

# I2G

## INTERGEO

**Deliverable N°: D3.6**

**i2g Common File Format Draft v2**

**The Intergeo Consortium**

**July 2009**

**Version:** version of August 3, 2009 3:27 P.M.

**Main Authors:**

Maxim Hendriks (Technische Universiteit Eindhoven)  
Ulrich Kortenkamp (University of Education Schwäbisch  
Gmünd & Cinderella)  
Yves Kreis (Université du Luxembourg & GeoGebra)  
Daniel Marquès (Maths for More & WIRIS)



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

<b>Project ref.no.</b>	ECP 2006 EDU 410016
<b>Project title</b>	Intergeo - Interoperable Interactive Geometry for Europe

<b>Deliverable status</b>	submitted
<b>Contractual date of delivery</b>	M22 – July 2009
<b>Actual date of delivery</b>	July 31 <sup>th</sup> 2009
<b>Deliverable title</b>	i2g Common File Format Draft v2
<b>Type</b>	Presentation
<b>Status &amp; version</b>	submitted version of August 3, 2009 3:27 P.M.
<b>Number of pages</b>	40
<b>WP contributing to the deliverable</b>	WP3
<b>WP/Task responsible</b>	Daniel Marquès
<b>Authors</b>	René Grothmann (Katholische Universität Eichstätt-Ingolstadt & C.a.R.), Maxim Hendriks (Technische Universiteit Eindhoven), Markus Hohenwarter (Florida Atlantic University & GeoGebra), Ulrich Kortenkamp (University of Education Schwäbisch Gmünd & Cinderella), Yves Kreis (Université du Luxembourg & GeoGebra), Jean-Marie Laborde (Cabrilog & Cabri), Paul Libbrecht (DFKI GmbH), Daniel Marquès (Maths for More & WIRIS), Ingo Schandeler (Université du Luxembourg & GeoGebra)
<b>EC Project Officer</b>	Spyridon Pilos
<b>Keywords</b>	dynamic geometric system, file format, open-math, standard
<b>License</b>	This document is available under the license <i>Creative Commons Attributions Sharealike Germany 2.5</i> [Cre08]

## Contents

<b>1</b>	<b>Executive Summary</b>	<b>5</b>
<b>2</b>	<b>Introduction</b>	<b>7</b>
2.1	The three layers of the file format specification . . . . .	8
2.2	Design decisions: Constraints versus Constructions . . . . .	9
2.3	Design Decisions: OpenMath . . . . .	11
<b>3</b>	<b>The container</b>	<b>12</b>
<b>4</b>	<b>intergeo.xml</b>	<b>14</b>
4.1	Construction . . . . .	15
4.2	Elements . . . . .	15
4.2.1	Additional elements restrictions . . . . .	17
4.3	Constraints . . . . .	17
4.3.1	Additional constraints restrictions . . . . .	18
4.4	Display . . . . .	19
4.4.1	Additional restrictions . . . . .	21
4.5	Atomic values . . . . .	21
4.5.1	Scalars . . . . .	21
4.5.2	Double numbers . . . . .	21
4.5.3	Complex numbers . . . . .	21
4.5.4	References to objects . . . . .	22
4.5.5	Output references to objects . . . . .	22
4.6	intergeo.xml XML-Schema . . . . .	23
<b>5</b>	<b>Symbol list</b>	<b>24</b>
<b>6</b>	<b>First (partial) implementations</b>	<b>25</b>
6.1	Saving . . . . .	25
6.2	Opening . . . . .	25
6.3	Implementation . . . . .	26
6.4	Testing . . . . .	26

<b>7 Future Work and Request For Comments</b>	<b>27</b>
7.1 More Elements and Polymorphism . . . . .	27
7.2 Circle Coordinates . . . . .	27
7.3 Ambiguity Resolving . . . . .	28
7.4 Styling Information . . . . .	28
7.5 Functions and Scripting . . . . .	29
7.6 Mathematical Typesetting . . . . .	29
7.7 Sending Commands to Constructions . . . . .	29
7.8 Macro Constructions . . . . .	30
7.9 Number Representation . . . . .	30
7.10 Acyclicity of Construction Dependencies . . . . .	30
7.11 Library Support . . . . .	30
7.12 Higher Dimensional Geometry . . . . .	31
<b>Appendix A : Intergeo file format OpenMath symbols</b>	<b>32</b>
A.1 The elements part . . . . .	32
A.1.1 Auxiliary symbols . . . . .	33
A.2 The construction part . . . . .	34
A.3 The display part . . . . .	39
A.3.1 Auxiliary symbols . . . . .	39
<b>Bibliography</b>	<b>40</b>

## 1 Executive Summary

The present document is the specification of the second version of the Intergeo File Format Draft, as of July 2009. Once the specification is finally released, by the end of the project, it will be renamed to Intergeo File Format v1.

The specification is the result of intensive collaboration between Dynamic Geometric System (DGS) software developers and experts. At the time of this writing, the format has not been completely specified and several partial implementations already exist; therefore, it is very likely that important modifications will arise.

The Intergeo file format aims at the interchange of content between DGS's. It is our goal to create a file format that could serve as a standard in the DGS industry.

The introduction chapter justifies the objectives, the motivation and the expected success of the Intergeo file format. It is specially important to note that among the authors of the Intergeo file format are some of the leading DGS's manufacturers. In the introduction it is also explained how the format is split into three specifications: the container, the file `intergeo.xml` and the symbol list.

The container specification explains how all necessary files used to define a construction are bundled into a single ZIP file. It also contains the important file `intergeo.xml`.

The `intergeo.xml` file is the core of the format and is explained in detail. It comprises three differentiated 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. The display part describes the styles and how the geometric objects are drawn. The XML schema that should validate any `intergeo.xml` file is presented with the skeleton of the elements, constraints and display part. The children of the elements, constraints and display part can be specified with their respective XML schema fragments. However, these fragments are not written in this document because they can be generated dynamically from the list of symbols. Thus, different templates are introduced according to the elements, constraints and display parts. Some additional restrictions are described without using XML schema.

The separation of the possible geometric elements and constraints from the file format implementation is done through the so called list of symbols which is a collection of OpenMath symbols classified in Content Dictionaries. Some generalities about the list of symbols are presented. The complete list of symbols is included in Appendix A. Such a list is important because choosing correctly its content is crucial to achieve the successful interoperability between all DGS's.

**Differences with the previous version** The main improvement is the amount of new symbols introduced in this version. The previous version only contained symbols concerning linear geometry. The current coverage is the common features that all DGS must share: linear geometry, conics and locus. A proposed list of symbols for the display part is also new in this version. The core of the format is the XML file intergeo.xml and an XML-Schema is introduced. The implementation of the File Format by the different DGS has also been evolving. But, it is not until deliverable 3.8 (Report on Implementations of File Formats) when we will provide full details.

## 2 Introduction

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.

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.

The file format proposed in this document aims to be the convergence of the common features of the current DGS together with the vision of future developments and the opinion of external experts. In addition, we state the following objectives:

- The resulting file format will be adopted by the manufacturers that are authors of this specification.
- The files that satisfy this specification should be interchangeable between the different DGS.
- The format will be based on modern technologies.
- The format will be extensible. In particular, this will allow capturing the flavour of the different DGS.

Providing a file format for DGS is a task related, but not equal, to express geometric constructions in general. However, some additional issues arise. For example, styling (colours, point sizes, etc.) and the capability of moving the objects with the mouse (interactivity) are very important for a DGS while they have not sense at all in geometry as a theoretical mathematical discipline. We conclude remarking that traditional techniques used to describe a geometry figure are not always suitable for a DGS.

## 2.1 The three layers of the file format specification

The Intergeo file format specification is split into three layers. Each layer is different in nature and can exist by itself.

The following diagram shows the structure of the layers

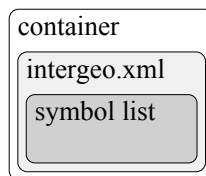


Figure 1: Specification structure in layers

**The container layer specification** describes the container as a ZIP file. This means that the container is a bundle of compressed files. It contains the important file `intergeo.xml` and other files like media (images, audio, video), data, text and a preview of the construction.

**The `intergeo.xml` layer specification** describes which rules must follow an XML file to be a valid DGS construction. Such a file is self-contained except for media files that are located in the container. An XML-schema is provided to help specifying and validating any construction. This layer specifies the general structure of the file, how the different statements are constructed from the symbols and how the most atomic values like (real or complex) numbers, variables, text and formulas have to be encoded. However, the list of symbols that is the origin of the statements is not part of this layer.

**The symbol list layer** is an OpenMath collection of Content Dictionaries. OpenMath is a standard used to express mathematical content through a series of functions, relations and constants called symbols which are specified with Content Dictionaries (CD). Each CD is an XML file that collects the description of a coherent set of symbols. This part of the specification is an exhaustive enumeration of valid geometric elements, constraints and styles that can be used to generate a construction. Although the symbols have been chosen to be part of a DGS construction, actually, they could live independently and be used for other purposes. Another difference with the `intergeo.xml` specification is that DGS's are free to create their own extensions (at the expense of losing interoperability with other systems).



## 2.2 Design decisions: Constraints versus Constructions

The general framework was clear from the outset: to design a semantically rich format, that could be interpreted by at least all DGS in the consortium. But also possibly by others, and maybe by other types of programs as well, for example computer algebra systems or proof assistants. One main design decision in this respect consists of the choice of constructions, as opposed to constraints. We will describe this in more detail now.

DGS deal with sets of geometrical objects that have certain relations. We call such a set of objects with given relations a *figure*. We conceive of these objects in some underlying space, for example the Euclidean plane. In principle, if nothing else is said about them, objects can move around freely in this space. The relations then specify *constraints* on the movement of these objects.

**Example 1** *Two points  $P$  and  $Q$ , together with a line  $l$ ; there is the following constraints:*

*line  $l$  is incident to both points  $P$  and  $Q$ .*

**Example 2** *A circle  $\Gamma$ , a point  $P$ , a line  $l$  and the constraints*

*$P$  is on  $l$   
 $l$  is tangent to  $\Gamma$   
the distance of  $P$  to the center of  $\Gamma$  is 10.*

We can make the simple observation that the constraints do not determine the positions of the objects uniquely. This causes multiple problems that lie at the heart of dynamic geometry.

First there is the problem of how to create an instance of the figure. (We say this has to do with the *static* aspect of the figure.) In example 1, the points  $P$  and  $Q$  could still lie anywhere on the line  $l$  as it stands. For any instance, we must specify where  $l$ ,  $P$  and  $Q$  should be. But once we have specified one, the other two are not completely free anymore. This particular example is not hard. Example 2, although more difficult, is still doable. But in general, it is very difficult to give any particular solution for a set of constraints. There is not even a quick method to decide whether there are any instances: a set of constraints could be too restrictive and leave none.

Second, there is the dynamic behaviour of a figure, caused by the freedom still left by the constraints. In example 1, what should the user be able to move? May the line be picked up and translated or rotated in its entirety, the points being translated and rotated with it? Can the user only move one of the two points, the line being adjusted accordingly? Constraints of a

strictly classical geometrical nature, such as the ones stated above, do not say anything about this behaviour. For the approach of a DGS, this is not enough.

