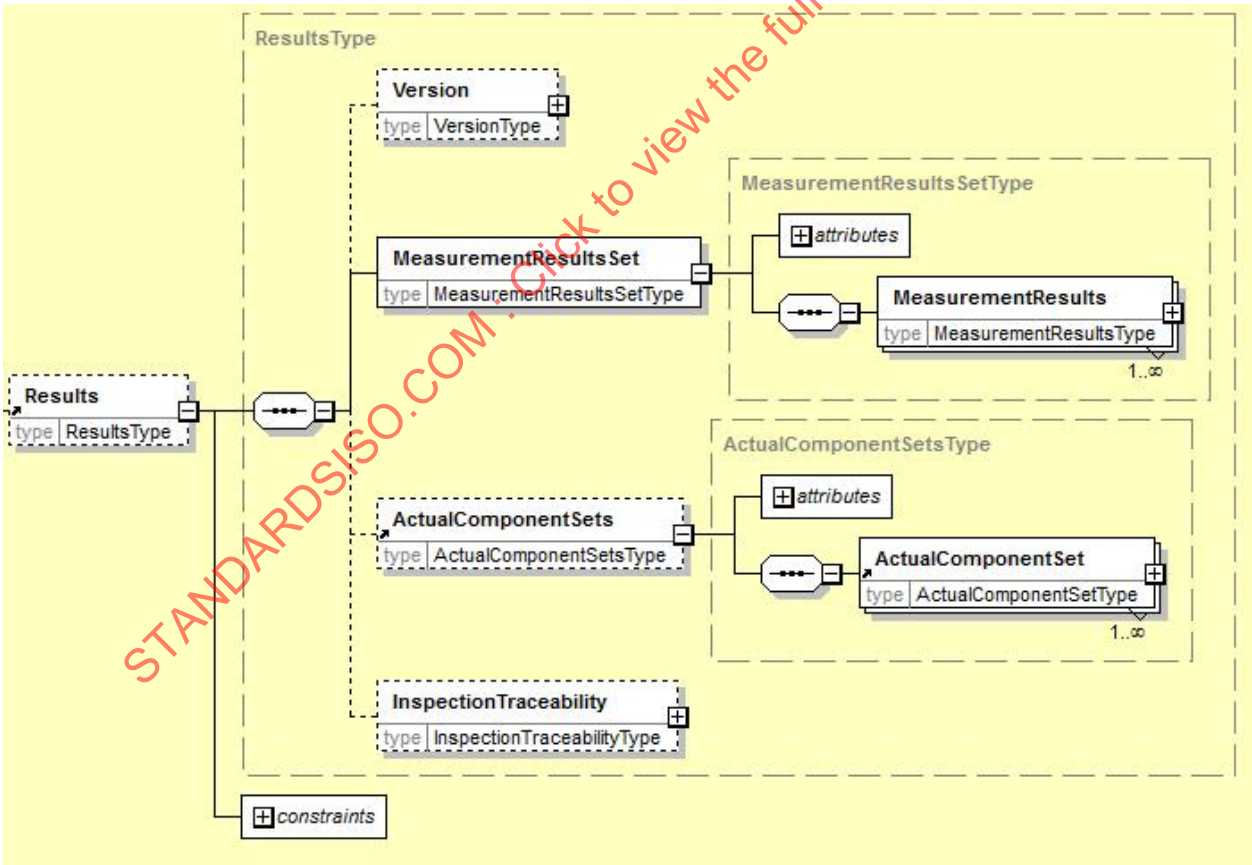


Figure 234 – The **Results** *element*

The *element*s of **Results** are:

