I2G

INTERGEO

i2g Common File Format Specification

The Intergeo Consortium

Project co-funded by the European Community under the eContentplus Programme

Contents

1 Introduction

The present document is the specification of the Intergeo File Format, Intergeo File Format v1, as of June 2010.

The Intergeo file format is a file format designed to describe any construction created with a Dynamic Geometry System (DGS). As a first application, the format can be used to interchange content between geometric software. At present, the format is restricted to the geometry in the plane, although it does not seem difficult to extend it, in the future, to the space.

The format is split into three layers: the container, the file intergeo.xml and the symbol list. The following diagram shows the structure of the layers

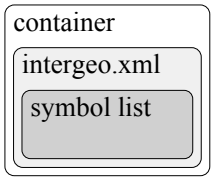

Figure 1: Specification structure in layers

- The container specification explains how all necessary files used to define a construction are bundled into a single ZIP file.
- The intergeo.xml is the core of the format and is futher divided in three parts:
  - the elements part declares all geometric objects;
  - the constraints part provides the relationships of the objects and, thus, defines the dynamic (or interactive) behaviour;
  - and the display part, which describes the styles and how the geometric objects are drawn.

  The XML schema that should validate any intergeo.xml file is partially presented with the skeleton of the elements, constraints and display part.
- The symbol list allows for additions and proprietary extensions to the format.

Dynamic Geometry Systems (DGS) is a kind of software used to experiment with geometry. A construction, a drawing with geometric elements, is displayed to the user. But the most exciting part of a DGS is that it is possible to move some of the elements with the mouse pointer and the whole construction is recomputed while keeping predefined geometric relationships,

which are the object of study. Although the origin of DGS is geometry, they can be applied to the study of other areas of mathematics or even subjects like, for example, physics.

A wide variety of DGS exists. Before this project, each system used incompatible proprietary file formats to store its data. Thus, most of the DGS makers have joined to provide a common file format that will be adopted either in the core of the systems or just as a way to interchange content.

For more information on the objectives of the Intergeo project, and proposed future additions to the i2g Common File Format, see Deliverable D3.10 at `http://svn.activemath.org/intergeo/Deliverables/WP3/D3.10/D3.10-Common-File-Format.pdf`.

2 The container

The container is the topmost structure of Intergeo file format. It is composed of a collection of files stored as a directory hierarchy in a ZIP file format.

The container comprises the construction description as well as other data, for example, media (images, sound, video, ...), previews (PDF, SVG or PNG), metadata or private data/legacy file formats that some software may keep, among other information. Thus, instead of encoding everything in a single XML file it is more natural to store all such information in a standard format like a ZIP package and, in addition, benefit of the compression.

The proposed directory structure is the following

The `construction` folder is always mandatory and contains the important file `intergeo.xml`. The specification does not enforce any preview format, however all files should be named `preview.*`. Files with names other than intergeo.xml or `preview.*` are not allowed in this part. Thus, any auxiliary file should be placed in the resources folder.

The whole construction can contain optionally metadata. We do not impose any specific format but we recommend including a file named `i2g-lom.xml` with the metadata as specified in the document [HLCMD08].

The `resources` directory contains any media or data file needed by the `intergeo.xml` or preview files. Constructions must be possible to open with solely the `intergeo.xml` and the resource files. Inside the `resources` folder it is permitted to add any hierarchy of subdirectories. However, we recommend placing the images, audio, video, data and text under the folders `images`, `audio`, `video`, `data` and `text`, respectively. The valid file formats for the resources depend on their usage. It is not the responsibility of the container specification to impose any specific format type for the media.

DGS's are free to store any file within their private directory and ignore completely the private directories belonging to other systems. Note that if a file is referred from the `intergeo.xml` it should be always placed in the `resources` directory. We strongly recommend placing all files inside a directory name based on the domain name of an organization. For example, if my organization has the domain `MyOrg.net`, the recommended directory name would be `net.myorg` (note the use of lowercase). This is done in order to distinguish my private directory from the private directories of other software developers.

construction/	mandatory
construction/intergeo.xml	mandatory
construction/preview.pdf	optional
construction/preview.svg	optional
construction/preview.png	optional
metadata/	optional
metadata/i2g-lom.xml	optional
resources/	optional
resources/<image-files>	optional
resources/<audio-files>	optional
resources/<video-files>	optional
resources/<data-files>	optional
resources/<text-files>	optional
private/	optional
private/<domain-name>/	optional
private/<domain-name>/<files>	optional

Figure 2: Containter structure

3 intergeo.xml

This is an XML file that contains the full description of the construction. The file must be always named and placed at `construction/intergeo.xml` when it appears inside the container. Except for other auxiliary media files, this file is always self contained. This means that this file alone, when it does not depend on media files, should be readable by DGS's.

There are three main distinct parts:

1. The `elements` part is a static view of all objects. The description of the objects in this part is minimal; it only indicates how the objects are constructed (their coordinates) and displayed (without styling).
2. The `constraints` part explains the geometric relations between objects. Some relations are purely algorithmic (an object can be constructed directly from existing ones) while others are constraint based (an object should satisfy a given property but, at the same time, it keeps a certain degree of freedom). Thus, the objective of this part is describing how some figures are recomputed when points (or other objects) are moved by the mouse pointer.
3. The `display` part comprises information necessary to render the objects. The display contains the styles and non-mathematical behaviour of the elements.

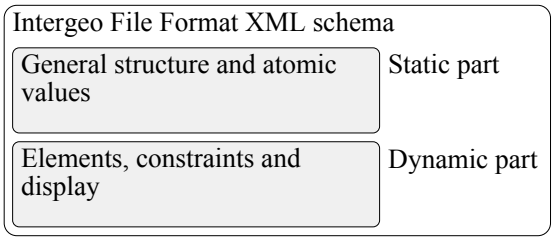

Figure 3: Intergeo file format XML schema

The splitting of data into these parts has some advantages. For example, it is possible to have more than one view or display for the same set of elements and constraints.

Because `intergeo.xml` is an XML file it is important to provide an XML schema (a file, also in XML format, that describes the structure of a family of XML files). There is not one single schema. Instead, there is one schema for each set of geometric symbols. The schema is composed of a static part which defines the general structure of the XML file and the atoms (leaves of the XML-tree). This static part is the same for all XML schemas. The dynamic part is generated automatically from the list of geometric symbols.

The current XML-schema that corresponds to the minimal set of symbols can be found at [Int09].

3.1 Construction

The construction is the root element of the intergeo.xml file. The schema fragment is the following

```
<xs:element name="construction">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="elements"/>
      <xs:element ref="constraints"/>
      <xs:element ref="display" minOccurs="0"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

and an example is

```
<construction>
  <elements>
  ...
  </elements>
  <constraints>
  ...
  </constraints>
  <display>
  ...
  </display>
</construction>
```

3.2 Elements

The <elements> element comprises the enumeration of objects that will be part of the construction. They will be often geometric objects like points, lines, circles, ... But it can hold also any object that can be drawn like images, slider bars, buttons, etc.

```
<xs:element name="elements">
  <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <!-- here goes the list of available elements -->
          <xs:element ref="point"/>
```

```
            ...
          </xs:choice>
  </xs:complexType>
</xs:element>
```

The previous schema fragment should be filled with all elements specified in the elements part of the symbols list.

The schema for each element has the form

```
<xs:element name=element–name>
  <xs:complexType>
    <xs:sequence>
      enumeration–of–arguments
    </xs:sequence>
  </xs:complexType>
  <xs:attribute name="id" type="xs:Name"
                use="required"/>
</xs:element>
```

For example, for the point with homogeneous or Euclidean coordinates:

```
<xs:element name="point">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="homogeneous_coordinates"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:Name"
                  use="required"/>
  </xs:complexType>
</xs:element>
```

And an extra definition for `homogeneous_coordinates` is needed:

```
<xs:element name="homogeneous_coordinates">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="scalar"/>
      <xs:group ref="scalar"/>
      <xs:group ref="scalar"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Note that both point and `homogeneous_coordinates` are specified with Content Dictionaries and the atomic elements scalar at the end of this chapter.

An example of the elements part is

```
<elements>
  <point id="A">
    <homogeneous_coordinates>
      <double>3.55</double>
      <double>-4</double>
      <double>0</double>
    </homogeneous_coordinates>
  </point>
</elements>
```

3.2.1 Additional elements restrictions

There are other restrictions that cannot be expressed with an XML schema and are described here.

Unique identifiers restriction. All values of the `id` attribute are used to identify the geometric objects and they must be used only once (unique identifiers).

Only constants allowed restriction. A declaration in this part is not allowed to refer to other objects. For example, each point needs coordinates and an XML coordinates element must appear as its child. Thus, scalars must be explicitly instantiated as real or complex floating numbers and cannot be replaced by variables or expressions pending to be evaluated.

Elements symbols restriction. Symbols are declared to be in the elements, the constraints or in the display part. Only symbols declared to be in the elements part can appear in the elements part.

3.3 Constraints

The <constraints> describes the relations between objects. Sometimes this relation is just a description of how an object is to be built and other times it explains how the object is constrained and partially free movable.

The schema is quite similar to the elements part:

```
<xs:element name="constraints">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <!-- here goes the list of available
           constraints -->
```

```
      <xs:element ref="line_through_two_points"/>
      ...
    </xs:choice>
  </xs:complexType>
</xs:element>
```

The list of available constraints is specified with Content Dictionaries. The schema for each constraint has the form:

```
<xs:element name=constraint-name>
  <xs:complexType>
    <xs:sequence>
      enumeration-of-arguments
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

For example, for the `line_trough_two_points` constraint the schema is:

```
<xs:element name="line_through_two_points">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="line"
                  type="output-reference"/>
      <xs:element name="point" type="reference"/>
      <xs:element name="point" type="reference"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

where `output-reference` and `reference` are defined below and are specified to act as references to objects using their id's. Example

```
<constraints>
  <line_through_two_points>
    <line out="true">l</line>
    <point>A</point>
    <point>B</point>
  </line_through_two_points>
</constraints>
```

3.3.1 Additional constraints restrictions

Defined input references restriction. All input and output references must be defined previously in the elements part.

Composition of constraints forbidden restriction. The arguments of the constraints cannot be other constraints. It is allowed only to refer to valid geometric objects, using the id's. Note that this rule still permits using constants in the arguments.

Unique output reference usage restriction. An identifier can be used only once as output reference except for some declarations. These declarations are based on symbols that explicitly declare that can coexist with other declarations with the same output reference.

Constraints symbols restriction. Symbols are declared to be in the elements, the constraints or in the display part. Only symbols declared to be in the constraints part can appear in the constraints part.

3.4 Display

The display part contains the styles and non-mathematical behaviour of the elements and the whole construction area.

Styling consists on specifying visual properties (like colours, line widths, point sizes, labels, etc.) and the interactive properties (fixed) of the element. While the elements and constraints parts are close to mathematical properties and behaviour, the display part contains visual properties and the behaviour that can not be formalized mathematically.

The current version of the styling system for Intergeo is inspired by the SVG and CSS standards. Thus, some symbols like *fill*, *stroke*, *stroke-width* are taken from the SVG and CSS standards but *point-size*, *label* and *fixed* are specific of the Intergeo File Format.

Each system will provide default values for the styles which may depend on the type of the element: points might be blue while lines might be red. Movable points might have a color different to fixed points. The point which currently has the mouse focus and is being moved might be distinguished with particular combinations of colors, borders, opacities, etc.

More than one display can be present and each one represents a possible view of the representation. A DGS could open two windows and changes in one would reflect in the other; each display would describe what is to be seen in each window.

Styles for elements will be specified individually in each display part. For instance, the same point could be movable in one window but not in another.

Each display can contain also styles applying to the whole construction, as opposed to one particular element; for instance, background-color, viewport, or axes.

The display contains a <style> for each element to be described.

```
<xs:element name="display">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="style"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The style must refer to an existing element and comprises the different possible styles of the element.

```
<xs:element name="style">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="label"/>
      <xs:element ref="fill"/>
      ...
    </xs:choice>
    <xs:attribute name="ref" type="xs:Name" use="required"/>
  </xs:complexType>
</xs:element>
```

For example, the XML-Schema for <label> is:

```
<xs:element name="label" type="xs:string"/>
```

A complete example of a point with styles is

```
<construction>
  <elements>
    <point id="P">
      <homogeneous_coordinates>
        <double>4</double>
        <double>5</double>
        <double>1</double>
      </homogeneous_coordinates>
    </point>
  </elements>
  <constraints>
    <!— —>
  </constraints>
  <display>
    <style ref="P">
      <label>The point P</label>
      <fill>#FFFFFF</fill>
      <stroke>#222222</stroke>
```

```
        <stroke_width>2</stroke_width>
        <point_size>5</point_size>
      </style>
    </display>
  </construction>
```

3.4.1 Additional restrictions

Defined reference restriction The reference must point to an existing element described in the elements part.

Single style node by element restriction All styles of an element must be specified in a single <style>.

3.5 Atomic values

Atomic values are the simplest ingredients of the Intergeo file format. They are specified also with an appropriate schema fragment.

Note: We might add more atoms in the future.

3.5.1 Scalars

Scalars represent either double or complex double numbers:

```
<xs:group name="scalar">
  <xs:choice>
    <xs:element ref="double"/>
    <xs:element ref="complex"/>
  </xs:choice>
</xs:group>
```

3.5.2 Double numbers

Doubles follow the [IEEE 754-1985] standard.

```
<xs:element name="double" type="xs:double"/>
```

Example

```
<double>4.37589837073</double>
```

3.5.3 Complex numbers

```
<xs:element name="complex">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="double" type="xs:double"/>
      <xs:element name="double" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Example

```
<complex>
  <double>2.4689</double>
  <double>−5.78231659</double>
</complex>
```

3.5.4 References to objects

Constraints accept as arguments references to objects.

```
<xs:complexType name="reference">
  <xs:simpleContent>
    <xs:extension base="xs:string"/>
  </xs:simpleContent>
</xs:complexType>
```

3.5.5 Output references to objects

Constraints usually have one argument that is the output. The output is the name of the object that is going to be computed from the other ones.

```
<xs:complexType name="output−reference">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="out" use="required"
                    fixed="true"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

3.6 intergeo.xml XML-Schema

As we have already said, the XML-schema is generated from the set of symbols and its signatures. The project that generates such schema can be get from the SVN repository `http://svn.activemath.org/intergeo/Format/xml`. Several examples of valid files can be found at `http://i2geo.net/xwiki/bin/view/I2GFormat/FileFormatSymbols`.

Although the original idea was using the OpenMath Content Dictionaries and their associated Small Type System files, we decided to write a very simple and single XML file with all symbols and generate the XML-Schema from it. Such XML file can be found at `http://svn.activemath.org/intergeo/Format/xml/symbols.xml`.

The symbols.xml is an XML file that contains essentially declarations like, for example,

```
<element name="point">
  <argument type="homogeneous_coordinates"/>
</element>
```

for declaring elements and

```
<constraint name="line_through_two_points">
  <argument type="line" out="true"/>
  <argument type="point"/>
  <argument type="point"/>
</constraint>
```

for the constraints.

The declarations in the symbols.xml file are much simpler than the corresponding ones at the XML-schema. Thus, it is possible to add more symbols to the schema without any advanced knowledge of XML. Another advantage of this approach is the possibility of generating different schemas addressing different purposes. For example, generate a schema that validates not only the official symbols of the format, but the proprietary ones. Another possibility is getting a schema to validate the file format for constrained based systems (as opposed to construction based which mainly targets the Intergeo File Format).

4 Symbol list

The symbol list is a collection of OpenMath Content Dictionaries describing symbols; in particular, some symbols have been chosen to be part of a DGS construction.

The symbol list layer will be used to allow for each DGS to create its own extensions to i2g, by possibly adding new Content Dictionaries in the container layer that can be read and handled in system dependent ways.

There will be a process for Content Dictionaries to become an official part of the file format. How this process will be structured and how new official Content Dictionaries can be added after the Intergeo project's lifetime is to be decided.

The following is a list of symbols that we agreed on as of June 2010. It will be developed further by the Intergeo A.s.b.l.

At any time, HTML documentation on the newest version can be found at `http://i2geo.net/xwiki/bin/view/I2GFormat/FileFormatSymbols`; even though not accessible to everybody, a second source, more convenient to download files, is the SVN server at `http://svn.activemath.org/intergeo/Format/` .

In the following list, the keyword is listed first, in boldface. On the next line, the types of its arguments are listed. The order of the arguments is fixed. Optional arguments are enclosed between [and] characters. Arguments that can be repeated any number of times are indicated with a *.

Observe that the word "undefined" is used with, at least, two different meanings. When the result of a computation is said to be undefined, it is meant a DGS dependent value that indicates error or voidness. When, under some circumstances, it is said that the behaviour of some operation is undefined, what is meant is that each DGS is free to provide its most sensible result; for example, two circles may or may not have intersections depending on whether the DGS handles complex coordinates.

4.1 The elements part

point
point_coordinates
This represents a point in space. It will have coordinates for initialization.

line
homogeneous_coordinates
This represents a line. The coordinates are the homogeneous coordinates of the line; this is, $(2, 3, 5)$ represents the line $2x + 3y + 5 = 0$.

line_segment
point_coordinates, point_coordinates [,point_coordinates]
This represents a segment from a straight line; the name is due in consideration to possible future implementations of other kinds of segments. A line_segment is indicated by providing the coordinates of its two endpoints; additionally, a third optional point, called *via point*, may be used in projective geometry to indicate a point in the middle of the segment to specify which of the two possible segments is meant. For instance, line_segment $([0,0],[0,2],[0,1])$ would represent the “normal" segment from $(0,0)$ to $(0,2)$, while line_segment $([0,0],[0,2],[0,10])$ would represent the projective segment that starts at $(0,0)$, goes to $(-\infty,0)$, and then continues from $(\infty,0)$ to $(0,2)$.

directed_line_segment
point_coordinates, point_coordinates [,point_coordinates]
A line_segment with associated direction; instead of two endpoints it has a starting point and an endpoint. The same remarks for the via point apply as in the case of **line_segment**.

ray
point_coordinates, direction | point_coordinates, point_coordinates
This represents a ray (half line) which starts at the given point and proceeds to infinity in the direction provided, or via a second point (alternative option). Projective DGS may treat it as a special form of a **directed_line_segment**. If the given coordinates are at infinity the notion or ray does not make sense, but a DGS can choose to use the line at infinity instead.

polygon
list_of_vertices_coordinates
A polygon is constructed from a list of vertices. No restrictions are imposed on the vertices, so that all degenerate cases are possible: two consecutive vertices can be the same, three consecutive vertices can be collinear, sides can intersect, the border does not have to be a simple curve.

vector
point_coordinates
Constructs a vector going from the origin to the point with the specified coordinates.

conic
matrix [,dualmatrix]
This will represent a general conic. The coefficients of its associated quadratic form are provided in a 3×3 matrix of (possibly complex) numbers. An additional 3×3 matrix may be given, being the matrix of the dual conic. This can assist if the conic is a degenerate case (two lines, etc)

The following symbols are special kinds of conics. Their initial position is stored in the same format as a general conic. A difference with the conic symbol is that a DGS may opt to only accept reading a circle and request

the intergeo java library to convert the matrix to a different form, whereas it might reject a general conic.

circle
matrix [,dualmatrix]
This will represent a circle, indicated in the same way as general conics.

ellipse
matrix [,dualmatrix]
This will represent an ellipse, indicated in the same way as general conics.

parabola
matrix [,dualmatrix]
This will represent a parabola, indicated in the same way as general conics.

hyperbola
matrix [,dualmatrix]
This will represent a hyperbola, indicated in the same way as general conics.

locus
void | a set of points
This will represent a locus. Its initial value can either not be provided (it will then have to calculated by the DGS) or a finite set of tracer points (specified together with the corresponding values of the mover point).

4.1.1 Families

Four families have been defined. They are not elements, but sets of elements. They are used to specify the arguments that some constraints can have, and so they provide constraints with a form of polymorphism.

- line_family = {line, ray, line_segment, directed_line_segment}
- circle_family = {circle, arc}
- conic_family = {circle, circle arc, conic, parabola, ellipse, hyperbola}
- element_family = {all elements}

4.1.2 Auxiliary symbols

point_coordinates
homogeneous_coordinates | euclidean_coordinates | polar_coordinates
The coordinates of a point, or the components of a vector, can be specified using any of the following three symbols:

homogeneous_coordinates
scalar, scalar, scalar

Provides the coordinates of a point in projective geometry; unlike the following two symbols, this one can be used also to provide for the coefficients of a line equation, see the symbol line above.

euclidean_coordinates
scalar, scalar
Provides the Euclidean coordinates of a point in the plane.

polar_coordinates
scalar, scalar
Provides the radius and argument of a point in the plane.

direction
scalar, scalar | *scalar, scalar, scalar*
Directions are auxiliary symbols, used to construct rays. They are specified either euclideanly as (a, b) or projectively (homogeneously) as (a, b, c).

list_of_vertices_coordinates
*point**
This auxiliary symbol is used only by the polygon element.

list_of_vertices
*point_reference**
This is not exactly an auxiliary symbol, but rather an auxiliary XML element (tag) used only within constraint polygon_by_vertices. It plays the same role as list_of_vertices_coordinates, only for constraints instead of elements, and it contains the references to the vertices as opposed to their coordinates.

4.2 The construction part

The *new* or *output* arguments (that is, the dependent element of the construction) are indicated with the word “new". The first argument is always a new argument. The arguments which are not new are identifiers of existing objects upon which the new arguments depend. These objects must be declared in the elements part of the file.

Remark: Whenever distances or angles are mentioned in this document we refer to Euclidean measurements, unless otherwise stated.

free_point
new point
A completely unconstrained point.

free_line
new line
A completely unconstrained line. Note that the user interface of many DGS does not allow for controlling lines directly, but rather two points are moved to control their common line; hence, many DGS cannot implement this constraint.

point_on_line
new point, line_family
A new point restricted to lie on a given line.

point_on_line_segment
new point, line_segment
A new point restricted to lie on a given line_segment.

point_on_circle
new point, circle_family
A new point restricted to lie on a given circle.

line_through_point
new line, point
A new line that goes through a given point and can be rotated by the user, by moving the mouse. Note that the user interface of many DGS does not allow directly for this operation, but rather an intermediate point which can be moved can be used to construct the line which passes through both points; hence, many DGS cannot implement directly this constraint.

line_through_two_points
new line, point, point
A new line that goes through two given points. If the two points are equal, the line is undefined.

line_angular_bisector_of_three_points
new line, point, point, point
A new line that is the angular bisector of three given points (in Euclidean measurements); it must pass through the second one. Degenerate cases: If the three points are the same, behaviour is undefined. If the first and third point are the same, then the line returned must be the line containing three points. If the second point is equal to the first or third, then the behaviour is undefined.

line_angular_bisectors_of_two_lines
new line, new line, line_family, line_family
Two lines that are the angular bisectors of the two given lines (in Euclidean measurements). If the lines are coincident, one bisector will be the common line and the other will be undefined. If they are parallel and not coincident, then one bisector is the line at infinity (for those DGS that support projective geometry) and the other bisector is undefined. A DGS can choose to use the line with the same Euclidean distance to both defining lines as the other bisector, as it is the continuous completion.

line_segment_by_points
new line_segment, point, point [,point]
A new line_segment passing through the first two points and, optionally, for those DGS that can handle projective geometry, also through the third. If the third point is equal to the first or the second, then the behaviour is undefined

and each DGS can return an undefined value or what it thinks is the best line_segment.

directed_line_segment_by_points
new line_segment, point, point [,point]
A new directed_line_segment passing through the first two points and, optionally, for those DGS that can handle projective geometry, also through the third. If the third point is equal to the first or the second, then the behaviour is undefined and each DGS can return an undefined value or what it thinks is the best line_segment.

ray_from_point_and_vector
new ray, point, vector
A new ray that starts at the point and moves in the direction specified by the vector.

ray_from_point_through_point
new ray, point, point
A new ray that starts at the first point and goes through the second to infinity.

line_parallel_to_line_through_point
new line, line_family, point
A new line that is parallel to a given line and goes through a given point.

line_perpendicular_to_line_through_point
new line, line_family, point
A new line that is perpendicular to a given line and goes through a given point.

point_intersection_of_two_lines
new point, line_family, line_family
Elements in the line_family can be extended to lines; see Appendix **??**. A new point that is the intersection point of two given lines. In the degenerate case the point is undefined, or a DGS may use an continuous completion.

midpoint_of_two_points
new point, point, point
A new point that is the middle point (in Euclidean measurements) between two given points.

midpoint_of_line_segment
new point, line_segment
A new point that is the midpoint of a given line segment.

endpoints_of_line_segment
new point, new point, line_segment
A set of two points that form the end points of the given line_segment.

carrying_line_of_line_segment
new line, line_segment
The unique line that contains the given line_segment. Degenerate cases: if

the segment is constructed to be always a point, the carrying line is undefined.

starting_point_of_directed_line_segment
new point, directed_line_segment
The starting point of a directed_line_segment.

end_point_of_directed_line_segment
new point, directed_line_segment
The end point of a directed_line_segment.

line_segment_of_directed_line_segment
new line_segment, directed_line_segment
The unique line_segment whose endpoints are the starting point and the end point of the given directed_line_segment.

vector_of_ray
new vector, ray
Returns the vector that indicates the direction of a ray; note that it is not uniquely defined.

starting_point_of_ray
new point, ray
The new point is the starting point of a ray.

carrying_line_of_ray
new line, ray
