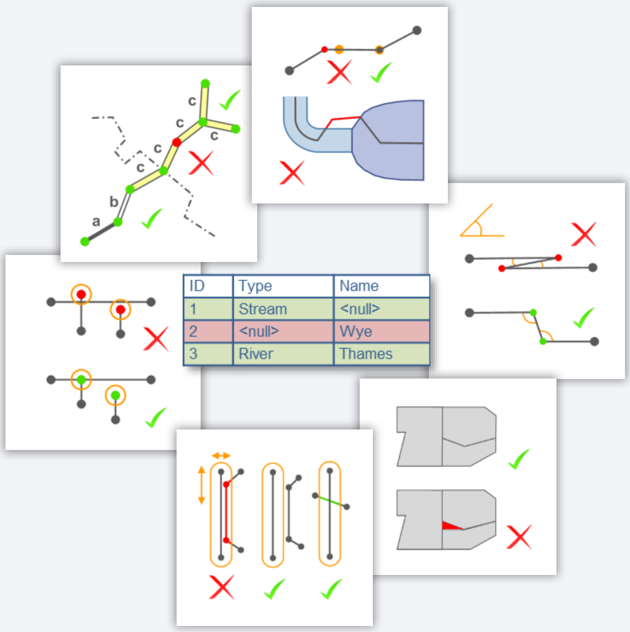
ProSuite QA

Test Quick Reference

Quality conditions

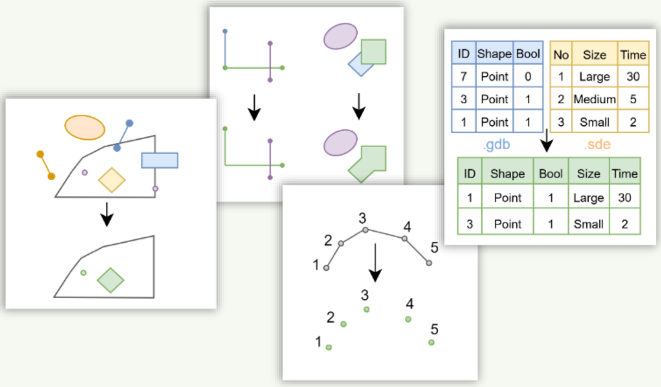
- Attributes and relationships
- Single feature geometries
- Linear networks
- Proximity (distance between features)
- Topological conditions
- Edge matching
- M values
- Z values
- 3D buildings (MultiPatch features)
- Polygon networks (boundary lines and centroids)
- Table/feature class schemas
- No Category



ID	Type	Name
1	Stream	<null>
2	<null>	Wye
3	River	Thames

Dataset Transformers

- Geometry
- Table
- Z-Value
- Input Feature Filter



ID	Shape	Bool
7	Point	0
3	Point	1
1	Point	1

No	Size	Time
1	Large	30
2	Medium	5
3	Small	2

ID	Shape	Bool	Size	Time
1	Point	1	Large	30
3	Point	1	Small	2

Issue Filters

Attributes

Conditions for attributes and relationships

Field values match domains and subtypes

Finds rows in tables and feature classes with field values that violate the attribute rules as defined in the geodatabase (domains, subtypes).

Test: *QaGdbConstraint, QaGdbConstraintFactory*

Domain for Field [Type]

Code	Description
100	Path
200	4m Road

Checked Table

	ID	Type
x	1	300
	2	100

Freely defined attribute conditions

Finds rows in tables and feature classes that do not fulfil the defined attribute conditions.

The conditions can be organized in a hierarchical structure. Parent conditions successively restrict the set of rows to be tested. Conditions defined at the lowest hierarchy level are then verified for the resulting set of rows.

Attribute conditions for joined tables can also be tested.

Alternatively, the attribute conditions can be defined in a separate database table.

Tests: *QaConstraint, QaDatasetConstraintFactory, QaRelConstraint, QaConstraintsListFactory*

```
Objecttype=Path  (Selection)
+Width IS NULL OR
 Width < 4  (Condition)

Objecttype=Motorway  (Selection)
+Width > 10  (Condition)
+Surface <> ‚Loose‘ (Condition)
```

	ID	Objecttype	Width	Surface
x	1	Path	6m	Loose
	2	Path	3m	Loose
	3	Motorway	12m	Hard
x	4	Motorway	12m	Loose

No NULL values in mandatory fields

Finds rows in tables and feature classes, that do not contain non-NULL values in definable mandatory fields.

Test: *QaRequiredFields*

[Type] is a mandatory field.
[Name] is not a mandatory field.

	ID	Type	Name
	1	Stream	<null>
x	2	<null>	Wye
	3	River	Thames

Referential integrity

Finds rows in tables and feature classes containing foreign keys, which do not refer to a key value in the reference table.

This test also supports key values which are composed of several fields and tables from different databases. Furthermore it is possible to report unwanted *existing* references between subsets of table rows as errors.

Test: *QaForeignKey*

Reference Table

ID	Country
1	Switzerland
2	United Kingdom

Checked Table

	ID	City	Country FK
x	1	Lisbon	5
	2	London	2
	3	Berne	1

No unreferenced rows

Finds rows in tables and feature classes which are not referred to by any row in one or multiple referencing tables.

The tables may exist in different databases. The following relationship types are supported: n:1, 1:1 and n:m.

Test: *QaUnreferencedRows*

Surveying Lines

ID	Contract-ID
1	10
2	10
3	11

Surveying Points

ID	Contract-ID
1	11
2	10

Referenced Contracts

	ID	Number	Year
	10	AB135324	2012
	11	XY453465	2009
x	12	PP512324	2010

Unique field values

Finds rows in tables and feature classes, which contain non-unique values in a field or a list of fields.

This test can be applied to multiple fields from joined tables.

Tests: *QaUnique, QaRelUnique*

The contents of the fields [Region] and [Name] must be unique in combination.

	ID	Region	Name
	1	A	X
x	2	A	Y
x	3	A	Y
	4	B	Z
	5	B	X

Regular expressions in text fields

Finds rows in tables and feature classes, whose value(s) from a single or from multiple text fields do not match a definable regular expression (regex).

Use this test to dictate arbitrary format rules to text values (e.g. structured keys, email addresses, telephone numbers, exclusion of special characters).

Test: *QaRegularExpression, QaRelRegularExpression*

Starts with no spaces or tabs
Expression: `^[ \t]+`

	ID	Description
x	1	••Example text
x	2	→ Example text
	3	Example text

Number of distinct values within a group of features

Checks if the number of distinct values of an expression within a group of rows/features lies within a minimum and a maximum value.

This test can be applied to fields from joined tables and the groups can be evaluated over multiple tables or feature classes, with individual “group by” and value criterions. Simple field names as well as complex SQL functions can be used as “group by” and value criterions.

Examples:

- In a denormalized table schema, check if per value of a field a unique value in another, depending field exists (see right).
- Check across multiple tables and feature classes if a group of rows in one table corresponds to only one single row in the other table.

Tests: *QaGroupConstraints, QaRelGroupConstraints*

The information regarding river courses lies (denormalized) on the individual river lines: there must be a unique name per river ID.

	ID	River.Course	River.Name
x	1	1	Thames
x	2	1	Wye
	3	2	Rhine
	4	2	Rhine

No leading/trailing spaces in text fields

Finds values in selected text fields, which contain one or more spaces at the beginning or the end.

Background: Spaces make table queries difficult. They are hard to find, especially when at the end of a text.

Test: *QaTrimmedTextFields*

	ID	Description
x	1	␣Description 1
x	2	Description 2␣
	3	Description 3

No empty strings in mandatory Text Fields

Checks if mandatory text fields (i.e., fields declared as NOT NULL) contain empty strings.

Background: Oracle converts empty strings automatically to a NULL value. Therefore, data imports from other databases, which contain empty strings in a NOT NULL fields, can fail (due to violation of the NOT NULL condition in Oracle). This test ensures that data from other databases can be imported into Oracle.

Test: *QaEmptyNotNullTextFields*

	ID	Description [NOT NULL]
x	1	""
	2	Description

Date field values without time of day

Finds rows in tables and feature classes which contain dates with a specified time of day.

Background: A specified time of day in a date field makes queries difficult because it requires either a range query (>=, <=, BETWEEN) or a SQL function to transform the date value. In some cases, a time of day is undesirable or unnecessary.

Test: *QaDateFieldsWithoutTime*

	ID	Date
x	1	2001-09-13-05-00-30
	2	2001-09-13-00-00-00

Valid date values

Finds rows in tables and feature classes which contain date values which are not within a specified range. Optionally, the time range can be specified relative to the current date.

Background: Apart from finding date values that are invalid based on the specific meaning of a field, this test can also be used for date ranges that are technically limited. For example, SQL Server only supports dates after 1.1.1753 in DateTime fields. So, with this test, other databases can be checked if a future export to SQL Server/Express might fail due to this date constriction.

Test: *QaValidDateValues*

```
minimumDateValue:
  01-01-1990
maximumDateValue:
  31-12-2019
```

	ID	Date
x	1	1850-09-13
x	2	2020-09-13
	3	2012-09-13

Attributes

Conditions for attributes and relationships

Valid URLs

Checks if attribute values in a table or feature class contain valid URLs. They're considered as invalid if the resource is not available (e.g. non existing file, server not accessible etc.). Currently, http URLs or file system references (UNC paths or connected network drives) are supported.

The URL can reside in a single field or be the result of a SQL expression, which refers to one or more attributes and can be used to concatenate URLs including constant parts such as a prefix or suffix.

Test: *QaValidUrls*

ID	URL
1	http://invalid.url.html
2	http://valid.url.html
3	\\server\directory\file.txt
4	file:///T:/directory/file.txt
5	

Valid data types

Finds rows in tables and feature classes which are not compatible with the data type of the field, e.g. text in a GUID field.

Test: *QaValue*

	ID	GUID
x	1	{invalid_guid}
	2	{1AF99E7F-255F-40EA-B711-ECABE73B74B6}

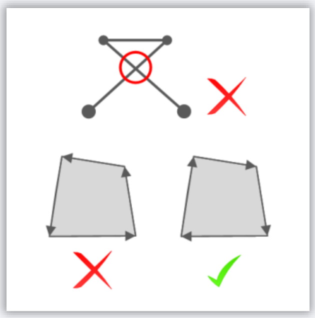
Matching values in coordinate fields

Finds point features with X, Y and/or Z attributes whose values differ by more than a specified tolerance from the coordinates of the feature.

Test: *QaValidCoordinateFields*

	ID	Shape	X	Y
	1	Point(100,100)	100	100
	2	Point(100,100.2)	100	100
x	3	Point(100,100)	200	200

Simple geometry



Finds line, polygon, multipoint or multipatch features whose geometry is not “simple”. The following properties define “simple” Geometry:

- There are no segments shorter than the XY tolerance of the feature class.
- Multi-points: No pairs of points have a distance smaller than the XY tolerance of the feature class
- No self-intersections
  - For line features self-intersections can be optionally allowed.
- Polygons: Rings have correct orientation
  - Outer rings: Clockwise direction
  - Inner rings (“holes”): Counter clockwise
- Polygons: Rings must be closed
- Polygons/polylines with multipart geometry (rings or paths): No empty geometry parts
- Polygons/polylines: All segments of a ring or path have an identical direction

Non-simple geometries can provoke geometric operations to fail with an error message. Therefore, this test should be used early on when integrating external data.

Test: *QaSimpleGeometry*

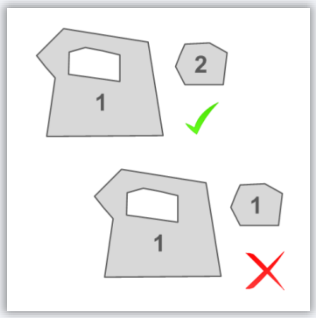
No features with empty geometry

Finds features with an empty geometry.

	ID	SHAPE
x	0	<null>
	1	Polygon

Test: *QaNonEmptyGeometry*

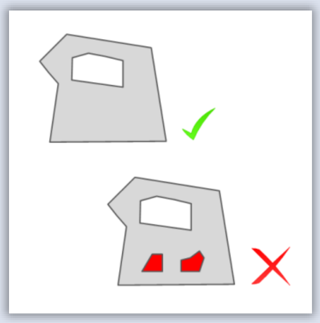
No multipart features



Finds features composed of multiple geometry parts (multipart features). In the case of polygons, only multiple *outer* rings are normally reported as errors. Optionally, inner rings (“holes”) can also be reported.

Test: *QaMultipart*

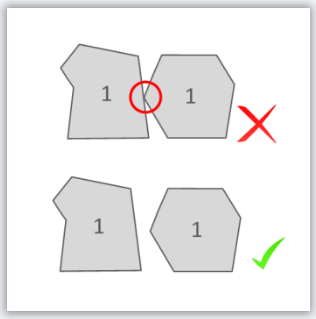
Maximum number of holes in polygons



Finds polygon features with more inner rings („holes“) than defined in a maximum value.

Test: *QaInteriorRings*

No touching geometry parts



Finds where geometry parts touch in a multipart polygon or polyline.

Test: *QaNoTouchingParts*

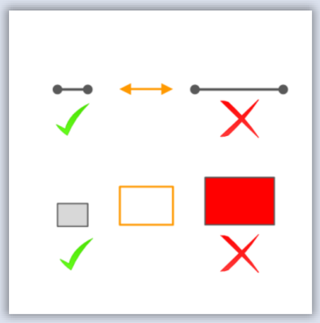
Geometry

Conditions for single feature geometries

Minimum and maximum dimensions

Finds line or polygon features whose length or area falls below or exceeds a definable value. Optionally, the geometry parts of multipart features can be tested individually.

Tests: *QaMaxArea, QaMinArea, QaMaxLength, QaMinLength*



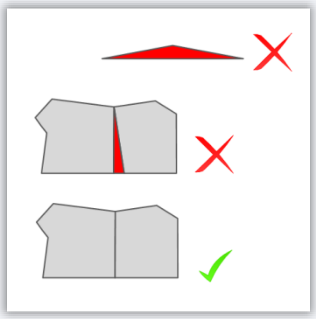
No sliver polygons

Finds elongated polygons („slivers“). Polygons are regarded as „slivers“ if the circumference raised to the second power and divided by the area exceeds a definable value. The following list shows the values (circumference ^{2} / area) for rectangles of different aspect ratios:

- Aspect ratio 1:1 (square): 16
- Aspect ratio 1:5: 28,8
- Aspect ratio 1:10: 48,4
- Aspect ratio 1:20: 88,2

Additionally, a maximum area value may be given. Polygons larger than this value are never considered as „slivers“.

Test: *QaSliverPolygon*

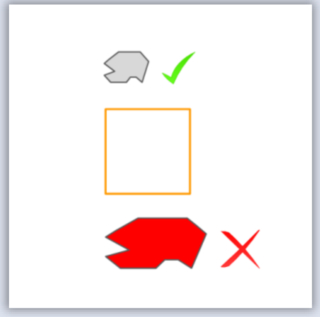


Maximum extent of a features

Finds features whose maximum extent in the X or Y direction exceeds a definable maximum value.

Optionally, the individual geometry parts of multipart features can be tested separately.

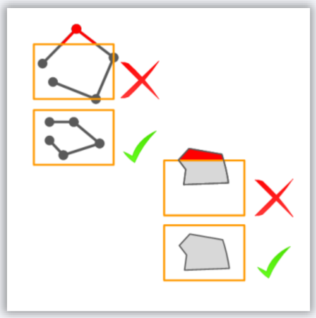
Tests: *QaExtent*



Geometry within a box

Finds features which do not lie completely within a specified box. The boundaries of the box are defined with a minimum and maximum X and Y value. For features that lie partially outside the box, the entire feature or only the outside part can be reported as an error.

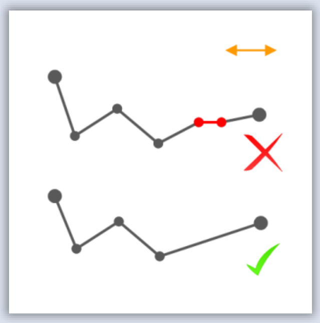
Test: *QaWithinBox*



Minimum segment length

Finds line or polygon segments that are shorter than a definable minimum length. For features with Z values, either the 2D or 3D length can be tested.

Test: *QaSegmentLength*



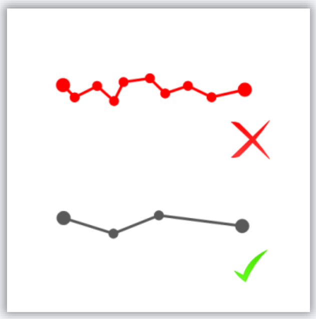
Maximum number of vertices

Finds line, polygon, multipoint and multipatch features with more vertices than a definable maximum value.

Optionally, the individual geometry parts of multipart features can be tested separately.

Background: Geometries with more than tens to hundreds of thousands vertices can significantly increase the computing cost for certain geometry operations and cause slow response times. This test identifies such features. In certain cases, the number of vertices can then be reduced without the loss of essential information.

Test: *QaMaxVertexCount*



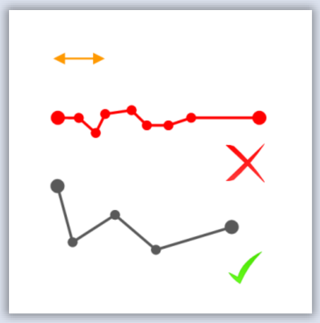
Minimum average segment length

Finds line and polygon features with a smaller than defined average segment length.

Optionally, the individual geometry parts of multipart features can be tested separately.

This test complements the *minimum segment length* and *maximum number of vertices* tests. It finds features with a high vertex density, even if the maximum number of vertices has not been exceeded or the minimum segment length has not been reached.

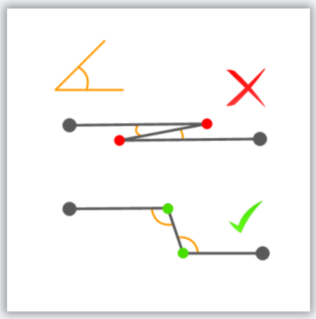
Tests: *QaMinMeanSegmentLength*



Minimum angle between consecutive segments

Finds consecutive segments in line and polygon features where the angle between the segments is smaller than a definable minimum value (e.g. „cutbacks“).

Tests: *QaMinSegAngle*



No closed loops in area boundaries

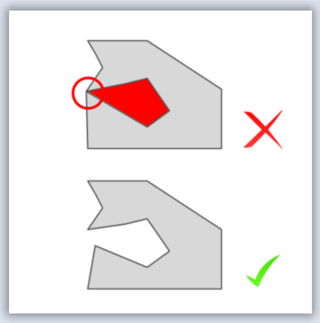
Finds closed inward loops in boundaries of polygons and multipatch rings (“boundary loops”).

Background: These loops comply to the rules of “simple” geometry, but can cause problems when further processed with other systems. To avoid such a loop, the loop can be opened at the point of contact or merged and widened to create a real inner ring (“hole”).

Alternatively, the entire polygon area or only the point of contact of the loop can be reported as error, the latter making it easier to locate the correction point in very large polygons.

Optionally, loops larger or smaller than a given area can be ignored.

Test: *QaNoBoundaryLoops*

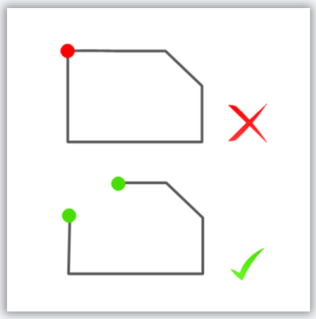


No closed lines

Finds closed paths in line features. These are paths, whose start and end points are coincident.

Background: geometric networks do not allow closed paths. With this test, affected features can be identified before building a geometric network or importing data into an existing geometric network.

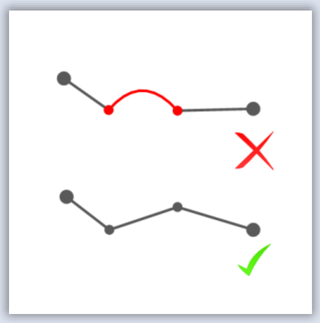
Test: *QaNoClosedPaths*



Geometry

Conditions for single feature geometries

No non-linear segments

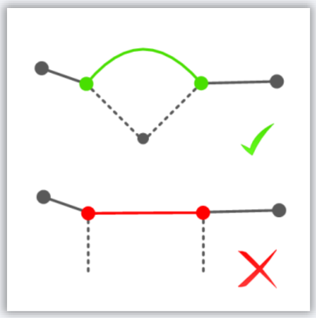


Finds non-linear line and polygon segments, i.e. circular or elliptic arcs and Bezier curves.

Background: Certain data formats cannot store non-linear segments. Therefore, in certain feature classes such segments must be avoided to allow the export to such a format without information loss.

Test: *QaCurve*

Valid circular arcs



Finds circular arc segments in line and polygon features which have been degenerated to straight lines and therefore are missing a centre point and have an invalid radius. Additionally, arcs can be identified whose chord height is smaller than a definable minimum value.

Background: This type of circular arcs may be produced upon export into a feature class with a bigger XY tolerance of the spatial reference system. The reason being that arcs which have a chord height that is smaller than the XY tolerance are transformed into straight arcs (with an infinite radius and without centre point). During certain forms of further processing these arcs can lead to problems.

With the possibility to check the minimum chord height before exporting, it can be tested whether in the target feature class degenerated arcs will be produced.

Test: *QaValidNonLinearSegments*

Constraints based on geometry properties

```
($ZMax - $ZMin) < 1000
```

	ID	Shape	*$ZMin*	*$ZMax*
	1	Polygon	*200*	*210*
	2	Polygon	*150*	*170*
x	3	Polygon	*200*	*1300*

Finds features that do not fulfill a defined constraint on properties of the feature geometry. The constraint can be based on the following geometry properties, and can optionally be applied per part (in case of multipart features).

- `$Area`: the area of the geometry
- `$Length`: the length of the geometry
- `$VertexCount`: the number of vertices of the geometry
- `$SliverRatio`: the ratio of perimeter^2 / area, expressing the shape elongation
- `$Dimension`: the geometry dimension (0, 1, 2)
- `$EllipticArcCount`: the number of elliptic arcs
- `$CircularArcCount`: the number of circular arcs
- `$BezierCount`: the number of Bezier segments
- `$LinearSegmentCount`: the number of linear segments
- `$NonLinearSegmentCount`: the number of non-linear segments (e.g. circular arcs)
- `$SegmentCount`: the total number of segments
- `$IsClosed`: indicates if the geometry is closed
- `$XMin`: the minimum X/easting coordinate
- `$YMin`: the minimum Y/northing coordinate
- `$XMax`: the maximum X/easting coordinate
- `$YMax`: the maximum Y/northing coordinate
- `$ZMin`: the minimum Z value
- `$ZMax`: the maximum Z value
- `$MMin`: the minimum M value
- `$MMax`: the maximum M value
- `$UndefinedMValueCount`: the number of vertices with undefined M values

For use on *entire* feature geometries:

- `$PartCount`: the number of geometry parts
- `$IsMultipart`: indicates if the geometry consists of multiple parts (rings or paths).
- `$ExteriorRingCount`: the number of exterior rings in the polygon or multipatch
- `$InteriorRingCount`: the number of interior rings in the polygon or multipatch

For use on geometry parts:

- `$IsExteriorRing`: indicates if the geometry part is an exterior ring
- `$IsInteriorRing`: indicates if the geometry part is an interior ring

For use on multipatch geometries:

- **`$RingCount`**: the number of rings in the multipatch
- **`$TriangleFanCount`**: the number of triangle fans in the multipatch
- **`$TriangleStripCount`**: the number of triangle strips in the multipatch
- **`$TrianglesPatchCount`**: the number of patches of triangles in the multipatch

These properties can be used in a WHERE-clause like expression. If the expression is *not* fulfilled for a given feature, then an issue is reported.

Examples:

```
$SliverRatio < 50 OR $Area > 10
$Area < 5 AND $VertexCount < 10
$Length > 10 OR $PartCount > 1
NOT ($IsClosed AND $EllipticArcCount = 1 AND $SegmentCount = 1)
```

Test: *QaGeometryConstraint*

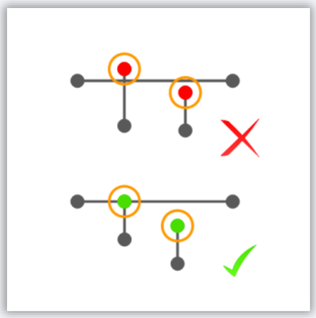
No over- and undershoots

Finds point features or line end points which are too close to other lines or polygon boundaries, but are not exactly coincident. This test allows locating line over- and undershoots.

Errors are not reported for (end) points which lie further away from another line or a polygon boundary than a definable distance. For feature classes with Z values, the distance can be measured in 2D or 3D. This allows applying the test to multi-level street crossings.

Coincidence of (end) points with another line or polygon boundary is determined based on the XY tolerance of the spatial reference system.

Test: *QaNodeLineCoincidence*



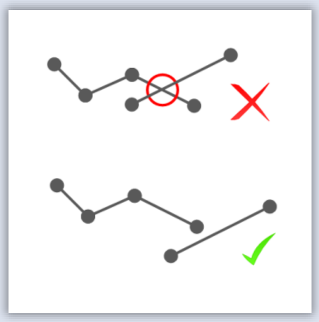
No invalid line intersections

Finds line features which cross or touch other lines illicitly. Punctual intersections as well as linear overlaps will be reported as errors.

For a line end point on the interior of another line, it can be specified if the intersection may lie on an arbitrary location along the line, if it is only allowed exactly on a vertex or if it is not allowed at all. In the last case, line end points may touch other lines only at their end points.

Exceptions can be defined with a SQL condition which compares the attribute values of the involved features at the intersection. If the condition is fulfilled for two intersecting features, then their intersection is allowed.

Test: *QaLineIntersect*



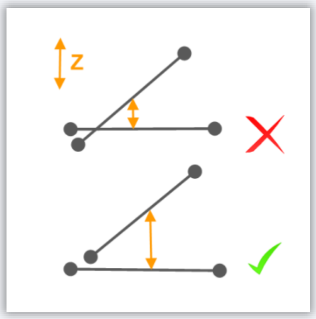
Z difference conditions at line intersections

Finds intersections where the Z value of the involved line features are smaller than a definable minimum value. Optionally, a maximum value for the Z difference may be specified.

Additionally, a SQL condition may be defined which compares the attribute values of the involved features at the intersection. The feature with the higher Z value is addressed with the alias “U”, the lower feature with the alias “L”. E.g., the expression `U.LEVEL > L.LEVEL` checks if the upper crossing line is assigned to a higher level (according to field “LEVEL”) than the lower crossing line.

With a minimum Z difference of 0, only the attribute condition is tested.

Test: *QaLineIntersectZ*

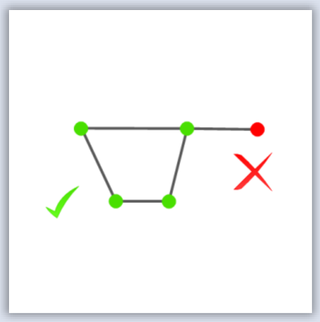


No dangles (unconnected line end points)

Finds line end points which are not coincident with other end points from a list of line feature classes (“dangles”).

Remark: This test checks *all* connections *after* applying the filter conditions. Specific attribute conditions for valid/invalid dangles may be defined with the tests *QaDangleCount* and *QaLineConnection*.

Test: *QaDangleFactory*



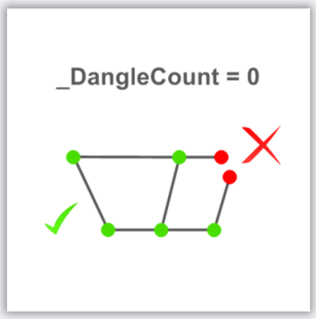
Valid number of dangles (unconnected line end points)

Finds line features with an invalid number of unconnected end points („dangles“). The valid number of dangles per feature may be defined using a SQL expression. The actual number of dangles can be retrieved from the calculated attribute `_DangleCount`. An error is reported if a feature does not fulfil the SQL expression.

SQL expression example: Features which are classified as dead ends with an attribute `is_dead_end` must have exactly one unconnected end point; all other features must be connected at both ends.

```
   _DangleCount == iif( is_dead_end == ‘yes’, 1, 0 )
```

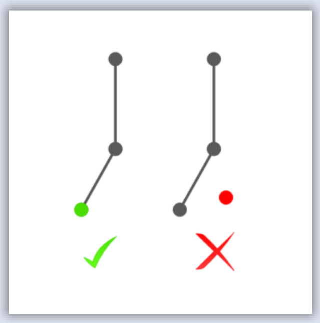
Test: *QaDangleCount*



No orphan nodes (points not connected to line end points)

Finds point features which are not coincident with start or end points of line features. Vice versa, it allows identifying line end points which do not coincide with point features. The latter allows checking the exact coincidence of junction features with line end points in geometric networks.

Test: *QaOrphanNode*



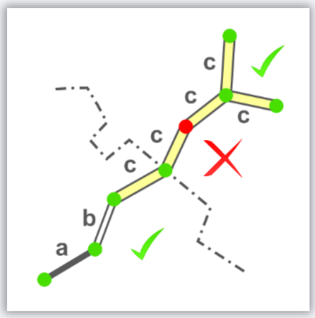
No pseudo nodes

Finds nodes between exactly two lines which hold the same values for relevant attributes ("pseudo nodes") and which are not justified by their location on any other feature (points, borderlines).

By default, all editable fields of a feature class are considered as relevant. Optionally, certain attributes may be excluded from the test. Differences only in these attributes do not lead to a node being considered valid.

Additionally, pseudo nodes can be allowed if they are located on certain points, lines or polygon boundaries. This allows for example to ignore pseudo nodes resulting from a forced splitting of features on administrative boundaries.

Test: *QaPseudoNodes*



No gaps in a connected group of lines

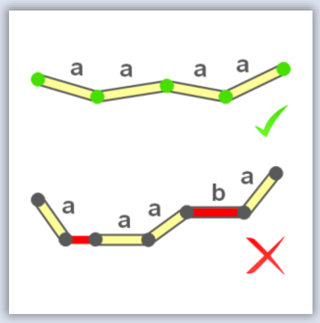
Finds gaps in groups of lines, from one or more feature classes, which must form a connected structure (e.g. rivers, road routes)

The features are grouped based on values in one or more fields, which may be from a joined table. All features with an identical value for this/these field(s) are assigned to the same group. A feature can belong to multiple groups defined in the same text field separated by a configurable delimiter. Thereby, a common method for modelling e.g. multiple routes running along the same road features is supported.

For a group containing unconnected features, the shortest line between disjoint parts will be reported as an error. Gaps longer than a specified tolerance value can be ignored. This allows to identify small gaps resulting from capturing errors, while allowing large, significant gaps between groups.

Optionally, special higher-order structures such as cycles and branches may be reported as errors.

Test: *QaGroupConnected, QaRelGroupConnected*



Linear Networks

Valid line and point connections

Checks if defined connection rules are fulfilled at connection points between line and point features. A list of rules may be specified defining the allowed feature combinations of a connection. If none of the rules are fulfilled, an error is reported for the connection.

The rules contain filter criterions for features as well as optional conditions for the number of involved features of a particular feature class (or for sub-selections of the involved features). The direction of the line (incoming, outgoing) at the connection point may also be used when formulating the rule.

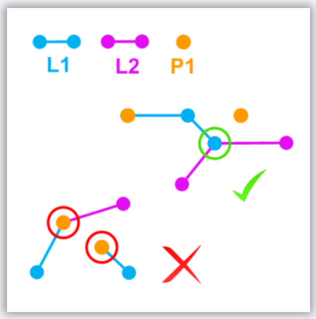
Connection points are defined as all line end points and points from the list of specified feature classes. For each connection point, the connected features are identified. These features are then tested with the connection rules.

The rules are therefore applied also to unconnected line endpoints, unconnected points and line connections without connected point objects. By defining a condition using the number of connected objects, locations with missing connections with other features can be detected.

Test: *QaLineConnection*

Feature class	P1	L1	L2
Rule group 1	True	True	False
Rule group 2	False	True	True

1. Connections of points in P1 and lines in L1 are allowed. Points in P1 may not connect to lines in L2.
2. Connections of lines in L1 and lines in L2 are allowed, as long as there is no point in P1 at the same location.

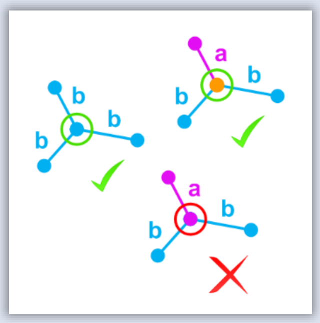


Consistent attribute value distribution at line connections

Checks conditions for distinct values of relevant attributes at connection points of line and point features.

For line features the following can be defined: All values of a relevant attribute must be identical, or they may/must be different. For point features the following can also be defined; a valid point must exist at a line connection and optionally, this point must have an attribute value identical to the most common or any attribute value of the connected lines.

Test: *QaLineConnectionFieldValues*



Consistent flow direction at connections of line features

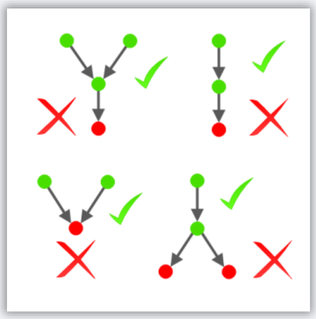
Checks flow logic in a linear network. All line end points which are not coincident with exactly one starting point of another line are reported as errors.

Usually it is assumed, that the flow direction corresponds to the direction of capture. Optionally, a SQL expression may be defined which allows the identification of features where the flow direction is opposite to that of capture.

At present, higher-ranking structures such as branches converging downstream, will not be considered.

The involved features may, but do not have to be part of a geometric network. Any existing flow direction information within the network will not be considered.

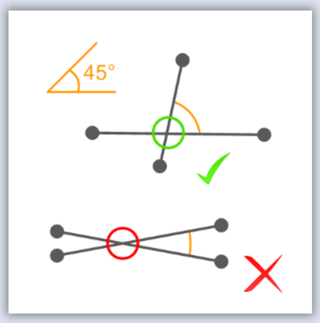
Test: *QaFlowLogic*



Minimum angle at line intersections

Finds intersections where line features cross at an angle which is smaller than a definable minimum value.

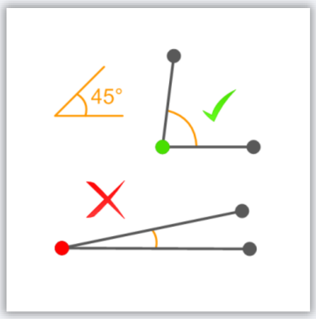
Test: *QaLineIntersectAngle*



Minimum angle between connecting lines

Finds connections where line features connect at an angle which is smaller than a definable minimum value.

Test: *QaMinAngle*

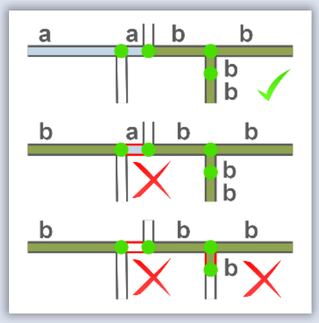


Constraints for line groups

Finds typical attribution errors in linear networks, where some features which should belong to a larger structure (e.g. a road route or a watercourse) have missing/incorrect attribute values. To detect these situations, constraints can be defined for connected groups of line features. The constraints may be based on the combined length of a connected group, on the length of dangling ends of a connected group, and the length of gaps to other groups along the underlying network.

The features are grouped based on values in one or more fields, which may be from a joined table. All features with an identical value for this/these field(s) are assigned to the same group. A feature can belong to multiple groups defined in the same text field separated by a configurable delimiter. Thereby, a common method for modelling e.g. multiple routes running along the same road features is supported.

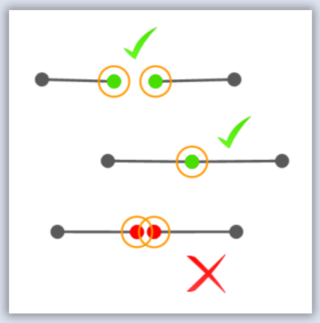
Test: *QaLineGroupConstraints, QaRelLineGroupConstraints*



Proximity

Conditions for distances between features

Points and line end points are either coincident or at a minimum distance

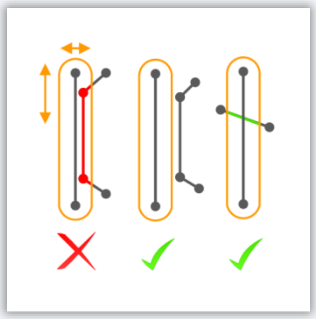


Finds point features and line end points which are closer to each other than a definable minimal distance, but are not exactly coincident. In the case of exactly coincident nodes with Z coordinates, it is also possible to check if the Z values do not differ too much.

To determine if two points are coincident, the XY tolerance of the spatial reference system is normally applied. Optionally, a further tolerance may be defined. With a negative tolerance, points considered as coincident will also be treated as an error. Thus, this test can also be used to determine if the general minimum distance between points and/or line end points is maintained.

Tests: *QaMinNodeDistance*

Minimum distance between lines/polygons



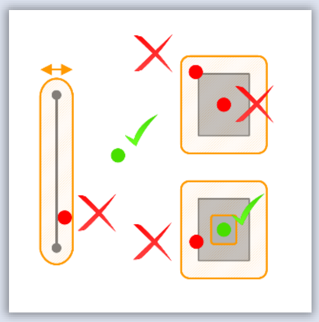
Finds sections of lines or polygon boundaries which do not maintain a minimum distance over a definable minimum length to other lines or polygon boundaries.

This test can be used to identify similar features or features which have been captured twice. With the definable minimum length, it can be defined over which distance two features need to run in close proximity, in order qualify as an error. The minimum length is therefore used to distinguish illegal nearly-coincident line sequences from allowed convergences such as line connections or crossings.

With a minimum length of 0 *every* violation of the minimum distance criterion is reported as an error.

Test: *QaNotNear, QaTopoNotNear*

Points are at a minimum distance to other lines or polygons



Finds point features that are within a specified distance from a line or polygon feature.

For each line or polygon class, the elements of the geometry to be considered can be defined. By default, the entire geometry will be used. Alternatively, the geometry boundary, the vertices, the end points or either the start or end point can be specified.

Optionally, points located exactly on the reference geometry can be allowed.

The minimum distance can either be defined as a constant, or be derived from attributes of the involved features.

Exceptions can be defined using SQL expressions which compare the attributes of two nearby features. In these expressions, the point feature can be addressed as “G1”, and the nearby feature as “G2”. If an expression is fulfilled for the two features, no error will be reported. A single expression can be specified to be used for all line/polygon feature classes, or individual expressions can be defined for each line/polygon feature class.

Test: *QaPointNotNear*

Proximity

Conditions for distances between features

Line sections are either coincident to or at a minimum distance from another line

Finds sections of lines or polygon boundaries which do not maintain a minimum distance to another line over a definable minimum length, without coinciding with the other line.

For the minimum length different values may be defined. On one hand for the case that a (boundary) line will coincide with the neighbouring line whilst within the proximity zone, on the other hand for the case that it will exit again the proximity zone, without ever coinciding with the neighbouring line.

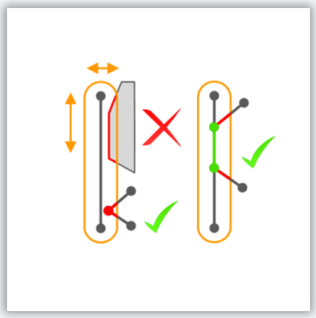
There is a tolerance for the coincidence of the line runs which can be set. If set to 0, the lines must lie exactly on top of one another to be considered as coincident. This allows to identify deviations smaller than the XY tolerance of the spatial reference.

This test detects small offsets between lines and/or polygons whose boundaries should be coincident. It can be used to find both data capturing errors and topology errors resulting from processing steps (e.g. geodetic transformations).

To get a complete picture of all convergences, gaps and overlaps among polygons, this test is often combined with *QaNoGaps* and *QaInteriorIntersectsSelf*.

This test exists in two variants, one for comparing features from two *different* sets of feature classes, and another one for comparing all features from a *single* set of feature classes.

Test: *QaPartCoincidenceOther, QaPartCoincidenceSelf*

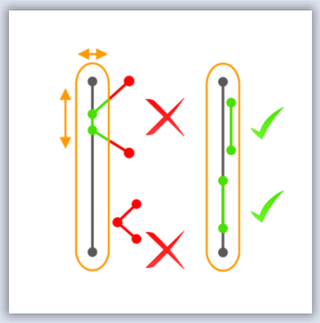


Complete lines run fully adjacent to reference lines

Finds all line and polygon sections which are further away from another line or polygon feature than a definable maximum distance. For polygon features, the boundaries are used.

For features with Z values, the distance can be measured either in 2D or 3D.

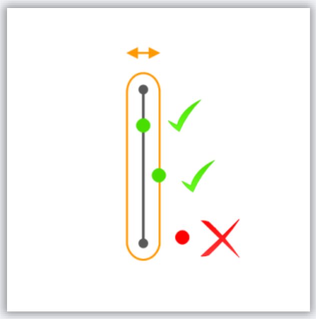
Test: *QaFullCoincidence*



Points are located near lines

Finds point features which are further away from a line feature or a polygon boundary than the allowed maximum distance.

Test: *QaPointOnLine*

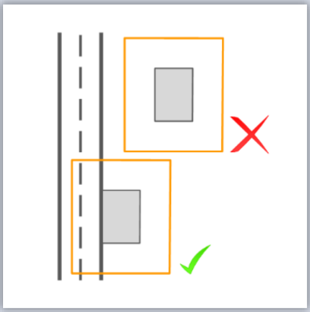


At least one expected feature in vicinity

Finds features (in a joined table) for which there is no neighbouring feature of an expected type within a defined distance. Examples are bus stops close to roads, required labelling objects or similar.

Optionally, a SQL expression can be defined which compares the attributes of two neighbouring features. In this expression, the features can be addressed as “G1”and “G2”. Is the expression for two neighbouring features not fulfilled, then the spatial relationship is considered as not relevant. In this case an error will be reported, unless there exists at least one, relevant feature within the search distance.

Test: *QaMustBeNearOther, QaRelMustBeNearOther*



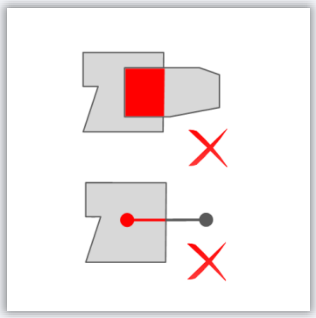
No violation of geodatabase topology rules

Reports errors detected by validating a geodatabase topology.

In case the verification process allows data editing, the topology will be validated during execution within the test extent. During this process the tested data may be modified by “cracking and clustering” the topology of the feature class. Thereby identified errors will be reported. If editing is not allowed, the topology validation is not executed, however previously identified topological errors are still reported. In this case, eventual areas with untested changes (“dirty areas”) will be reported as an error.

Test: *QaGdbTopology*

No interior intersection



Finds features whose interiors intersect with the interior of another feature.

Optionally, a SQL expression can be defined which compares the attributes of the polygons involved in the intersection. In this expression, the features can be addressed as “G1”and “G2”. In case the expression for two intersecting features is fulfilled, then the intersection is allowed and no error will be reported.

An additional SQL expression can be specified to allow certain interior intersections, based on properties of the intersection geometry (e.g. intersection areas smaller than a defined threshold). See description for “Constraints based on geometry properties” (Geometry tests) for a list of available geometry properties.

Example: The expression

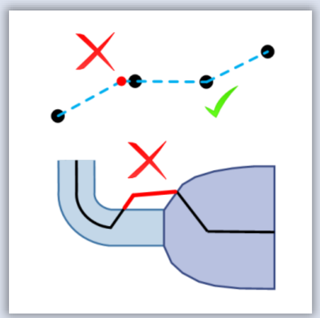
```
      $Area < 0.1 AND $SliverRatio > 100
```

ignores small and elongated intersection polygons.

This test exists in two variants, one for comparing features from two *different* sets of feature classes, and another one for comparing all features from a *single* set of feature classes.

Test: *QaInteriorIntersectsOther, QaInteriorIntersectsSelf*

Complete coverage



Finds features which are not fully covered by other features. Any geometry parts lying outside the covering features will be reported as errors. For each involved feature class, the elements of the geometry to be considered can be defined. By default, the entire geometry will be used. Alternatively, the geometry boundary, the vertices, the end points or either the start or end point can be specified.

In the case that a feature is not required to be *fully* covered, the maximum allowed percentage for the uncovered part can be defined. If the part of the uncovered area or length of a feature is greater than this value, an error will be reported. Additionally, a distance around the covering features can be specified within which the covered features must be located.

Optionally a SQL expression can be defined which compares the attributes of a feature that should be covered with a potentially covering feature. In this expression, the potentially covering feature is addressed as “G1” and the analysed feature as “G2”. If such an expression is defined and not fulfilled for both features, the coverage is ignored. This leads to an error being reported, unless another feature with valid coverage is present.

Test: *QaIsCoveredByOther*

Coincidence of vertices and edges

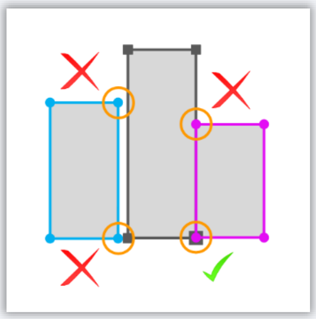
Finds vertices which are not coincident with a vertex and/or edge of a neighbouring feature. This test can be optionally carried out within the geometry parts of a feature, for example for multipatch rings.

Optionally, it can be defined if an error will be reported when a coincident vertex is missing on the neighbouring edge, even if this edge runs exactly through the tested vertex.

Background: Vertices which are not exactly snapped on each other, respectively missing vertices on otherwise coincident lines lead to small gaps or overlaps between features. With this test, the vertex differences that are responsible for these gaps/overlaps can be precisely identified.

This test exists in three variants: for comparing features from two *different* sets of feature classes, for comparing all features from a *single* set of feature classes, and for within-feature verification only (mainly useful for multipatch features).

Test: *QaVertexCoincidenceOther, QaVertexCoincidenceSelf, QaVertexCoincidence*



No gaps between polygons

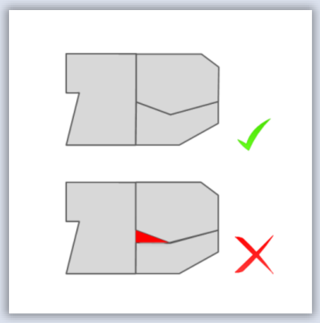
Finds gaps between polygons. Optionally, the search can be refined to small and/or elongated gaps. For this, a maximum area and/or a maximum “sliver” ratio for gaps can be specified. The “sliver” ratio is defined as the circumference raised to the second power and divided by the area. The following list shows the values (circumference ^{2} / area) for rectangles of different aspect ratios.

- Aspect ratio 1:1 (square): 16
- Aspect ratio 1:5: 28,8
- Aspect ratio 1:10: 48,4
- Aspect ratio 1:20: 88,2

Optionally, areas of interest can be specified within which gaps should be found. In this case, gaps between the polygons and the boundary of the containing area of interest will additionally be re-ported, and gaps outside any areas of interest will be ignored. This allows for example to test for complete coverage of administrative units by other polygons.

Further options can be used for testing gaps which are smaller than the XY tolerance of the spatial reference system and for memory optimization while checking very complex sets of polygons.

Test: *QaNoGaps*



No geometry duplicates

Finds features with equal geometry. Features with an empty 2D geometry difference are treated as duplicates. Z and M values are therefore ignored, as well as vertices that do not alter the geometry footprint (i.e. vertices along straight lines). The XY tolerance of the spatial reference system is used to detect significant differences.

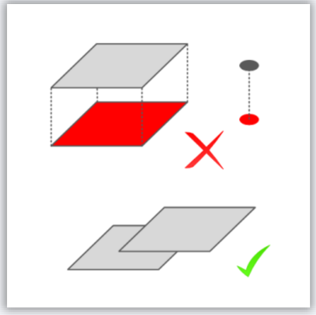
Optionally, a SQL expression can be defined which compares the attributes of two features with equal geometry. The matching features can be addressed as “G1” and “G2”. If the expression is ful-filled, the match is allowed and no error will be reported.

Example: The expression

```
G1.AdminLevel <> G2.AdminLevel
```

allows equal geometry of administration units, if they are not on the same administrative level.

Test: *QaDuplicateGeometrySelf*

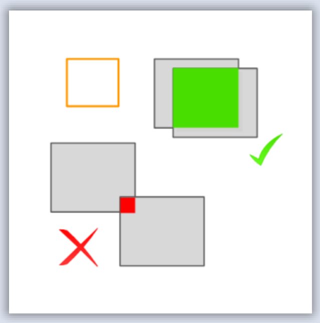


Minimal dimensions for intersection areas

Finds areas where polygons intersect, which are smaller than a definable minimum value.

For polygon features that may have valid intersection areas, this test detects allows identifying small, insignificant intersections.

Test: *QaMinIntersect*



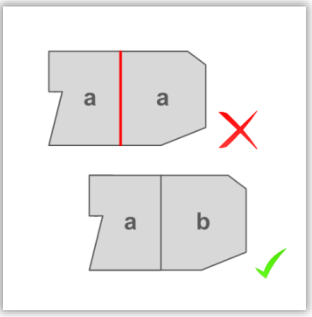
No unnecessary boundaries between polygons with identical field values

Finds touching polygons with identical values for a definable list of relevant fields.

As an alternative to the field list, a SQL expression can be used to compare attributes of two touching polygons. In this expression, the two involved features can be addressed as “L” and “R”. If the expression is fulfilled, no error will be reported.

Optionally, polygons touching at a single point can be ignored.

Test: *QaNeighbourAreas*



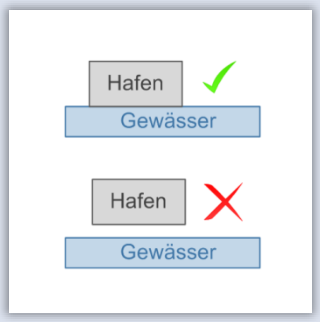
Expected neighboring features

Finds point, line and polygon features which do not touch any expected neighbouring feature.

Optionally, a SQL expression can be defined which compares the attributes of two touching features. In this expression two involved features can be addressed as “G1” and “G2”. If the expression for two features is not fulfilled, then the tangency is not considered as relevant. In this case an error will be reported, if no other relevant tangency with another feature exists.

This test exists in two variants, one for comparing features from two *different* sets of feature classes, and another one for comparing all features from a *single* set of feature classes.

Test: *QaMustTouchOther, QaMustTouchSelf*



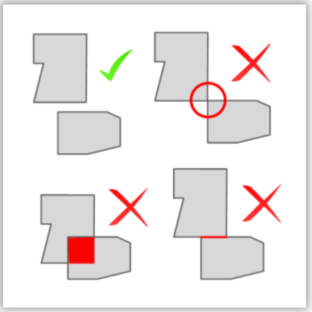
No intersection or tangency (spatial relation „intersects“)

Finds features which are not disjoint from other features.

Exceptions can be defined with a SQL expression which compares the attributes of two touching features. In this expression two involved features can be addressed as “G1” and “G2”. If the expression is fulfilled for the two features, the tangency is allowed and no error will be reported.

This test exists in two variants, one for comparing features from two *different* sets of feature classes, and another one for comparing all features from a *single* set of feature classes.

Test: *QaIntersectsOther, QaIntersectsSelf*

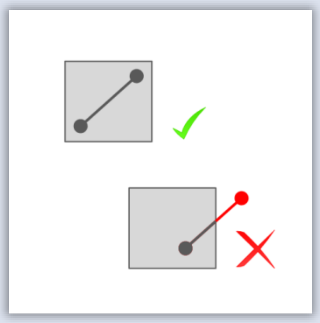


Complete enclosure (spatial relation „contains“)

Finds features which are not completely contained by another feature. Geometry parts lying outside a single enclosing feature will be reported as errors.

Optionally, a SQL expression can be defined which compares the attributes of the containing and the enclosed feature. In this expression, the containing feature can be addressed as “G1” and the enclosed feature as “G2”. If the expression is not fulfilled for the two features, then the enclosure is not considered relevant. In this case an error will be reported, if there’s no other relevant containing feature.

Test: *QaContainsOther*



No crossing (spatial relationship „crosses“)

Finds features that “cross” another feature. Applicable for the following geometry type combinations: line/line, line/polygon, multipoint/polygon and multipoint/line.

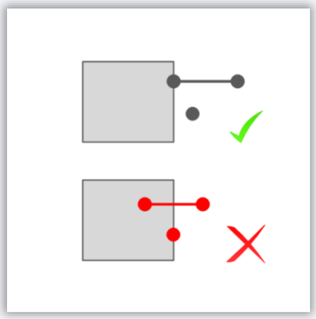
Criteria for “crosses”

- The features intersect the exterior of another feature.
- The dimension of intersection geometry of the features interiors is *smaller* than the maximum dimension of the two features.

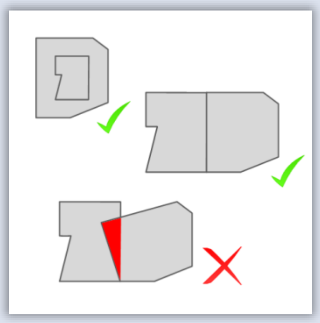
Exceptions can be defined with a SQL expression which compares the attributes of two crossing features. In this expression, the crossing features can be addressed as “G1” and “G2”. If the expression is fulfilled for the two features, the crossing is valid and no error will be reported.

This test exists in two variants, one for comparing features from two *different* sets of feature classes, and another one for comparing all features from a *single* set of feature classes.

Test: *QaCrossesOther, QaCrossesSelf*



No overlap (spatial relationship „overlaps“)



Finds features that “overlap” another feature. Applicable for the following geometry type combinations: line/line, polygon/polygon and multipoint/multipoint.

Criteria for “overlaps”:

- The features intersect the exterior of another feature
- The dimension of the intersection geometry of the features interiors is *equal* to the dimension of the two features.

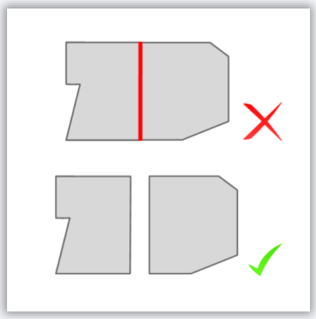
So the technical definition of “overlaps” does not correspond to the colloquial meaning of overlaps. For example, a polygon that is completely contained by another polygon does not fulfil the “overlaps” relationship because the contained polygon does not intersect the exterior of the other polygon.

Exceptions can be defined with a SQL expression which compares the attributes of two “overlapping” features. In this expression, they can be addressed as “G1” and “G2”. If the expression is fulfilled for the two features, then the overlap is valid and no error will be reported.

This test exists in two variants, one for comparing features from two *different* sets of feature classes, and another one for comparing all features from a *single* set of feature classes.

Test: *QaOverlapsOther, QaOverlapsSelf*

No touching features (spatial relationship „touches“)



Finds point, multipoint, line or polygon features which touch other features. The test is applicable to polygon/polygon, line/line, line/polygon, point/line, multipoint/line relationships.

Criteria for “touches”:

- The features are not separated from each other (their intersection geometry is not empty)
- The intersection geometry of the *interiors* of two features is empty

Exceptions can be defined with a SQL expression which compares the attributes of two touching features. In this expression the touching features can be addressed as “G1” and “G2”. If the expression is fulfilled for the two features, then the “touches” relationship is valid and no error will be reported.

This test exists in two variants, one for comparing features from two *different* sets of feature classes, and another one for comparing all features from a *single* set of feature classes.

Test: *QaTouchesOther, QaTouchesSelf*

No spatial relationship according to the 9IM intersection matrix

Finds features which have a definable spatial relationship with another feature. The type of spatial relationship is specified by the 9IM intersection matrix.

Optionally, exceptions may be defined for the intersection geometry occurring between two features according to the matrix, this can be potentially multipart. A list of allowed geometry dimensions can be specified for the intersection geometry (point, line or area parts of the intersection geometry). Errors will only be reported for parts of the intersection geometry whose geometry dimension is not listed.

Exceptions can be defined with a SQL expression which compares the attributes of two features with a relationship according to the 9IM intersection matrix. In this expression the two features can be addressed as “G1” and “G2”. If the expression is fulfilled for the two features, then the spatial relationship is valid and no error will be reported.

This test exists in two variants, one for comparing features from two *different* sets of feature classes, and another one for comparing all features from a *single* set of feature classes.

Test: *QaIntersectionMatrixOther, QaIntersectionMatrixSelf*

Example: Intersection of the interiors (“interior intersects”)

9IM-Matrix		G2		
		I	B	E
G1	I	True	*	*
	B	*	*	*
	E	*	*	*

I: Interior
B: Boundary
E: Exterior

The cells of the matrix are read row by row and linked together one after another, in order to generate the matrix text string. For example:

```
T********
```

Spatial relationship according to 9IM intersection matrix must exist

Finds features which do not have a neighbouring feature for which the specified 9IM intersection matrix is fulfilled.

Optionally, additional rules may be defined for the intersection geometry of two neighbouring features, according to the matrix. A list of geometry dimensions can be specified for the intersection geometry (point, line or area parts of the intersection geometry), which are either *mandatory* or *not allowed*. Only if these additional rules are fulfilled, is the spatial relationship to the neighbouring feature considered valid. For example, in case of a mandatory boundary intersection of the features, it can be additionally required for the intersection geometry to be linear, so that the spatial relationship with the neighbouring feature is considered relevant.

Further constraints can be defined using a SQL expression which compares the attributes of two features with a relationship according to the 9IM intersection matrix. In this expression, the two features can be addressed as “G1” and “G2”. A pair of features is considered to have a relevant relationship only when the attribute condition is either undefined or fulfilled.

Test: *QaMustIntersectMatrixOther*

Example: Intersection of geometry boundaries

9IM-Matrix		G2		
		I	B	E
G1	I	*	*	*
	B	*	True	*
	E	*	*	*

I: Interior
B: Boundary
E: Exterior

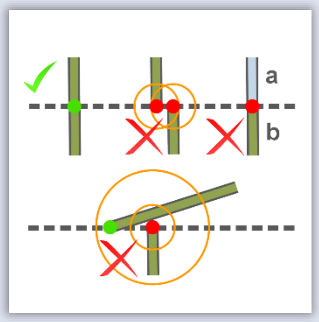
The cells of the matrix are read row by row and linked together one after another, in order to generate the matrix text string. For example:

```
****T****
```

Edge matching

Topological and attribute consistency along borders

Consistent continuation of border-crossing lines



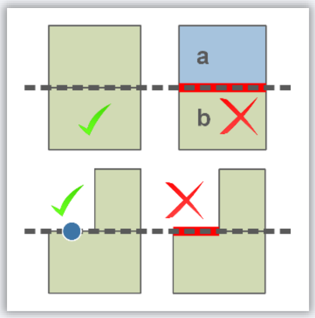
Finds line features that connect to a border line, but do not have an expected continuation on the other side of the border.

Rules may be specified regarding the topological and attribute consistency of a corresponding pair of lines connected to each side of the border, and whether a continuing line is expected on the other side of the border. Attribute rules may be expressed as comparison expressions, and/or a list of fields that are expected to have equal values (with options for ignoring specific values and support for multi-valued fields, using a separator character).

Borders may be represented as line or polygon features. If separate border representations exist per area, then the reported issues indicate if a mismatch is caused by the fact that borders are not coincident at the crossing location.

Test: *QaEdgeMatchCrossingLines*

Consistent continuation of border-crossing areas



Finds polygon features that touch a border line, but do not have an expected continuation on the other side of the border.

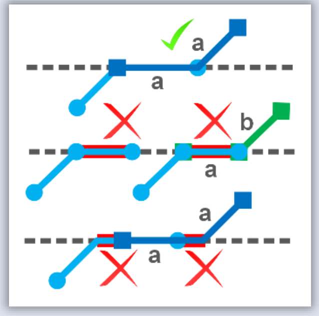
Rules may be specified regarding the topological and attribute consistency of a corresponding pair of polygons adjacent to each side of the border, and whether a continuing polygon is expected on the other side of the border. Attribute rules may be expressed as comparison expressions, or list of fields that are expected to have equal values (with options for ignoring specific values and support for multi-valued fields, using a separator character).

Additional features may be specified which justify an unmatched polygon boundary section along the border, i.e. no issue is reported if such a feature touches the border at that location.

Borders may be represented as line or polygon features. If separate border representations exist per area, then the reported issues indicate if a mismatch is caused by the fact that borders are not coincident where the polygons touch the border.

Test: *QaEdgeMatchCrossingAreas*

Correspondence of bordering lines



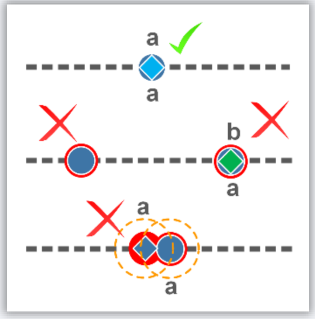
Finds line features that follow along a border line, but do not have a corresponding feature on the other side of the border.

Rules may be specified regarding the topological and attribute consistency of a corresponding pair of lines that follow along each side of the border, and whether a corresponding line is expected on the other side of the border. Attribute rules may be expressed as comparison expressions, and/or a list of fields that are expected to have equal values (with options for ignoring specific values and support for multi-valued fields, using a separator character).

Borders may be represented as line or polygon features. If separate border representations exist per area, then the reported issues indicate if a mismatch is caused by the fact that borders are not coincident where the lines follow along the border.

Test: *QaEdgeMatchBorderingLines*

Correspondence of bordering points



Finds point features that are located on a border line, but do not have a corresponding feature on the border line associated with the neighboring area.

Rules may be specified regarding the topological and attribute consistency of a corresponding pair of points located on the border, and whether a corresponding point is expected for the neighboring area. Attribute rules may be expressed as comparison expressions, and/or a list of fields that are expected to have equal values (with options for ignoring specific values and support for multi-valued fields, using a separator character).

Borders may be represented as line or polygon features. If separate border representations exist per area, then the reported issues indicate if a mismatch is caused by the fact that borders are not coincident where the point features touch the border.

Test: *QaEdgeMatchBorderingPoints*

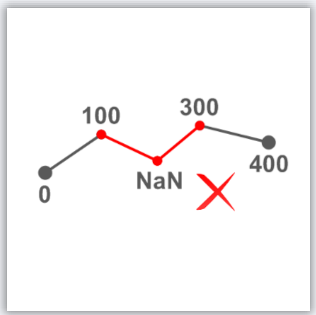
M Values

Conditions for M values of features

Valid M values

Finds M values on line, polygon, point, multipoint and multipatch features which are not defined (NaN), or which have a specified invalid value.

Test: *QaMeasures*



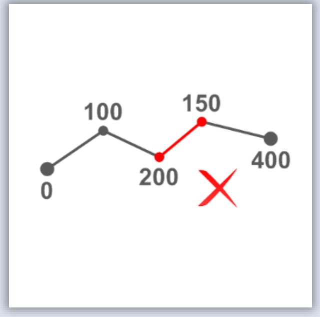
Monotonically increasing or decreasing M values

Finds segment sequences on line features which do not have monotonically increasing or decreasing M values, i.e. their M values change contrary to the expected direction. The expected direction of monotonicity can be specified based on the direction of capture or the M values of the end points.

Optionally, a SQL expression can be defined which identifies features where the direction should be inverted compared to that of capture.

Additionally, segment sequences with constant M values can optionally be allowed.

Test: *QaMonotonicMeasures*



Consistency of line M values with calibration point values

Finds M values along line features which differ from an attribute or M value of a neighbouring point feature. The tolerance for the maximum allowed value of deviation may be specified.

This test identifies locations along lines where the M values do no longer correspond to the calibration base and might have to be re-calculated.

For the allocation of points to line features the search distance and an optional SQL expression may be defined. In this expression, the point can be addressed as “P” and the line as “L”. Only if the expression is fulfilled for a point and a line, the expected M value of the point feature is relevant for the line.

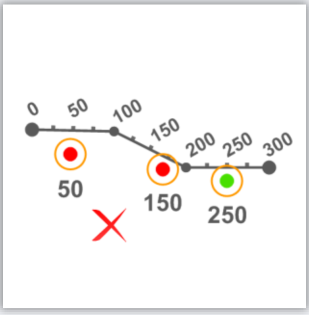
Example: The expression

```
    P.WATERCOURSE_ID = L. WATERCOURSE_ID
```

guarantees that only points assigned to the same watercourse are considered when checking the line’s M values.

Additionally, rules can be specified on how to determine the M value on the line (nearest vertex, nearest location on line) and how the desired M value of the point should be determined (deduction from one or several attribute(s) or directly from the M value).

Test: *QaMeasuresAtPoints*

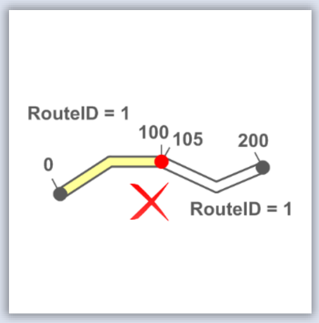


Continuous M values on line connections along a route

Finds discontinuities in M values at connections of line features which are part of the same route.

Connected line end points that are part of the same route (having an identical value for the route ID attribute) must have M values which do not differ from each other more than the M tolerance of the spatial reference system.

Test: *QaRouteMeasuresContinuous*



M Values

Conditions for M values of features

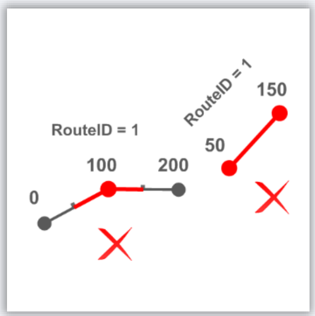
Unique M values along routes

Finds segment sequences in line features belonging to the same route that have identical M values. Such non-unique M ranges prevent the unambiguous localization of M values along a route.

At line connections along a route, overlaps of M values within a definable M tolerance are allowed. In all other cases, identical or overlapping M values are not allowed.

The individual, non-unique M value ranges will be reported as errors. The error geometry of such an M value range can span several features.

Test: *QaRouteMeasuresUnique*



Z Values

Conditions for Z values of features

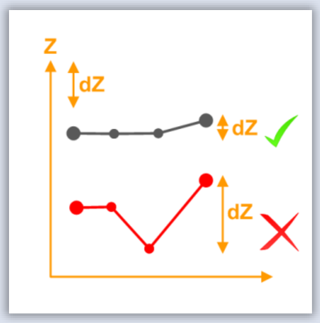
Maximum Z difference within a geometry

Finds line, polygon, multipoint or MultiPatch features whose maximum or minimum Z values differ more from each other than a definable tolerance.

This test checks for example whether lakes have (sufficiently) constant Z values.

In case of an error, the vertices will be reported that differ by more than half the defined tolerance of from the most common Z value.

Test: *Qa3dConstantZ*

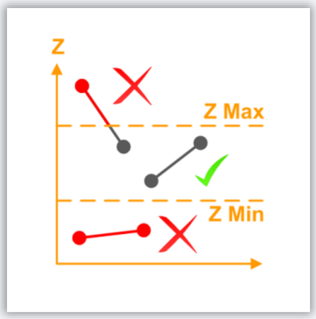


Height values within a Z range

Finds points, lines, polygon, multipoint and MultiPatch features whose Z values lie outside an allowed Z range. If the Z value is smaller than a specified lower Z limit or bigger than a upper limit, it is considered as an error.

For lines and polygons, segments and segment parts lying outside the Z range limits will be reported as errors. For multipoint and MultiPatch features, an error will be reported for all points outside the same range limits.

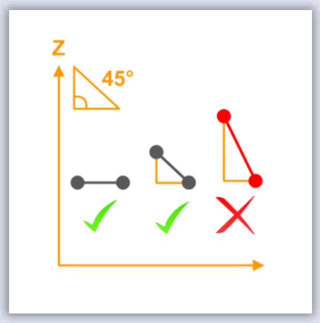
Test: *QaWithinZRange*



Maximum angle of slope

Finds segments of line or polygon features whose slope is steeper than a specified maximum angle.

Tests: *QaMaxSlope*



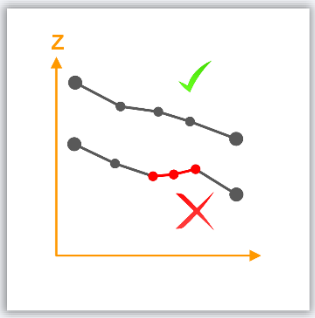
Monotonically increasing or decreasing Z values

Finds segment sequences on line features which do not have monotonically increasing or decreasing Z values, i.e. their Z values change contrary to the expected direction. The expected direction of monotonicity can be specified based on the direction of capture or the Z values of the end points.

Optionally, a SQL expression can be defined which identifies features where the direction should be inverted compared to that of capture.

Additionally, segment sequences with constant Z values can optionally be allowed.

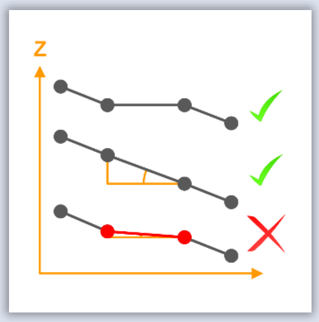
Test: *QaMonotonicZ*



Horizontal Segments

Finds segments in polygon, line and multipatch features that are almost, but not sufficiently horizontal.

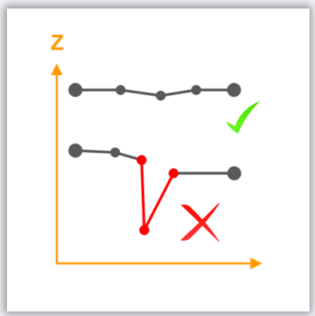
Test: *QaHorizontalSegments*



Maximum curvature

Finds segments of line and polygon features for which the curvature along the Z profile (i.e. the rate of slope angle change, or second deviation of the profile curve) exceeds a defined value.

Test: *QaSmooth*

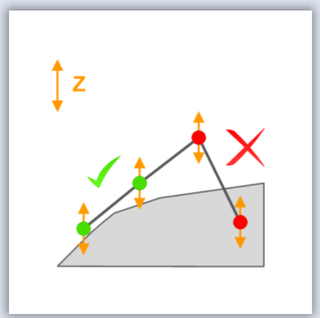


Distance from terrain surface (vertices)

Finds point features and line/polygon vertices whose Z values differ more than a specified value from the surface of a digital terrain model (stored as a raster, mosaic, or terrain dataset). This allows finding features that are expected to have their vertices on or near the terrain surface, but have a Z deviation exceeding a defined tolerance at their vertex locations.

Alternatively, this test can be reversed to test if points/vertices keep a minimum distance from the terrain. It can be specified if the points/vertices are expected to lie only above, only below, or either above and below, the terrain surface.

Test: *QaSurfaceVertex*



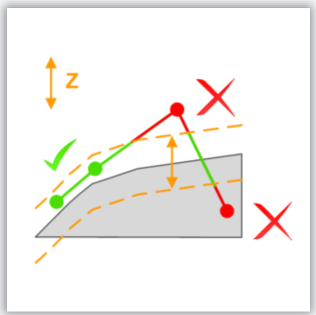
Distance from terrain surface (feature Z profiles)

Finds sequences of segments and segment parts of line and polygon features which lie outside a specified tolerance range from the surface of a digital terrain model (stored as a raster, mosaic, or terrain dataset). This allows finding features that are expected to follow the terrain surface along their entire Z profile, i.e. not only at the location of their vertices, but which have a Z deviation exceeding a defined tolerance anywhere along their profile.

Alternatively, this test can be reversed to test if the entire Z profile of the features maintains a minimum distance from the terrain surface. It can be specified if the feature's Z profile is expected to lie above, below, or either above and below, the terrain surface.

For line features, a zone around the end points can optionally be defined which will be ignored. This allows testing that bridges have a minimum distance from the terrain surface, without reporting their end points (which connect to the terrain surface) as errors.

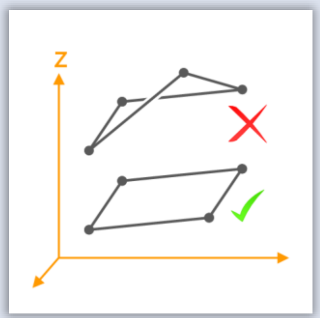
Test: *QaSurfacePipe*



Coplanar Rings

Finds MultiPatch or polygon rings whose vertices are not coplanar, which means they are not in the same geometric plane (according to a tolerance for the maximum allowed Z offset from the regression plane through the vertices).

Test: *QaCoplanarRings*



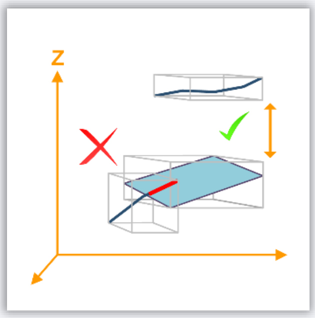
Minimum Z distance between intersecting features

Finds features intersecting in 2D which have an insufficient Z distance. The Z distance is determined based on the 3D envelopes of the involved features.

Additionally, a SQL condition may be defined which compares the attribute values of a pair of intersecting features. The feature with the higher minimum Z value is addressed with the alias "U", the lower feature with the alias "L". E.g., the expression `U.LEVEL > L.LEVEL` checks if the upper feature is assigned to a higher level (according to field "LEVEL") than the lower feature.

This test exists in two variants, one for comparing features from two *different* sets of feature classes, and another one for comparing all features from a *single* set of feature classes.

Test: *QaZDifferenceOther, QaZDifferenceSelf*



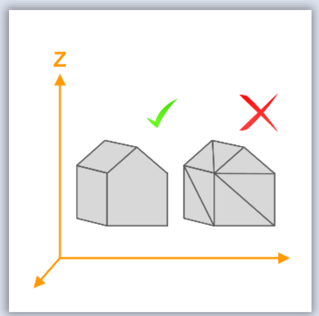
3D Buildings

Restriction of allowed MultiPatch part types

Finds MultiPatch features containing specific types of geometry parts.

MultiPatch features can consist of a combination of rings, triangle fans, triangle strips and triangle patches. In certain situations, e.g. when using certain editing workflows, some of these types may be undesired. The allowed geometry types can be individually defined.

Test: *QaMpAllowedPartTypes*

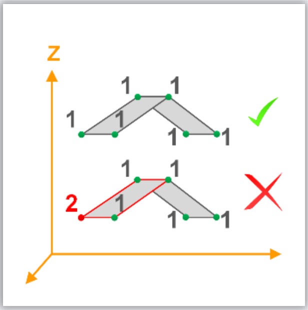


Constant point IDs per ring

Finds MultiPatch features where the point IDs are not constant within a ring.

Some MultiPatch editing tools use point IDs to define higher-level structures consisting of multiple rings. These tools require that the point IDs within a ring are constant.

Test: *QaMpConstantPointIdsPerRing*



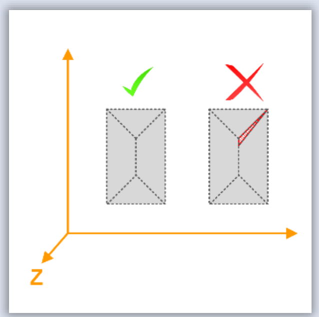
No holes in MultiPatch footprints

Finds MultiPatch features whose 2D footprints contain holes.

Optionally, all, or all horizontal, inner rings can be allowed. Also, holes greater than a specified area can be allowed.

To detect very small holes, the test may use an XY resolution/tolerance that is reduced by a specified factor.

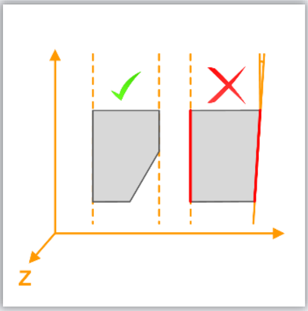
Test: *QaMpFootprintHoles*



Consistent azimuth angles of horizontal MultiPatch segments

Finds pairs of horizontal segments in MultiPatch features whose azimuth angles are almost, but not sufficiently equal. The comparison can either be applied to the entire feature, or separately to each MultiPatch ring.

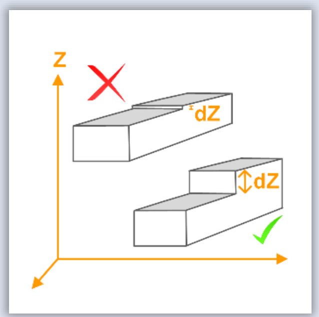
Test: *QaMpHorizontalAzimuths*



Consistent Z values of horizontal MultiPatch segments

Finds pairs of horizontal segments in MultiPatch features whose Z values are almost, but not sufficiently equal.

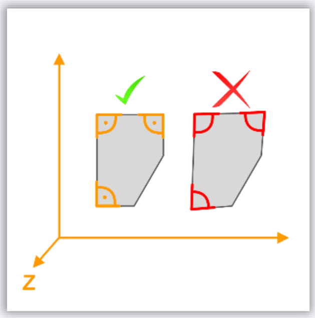
Test: *QaMpHorizontalHeights*



Consistent right angles between horizontal MultiPatch segments

Finds pairs of horizontal segments in MultiPatch features that are almost, but not sufficiently perpendicular to each other. Optionally, the comparison can be restricted to connected or nearby segments.

Test: *QaMpHorizontalPerpendicular*



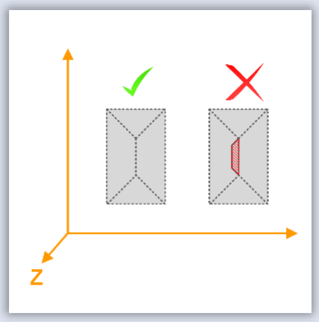
No intersection between MultiPatch ring footprints

Finds intersections between 2D footprints of different rings of the same MultiPatch feature.

Optionally, intersections may be allowed between rings whose vertices all have a constant point ID value per ring, and where this constant value is different for the intersecting rings.

To detect very small intersections, the test may use an XY resolution/tolerance that is reduced by a specified factor.

Test: *QaMpNonIntersectingRingFootprints*

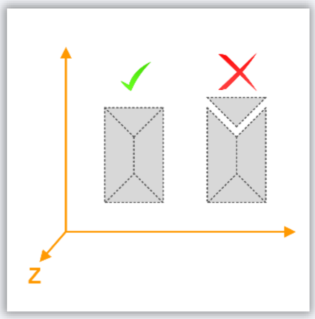


No multipart 2D footprints of MultiPatch features

Finds MultiPatch features with 2D footprints consisting of multiple disjoint parts.

To detect very small disjoint parts, the test may use an XY resolution/tolerance that is reduced by a specified factor.

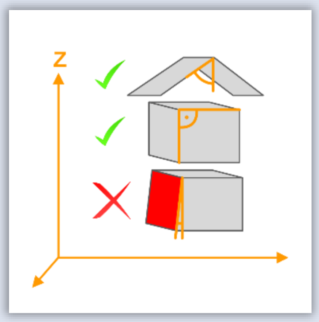
Test: *QaMpSinglePartFootprint*



Consistent vertical walls in MultiPatch features

Finds faces in MultiPatch features that are almost, but not sufficiently vertical.

Test: *QaMpVerticalFaces*

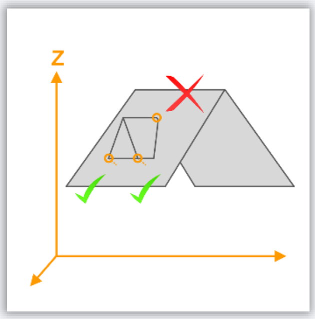


No small Z offsets between feature vertices and MultiPatch faces

Finds vertices whose Z values are near, but not sufficiently close to a ring or triangle of a MultiPatch feature. The vertices can be from any type of feature class.

Separate near tolerances can be specified for vertices above or below the MultiPatch face. The treatment of not sufficiently coplanar rings can be defined. Optionally, faces with a slope smaller than a specified value are ignored.

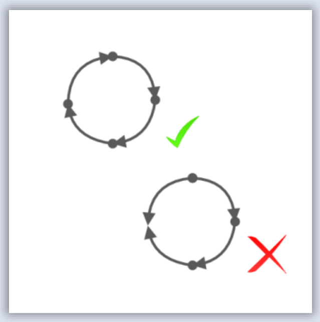
Test: *QaMpVertexNotNearFace*



Invalid rings

Checks if line features form closed and consistently oriented rings. Line features that do not belong to a correctly formed ring will be reported as errors. The expected direction of rotation of the rings (clockwise or counter-clockwise) can be specified.

Test: *QaBorderSense*

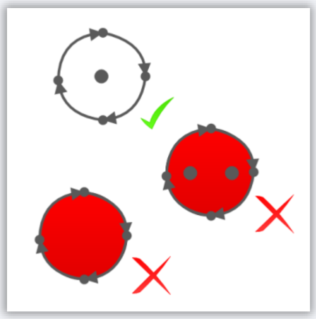


Centroids

Checks if closed rings formed by connected line features contain exactly one point feature.

Additionally, the attributes of a boundary line and the centroids of the adjacent rings may be compared. For this, a SQL expression can be defined which addresses the boundary line as “B”, the right-hand centroid as “R” and the left-hand centroid as “L”. This allows checking if the neighbouring centroids of boundary lines of a given type have identical or different attributes (useful for certain representations of administrative boundaries / areas).

Test: *QaCentroids*



Schema

Conditions for schemas of tables/feature classes

No invalid field alias names

Checks the alias names of fields in a table. Rules may be specified for the maximum length, for the capitalization (upper/lower case) and for expected differences from the field name. For system fields, it may be specified if custom alias names are allowed which then must obey the defined rules. Optionally, the uniqueness of field alias names within the table can also be checked for.

Test: *QaSchemaFieldAliases*

No invalid field names

Checks the field names of a table. Rules may be specified for the maximum length and for the capitalization (upper/lower case). It can be tested if the field names are unique according to a definable number of characters at the name's beginning. This is relevant for example when exporting to shapefiles, where field names are cut off after the first 10 characters.

Test: *QaSchemaFieldNames*

Regular expressions for field names

Checks if a table's field names match a definable regular expression.

This offers the possibility for checking if field names contain for example no umlauts or any other special characters.

Test: *QaSchemaFieldNameRegex*

No fields with reserved field names

Checks that the fields of a table don't use prohibited, reserved names. These reserved names can be listed in the test itself or read from an external table. If the reserved names are stored in a table, the reason for not using that name may also be specified along with an alternative valid name which should be used instead. This may be used to standardize the names for commonly used fields, by listing non-standard name variants and indicating, the expected standard name.

Test: *QaSchemaReservedFieldNames, QaSchemaReservedFieldNameProperties*

Expected field properties

Checks if a specified field in a table has expected field properties. The following properties can be checked: data type, field length, field alias, domain name. Additionally, it can be specified if the field *must* exist in the table.

QaSchemaFieldPropertiesFromTable checks if a specified field in a table has expected field properties based on a list of field specifications defined in another table. The field specifications table can be filtered to a subset of rows relevant for the verified table.

Test: *QaSchemaFieldProperties, QaSchemaFieldPropertiesFromTable*

Fields refer to domains with a compatible data type

Checks if the domains referenced by the fields of a table match the data type of the referencing fields.

Test: *QaSchemaFieldDomains*

No invalid domain names

Checks the names of domains that are referenced from a specified table. Rules may be specified for the maximum name length and for capitalization (upper/lower case). Optionally, it can be specified that the domain name must contain the name of the referencing field, or that it starts with an expected prefix.