The new line is the unique line that contains the ray.

circle_by_center_and_radius
new circle, point, line_segment
A new circle with a given point as center and a radius as long as the given line_segment.

circle_by_center_and_point
new circle, point, point
A circle whose center is the first point and passes through the second.

circle_by_three_points
new circle, point, point, point
A circle that passes through the three provided points. Degenerate cases: if the three points are by construction collinear (but not equal), a DGS may give an error or return the line through the points (as a line or a degenerate conic). If the three points are by construction the same, a DGS may give an error or return the point (as a point or a degenerate conic).

intersection_points_of_two_circles
new point, new point, circle_family, circle_family
The intersection points of two circles. Elements in the circle_family can be extended to circles; see Appendix **??**. If there are fewer than two intersection points, a DGS may return the same point twice (in case of tangency),

“undefined" or points with complex coordinates. If the circles are equal by construction, a DGS may give an error or return the common circle.

other_intersection_point_of_two_circles
new point, point, circle_family, circle_family
The new point is an intersection point of the two circles, and is different to the provided point. Elements in the circle_family can be extended to circles; see Appendix **??**.

intersection_points_of_circle_and_line
new point, new point, circle_family, line_family
The intersection points of a circle and a line. If there are less than two intersection points, each DGS is free to return repeated points, undefined values, or points with complex coordinates. Elements in the line_family and circle_family can be extended to lines and circles; see Appendix **??**.

other_intersection_point_of_circle_and_line
new point, point, circle_family, line_family
The new point is the intersection point of the circle and the line, and is different to the provided point. Elements in the line_family and circle_family can be extended to lines and circles; see Appendix **??**.

intersection_points_of_two_conics
new point, new point, new point, new point, conic_family, conic_family
The up to four intersection points between two conics. If there are less than four real intersection points, each DGS is free to return repeated points, undefined values, or points with complex coordinates. However, the algebraic multiplicity of the intersections has to be respected. If any of the elements intersected is an arch, it can be extended into a circle; see Appendix **??**.

intersection_points_of_conic_and_line
new point, new point, conic_family, line_family
The up to two intersection points between a conic and a line. If there are less than two intersection points, each DGS is free to return repeated points, undefined values, or points with complex coordinates. If the conic was degenerate and contained a line coincident with the intersecting line, a DGS may return “undefined" or the intersection line. If the element in the conic_family is an arc, it can be extended into a circle; and the element in the line_family can be extended into a line, see Appendix **??**.

other_intersection_point_of_conic_and_line
new point, point, conic_family, line_family
Returns the intersection point of a conic and line which is different to a provided point. The behaviour of this function must be consistent with that of intersection_points_of_conic_and_line. If the element in the conic_family is an arc, it can be extended into a circle; and the element in the line_family can be extended into a line, see Appendix **??**.

circle_tangent_lines_by_point
new line, new line, circle_family, point

The two lines which are tangent to a circle and pass through a point. If the point is on the circle and so there is only one tangent, a DGS may return the tangent (repeated), or the tangent and “undefined", or just “undefined". If the point was inside the circle, one or two times “undefined" may be returned. If the circle has radius zero and the point coincides with the center, two times “undefined" must be returned.

foci_of_conic
new point, new point, conic_family
The new points are the two foci of a conic. If the conic is degenerate and has only one focus, a DGS may return the focus (repeated), or the focus and an undefined value. One could also give circles, parabolas, ellipses and hyperbolas instead of conics, the former being subtypes of the latter.

center_of_circle
new point, circle_family
The new point is the center of the given circle (or arc).

locus_defined_by_point_on_line
new locus, point, point, line
Defines a locus as the trace of the first point when the second point moves over the line.

locus_defined_by_point_on_line_segment
new locus, point, point, line_segment
Defines a locus as the trace of the first point when the second point moves over the line_segment.

locus_defined_by_point_on_circle
new locus, point, point, circle
Defines a locus as the trace of the first point when the second point moves over the circle.

locus_defined_by_point_on_locus
new locus, point, point, locus
Defines a locus as the trace of the first point when the second point moves over the locus.

locus_defined_by_line_through_point
new locus, point, line, point
Defines a locus as the trace of the first point when the line moves around the second point.

symmetry_by_point
new element_family, element_family, point
Reflects an element about a point (central symmetry).

symmetry_by_line
new element_family, element_family, line
Reflects an element about a line (axial symmetry).

symmetry_by_circle
new element_family, element_family, circle
Reflects an element about a circle (circle inversion).

translate
new element_family, element_family, vector
Translates an element as indicated by the vector.

4.3 The display part

Since the display part does not add *mathematical* information to the resource, this part will not have an OpenMath equivalent.

Most default values of visual aspects are system dependent, as explained in Subsection **??**.

stroke
Stroke is used to specify whether a path has to be drawn and with which color. Its default value is system dependent and, in particular, may depend on the type of element being painted. The value has to be given as a **paint**.

stroke-width
Stroke-width specifies the stroke width of the path. The value has to be given as a **length**. See also borderwidth.

stroke-dasharray
Can be either “none", the default value that indicates that the element is to be painted with solid strokes, or a list of lengths specifying in pixels the lengths of alternating dashes and gaps. Each length is a positive real number. If an odd number of lengths is provided, then the list of values is repeated to yield an even number of values. Thus, the length list 5,3,2 is equivalent to 5,3,2,5,3,2. By default, strokes are solid.

borderwidth
A real number, measured in pixels. Applies to elements that have a border, and so its applicability may be system dependent. It is used to indicate the width of the region painted with the “stroke" color as opposed to the “fill" color. The value has to be given as a **length**. See also stroke-width; as an example to illustrate the difference, some DGS allow lines to have borders, and so stroke-width would be the with of the line and borderwidth would be the width of the line border.

border-opacity
A number between 0 and 1, default value 1, indicating how opaque the border must be. Default value is 1.

fill
Fill is used to specify whether the interior of an element (such as a polygon) has to be filled and with which color. The value has to be given as a **paint**.

fill-opacity
This provides a number between 0 and 1 indicating how opaque is the interior of an element (such as a polygon).

point-size
Point-size specifies the size of the points. Points are usually displayed as circles with diameter equal to the value of this style. The value has to be given as a **length**.

point-style
A string indicating the shape of points; such as “circle", “star", etc. No particular values have been specified yet.

label
Label is the text that is displayed as the label of an element. This should be viewed as the visible name of the object, handy for the user but not necessarily unique, as opposed to the identifier name, the unique identifier the software uses. Possible values are plain text (anything valid as XML text). The current format does not allow any mathematical specific format like TEX/LATEXor MathML.

visible
Indicates whether the element is visible (default) or must be hidden, maybe because it is just an uninteresting mid step in a construction. Possible values are “true" or “false".

fixed
Fixed specifies whether a given graphical object can not be moved by the user. Possible values are “true" and “false".

background-color
It specifies the background color of the drawing area. This is not a real style, since it does not affect a particular element.

4.3.1 Auxiliary symbols

length
Length specifies a length. Lengths are decimal numbers (e.g. 12.345) that represent pixels. For simplicity, units are not allowed. Note that a given length measures the same independently of the scale of a construction.

paint
Paint can be none or a **color**.

color
Color has the form #RRGGBB where RR, GG and BB stands for the 0–255 hexadecimal values of the red, green and blue component of the color.

5 How to Identify Objects of the I2G Format

This section specifies ways of naming files that should be recognized as I2G files in the common file format. This part is normative, developers should follow them strictly so as to ensure that interoperability can be started.

Format name: Intergeo

Media type name: `application`

Media subtype name: `vnd.intergeo`

Encoding: binary
(This media type may require encoding on transports not capable of handling binary.)

Security considerations: There has been no examination of possible security risks associated with I2G files.

Interoperability considerations: This file format is cross-platform. Files are encoded in UTF-8 and compressed using zip.

Published specification: this specification (Inter2geo's *Deliverable D3.10: i2g Common File Format Final Version*).

Applications which use this media : Cinderella, GeoGebra, GEONExT, JSXGraph, Cabri, WIRIS,...

Additional information:
1. Magic number(s) : NOT USED
2. File extension(s) : `i2g`
3. Apple Macintosh file type code : NOT USED
4. Object Identifiers: NOT USED
5. MicroSoft Windows Clipboard Names: `I2G`
6. Apple Macintosh Uniform Type Identifier: `eu.inter2geo` extends `public.archive` and `public.zip-archive`

Person to contact for further information :
1. Name : Intergeo A.s.b.l.
2. Email : asbl@inter2geo.eu

Intended usage: Common
This mime type shall be used to identify data files for the common file format for interactive geometry software.

Author/Change controller: Intergeo A.s.b.l. (asbl@inter2geo.eu)

Procedures taken to ensure such names A media type registration has been filed to the Internet Assigned Numbers Authority for the media-type Intergeo and granted from them as can be seen at: `http://www.iana.org/assignments/media-types/application/vnd.intergeo`.

The text above is based on this registration and extends it. It is the intention of the consortium to update that registration to match the above text.

This text contains recommendations for implementors of desktop applications to run on MicroSoft Windows and Apple Macintosh operating systems. Conforming to Apple's documentation (`http://developer.apple.com/mac/library/documentation/FileManagement/Conceptual/understanding_utis/`) about Uniform Type Identifiers, it is enough for applications to declare their support for this file format within their application descriptor. Conforming to MicroSoft's documentation (`http://msdn2.microsoft.com/en-us/library/ms649013.aspx`), it is enough for applications to call the appropriate functions at startup.

References

[CIM08] David Carlisle, Patrick Ion, and Robert Miner. Mathematical markup language (mathML) version 3.0. World Wide Web Consortium, Working Draft WD-MathML3-20080409, April 2008.

[Cre08] Creative Commons Inc. (CC). Namensnennung-Weitergabe unter gleichen Bedingungen 2.0 Deutschland. Available on the web, May 2008.

[HKKM09] Maxim Hendriks, Ulrich Kortenkamp, Yves Kreis, and Daniel Marquès. i2g common file format draft v2. http://svn.activemath.org/intergeo/Deliverables/WP3/D3.6/D3.6-Common-File-Format-v2.pdf, July 2009.

[HLCMD08] Maxim Hendriks, Paul Libbrecht, Albert Creus-Mir, and Michael Dietrich. Metadata specification. http://svn.activemath.org/intergeo/Deliverables/WP2/D2.4-Metadata/D2.4-Metadata-Spec.pdf, June 2008.

[Int09] Intergeo. Intergeo file format xml-schema for intergeo.xml. http://svn.activemath.org/intergeo/Drafts/Format/xml/intergeo.xsd, 2009.

[KCP08] KCP Technologies Inc. Geometer's sketchpad v4, 2008.

[Knu84] Donald E. Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1984.

[Kor99] Ulrich Kortenkamp. *Foundations of Dynamic Geometry*. Dissertation, ETH Zürich, Institut für Theoretische Informatik, Zürich, 11 1999.

[Kor09] Ulrich Kortenkamp. I2g api specification. Deliverable D3.5, The Intergeo Consortium, 2009.

[MP02] David Marcheix and Guy Pierra. A survey of the persistent naming problem. In *SMA '02: Proceedings of the seventh ACM symposium on Solid modeling and applications*, pages 13–22, New York, NY, USA, 2002. ACM.

[Sal08] Saltire Software. Geometry expressions v1.1, 2008.