Test: *QaSchemaFieldDomainNames*

No invalid domain descriptions

Checks the descriptions of all domains which are referenced from a specified table. The maximum length of the description text can be specified. Optionally it can be tested if the domain description is unique within the workspace of the tested table or within another workspace (into which the domain might have to be imported).

Test: *QaSchemaFieldDomainDescriptions*

Schema

Conditions for schemas of tables/feature classes

Regular expressions for domain names

Checks the names of domains that are referenced from a specified table match a definable regular expression.

This offers the possibility for checking if domain names contain for example no umlauts or any other special characters.

Test: *QaSchemaFieldDomainNameRegex*

Coded value domains with valid value lists

Checks the value lists of the coded value domains that are referenced from a specified table. Rules may be defined for the maximum length of coded value names, for their uniqueness and the for minimum number of values within a domain. Additionally, it may be specified that a definable minimum number of value names (description) must differ from their codes.

Test: *QaSchemaFieldDomainCodedValues*

Expected spatial reference

Checks that the horizontal coordinate system of a spatial reference of a feature class conforms to a specified expected spatial reference. The expected spatial reference may be defined by a reference feature class or an XML String. It can be defined if the vertical coordinate system must also be identical, and if the coordinate storage settings (origin and precision of XY, Z and M values), optionally with the corresponding tolerances, must be equal.

Test: *QaSchemaSpatialReference*

No Category

Expected number of rows in tables or feature classes

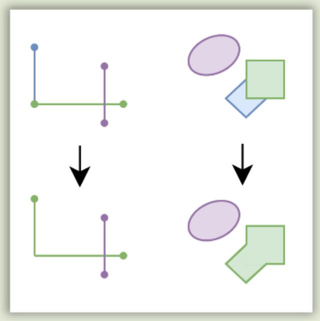
Checks if the number of rows in a table or feature class is within an expected range. The range is defined with a lower and an upper limit, either as absolute numbers or relative to the number of rows in a reference table. Optionally, the upper limit can be ignored.

Test: *QaRowCount*

Dissolving connected features

Finds and dissolves connected features into a single feature, optionally grouped by specific attributes. This transformer is particularly useful to simplify fragmented features by dissolving the related elements, e.g. road sections with the same name.
Supported geometry types: polylines and polygons

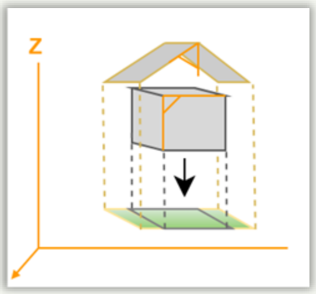
Transformer: *TrDissolve*



Creating a 2D footprint from 3D objects

Generates 2D footprint polygons from multipatch geometries. This is helpful for further analysis of 2D geometry relations, such as intersections or proximity.

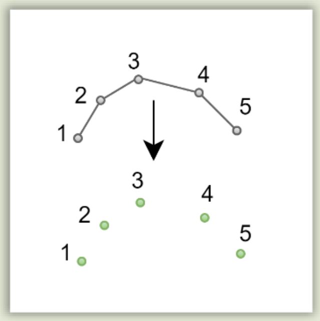
Transformer: *TrFootprint*



Extracting points from geometries

Extracts points from multipoints, lines, polygons or multipatches. It can be specified whether to extract all vertices or only specific points, such as the start point, end point, centroid, label point etc.

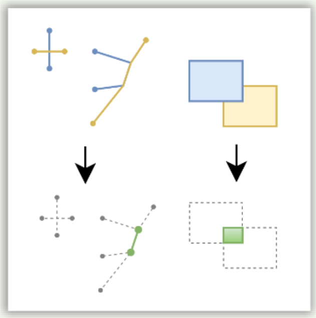
Transformer: *TrGeometryToPoints*



Finding intersections between features

Finds intersecting areas between two geometry data sets and creates new geometries from them. The optional ResultDimension parameter controls the dimension of the resulting geometries, whereby the minimum dimension of the input geometries determines the maximum dimension of the result geometry. The figure shows the result for the default value of the ResultDimension (the result dimension is determined automatically based on the input data).

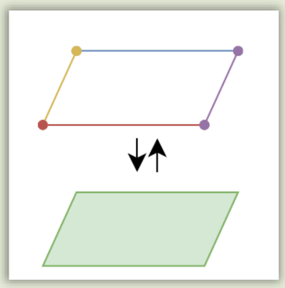
Transformer: *TrIntersect*



Converting lines to polygons or vice versa

Converts closed lines into polygons or the boundary of polygons into lines.

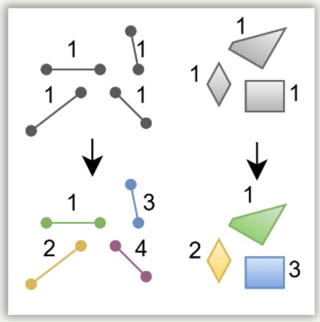
Transformer: *TrLineToPolygon, TrPolygonToLine*



Splitting multipart geometries

Splits multipart lines or polygons into individual elements. This helps simplify and analyze features more effectively by treating connected segments as separate units.

Transformer: *TrMultilineToLine, TrMultipolygonToPolygon*



Coordinate transformation of a dataset

Projects a feature class into a new coordinate system. This is useful to harmonize datasets with different spatial references or to prepare them for specific analyses in a given reference system.

Transformer: *TrProject*

Creating a table from existing data

Creates a table from existing data or SQL queries for further analysis or data linkage. This is particularly useful when tables or views are not directly managed within a geodatabase.

Transformer: *TrMakeTable*

Spatial join of features datasets

Performs a spatial join between two feature datasets. Attributes of features that spatially overlap or meet a defined relationship are transferred. This is used to integrate additional information from adjacent or overlapping objects into a table.

Transformer: *TrSpatialJoin*

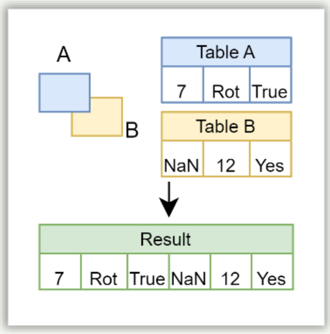


Table A		
7	Rot	True

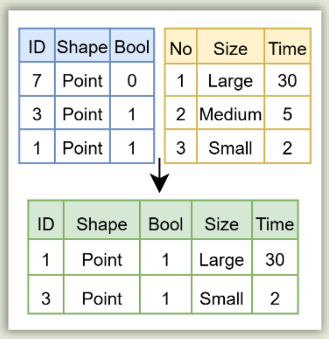
Table B		
NaN	12	Yes

Result					
7	Rot	True	NaN	12	Yes

Table join by attributes

Joins two tables based on a common key field. This method is used to transfer information from a referenced table into a main table, such as adding detailed metadata or classifications.

Transformer: *TrTableJoin*



ID	Shape	Bool
7	Point	0
3	Point	1
1	Point	1

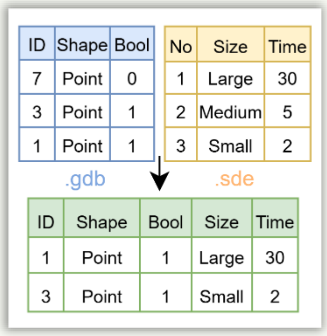
No	Size	Time
1	Large	30
2	Medium	5
3	Small	2

ID	Shape	Bool	Size	Time
1	Point	1	Large	30
3	Point	1	Small	2

In-Memory table join

Enables fast joining of two tables directly in memory, without storing the result permanently in the database. This improves performance for temporary or frequently updated data joins. It is particularly useful for complex data analysis with multiple queries.

Transformer: *TrTableJoinInMemory*



ID	Shape	Bool
7	Point	0
3	Point	1
1	Point	1

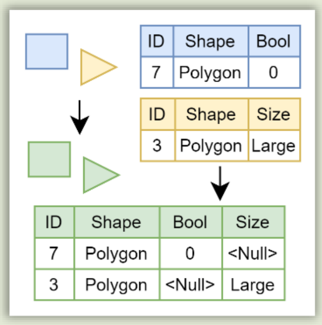
No	Size	Time
1	Large	30
2	Medium	5
3	Small	2

ID	Shape	Bool	Size	Time
1	Point	1	Large	30
3	Point	1	Small	2

Table append

Appends one table to another. This is used to merge datasets while preserving their structure. The prerequisite is that both tables have the same geometry type (e.g., point to point, line to line, polygon to polygon).

Transformer: *TrTableAppend*



ID	Shape	Bool
7	Polygon	0

ID	Shape	Size
3	Polygon	Large

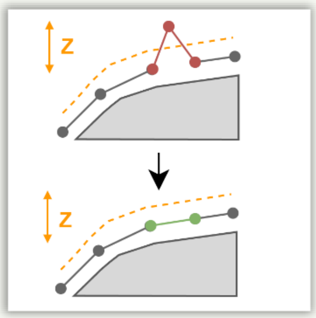
ID	Shape	Bool	Size
7	Polygon	0	<Null>
3	Polygon	<Null>	Large

Z-Values

Assigning Elevation Values from Raster Data

Assigns elevation values from a raster dataset to vector features. This is important for calculating elevation profiles or analyzing topographic characteristics.

Transformer: *TrZAssign*



Input Feature Filter

Combined filtering of features

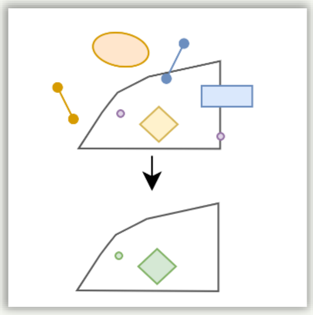
Allows the combination of several input feature filters (transformers with the name ‘TrOnly…’ conditions for spatial based checks). This is useful for defining complex selections by using and combining filters with logical operations such as **AND**, **NOT** or **OR**.

Transformer: *TrCombinedFilter*

Filtering Features that are fully contained within others

Filters features based on their spatial relationship and returns only those that are entirely contained within another geometry. This can be used, for example, to select buildings within a city boundary.

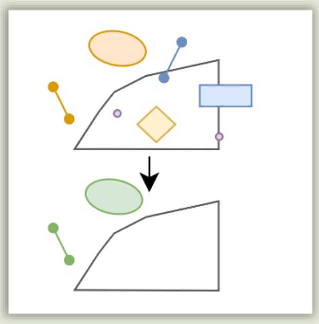
Transformer: *TrOnlyContainedFeatures*



Filtering only non-overlapping features

Allows the selection of features that do not overlap with another feature. This function is useful for identifying independent or isolated features.

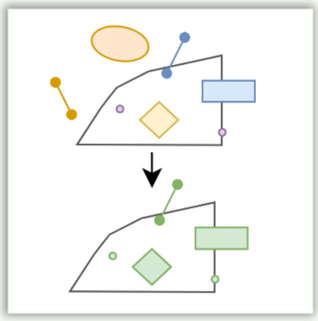
Transformer: *TrOnlyDisjointFeatures*



Filtering only intersecting features

Selects only features that intersect with another geometry.

Transformer: *TrOnlyIntersectingFeatures*



Issue Filters

Selection and filtering of issues

Subsequent IssueFilters can, if desired, be used multiple times in multiple conditions by adding them to the IssueFilter tab. Several filters can also be combined with each other in the filter expression by linking the listed filters with logical operators (**AND**, **NOT** and **OR** etc.). If the filter expression is not used, several filters are linked with **OR**.

Filtering all issues

Filters all detected issues. This can be used to completely ignore certain issue categories or temporarily hide them.

Issue Filter: *IfAll*

Filtering intersecting issues

Includes only issues that overlap with another feature. This is helpful for focusing on problematic overlaps or unexpected intersections.

Issue Filter: *IfIntersecting*

Filtering issues based on involved table rows

Filters issues where any involved row fulfills the given constraint.

Issue Filter: *IfInvolvedRows*

Filtering issues near another feature

Displays only issues that are located near another feature. This can be useful for quality control by identifying problematic spatial configurations, such as unconnected road segments.

Issue Filter: *IfNear*

Filtering issues within a specific area

Includes only issues that occur within a defined study area. This enables targeted issue checks within specific regions, such as within administrative boundaries.

Issue Filter: *IfWithin*