A natural way to shed light on both these problems is a more precise specification of how the objects depend on each other. We could stipulate first of all which objects are *free*, meaning that they can be varied over the whole range of possibilities in the underlying space (think of the plane) by the user. We would then proceed step by step saying which objects depend only on the free objects, which ones depend only on these new objects and the free objects, etcetera. Such a specification is called a *construction*. It allows for an algorithm to rapidly create instances or decide that there are none. It also enables a DGS to give more consistent dynamic behaviour: objects are only movable insofar as they still have some degrees of freedom left if the objects they depend on are kept fixed. The behaviour for all different cases (e.g. a line through a fixed point) can be decided in advance. Other objects dependent on the object being varied have to change as well, and this still leads to decision problems, but they are less severe. We could give a construction for example 1 as follows:

**Example 3** *Two points  $P$  and  $Q$ , together with a line  $l$ , and the following construction:*

$$\begin{aligned} & \textit{free\_point}(P) \\ & \textit{line\_through\_point}(l,P) \\ & \textit{point\_on\_line}(Q,l) \end{aligned}$$

The line  $l$  would then depend on where  $P$  is placed. That point could be varied freely. The line could then be rotated around  $P$  (and  $Q$  would most logically rotate with it), and while  $P$  and  $l$  are kept fixed,  $Q$  could still slide over the line. Note that such information could not be gleaned from the figure.

It thus seems like a figure might be too general to be practical, and we might be better off with a construction. We therefore decided to go with constructions. This decision implies less interoperability with constraint-based systems, since some of their resources will not be encodable into the format. But it ensures that construction-based DGS will be able to interpret the resources, which they might not if we used figures. Indeed, although there are systems like Geometry Expressions [Sal08] that take a constraint-based approach, and some systems like Geometer's Sketchpad [KCP08] support it, most systems only use constructions.

Another effect of the decision is the explosion of keywords. We have to distinguish between "line\_through\_point" and "point\_on\_line". This is in sharp contrast to figures, where one relation "incident" would suffice, as can be seen from the plangeo Content Dictionaries. In general, if there are  $n$  different types of objects, the construction approach now forces  $n^2$  different

types of incidence on us. This means a more bloated specification of the file format. On the other hand, it is easier for software developers to parse constructions, so it saves trouble there.

### **2.3 Design Decisions: OpenMath**

As stated in the Description of Work, OpenMath will be used for the file format. The advantage of using OpenMath as opposed to a self-chosen xml-format lies in the fact that the use of a Content Dictionary makes for a flexible, open, and reusable standard. First of all, the use of OpenMath enables Intergeo to use other Content Dictionaries already in existence, so it not only saves development time, but relies on already existing mathematical considerations for, e.g., algebraic expressions. Second, other kinds of software that want to use the format in the future can combine it with other Content Dictionaries to enrich its expressive power.

We started looking at the plangeo Content Dictionaries already present on the OpenMath platform (<http://www.openmath.org/>). It turned out that they were aimed more at a constraint-based approach, and the WP3 members decided this was not the road they wanted to take, as elaborated above. The problems having to do with OpenMath that remain are about details of implementation.

A list of all the OpenMath symbols defined so far is listed in appendix A.

### 3 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

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

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.

## 4 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.

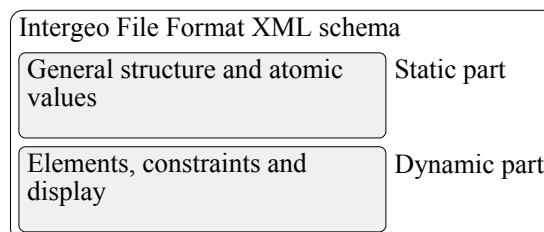


Figure 3: Intergeo file format XML schema

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].

## 4.1 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>
```

## 4.2 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>
```

```
        ...  
    </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>
```

#### 4.2.1 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.

### 4.3 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_through_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">|</line>
    <point>A</point>
    <point>B</point>
  </line_through_two_points>
</constraints>
```

#### 4.3.1 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.

## 4.4 Display

**Note.** The display part is already under development.

The display contains the styles and non-mathematical behaviour of the elements.

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 constrains parts are close to mathematical properties and behaviour, the display part contains visual properties and the behaviour that can not be formalized mathematically.

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.

Styles will be specified individually in the display part.

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.

More than one display can be present and each one represents a possible view of the representation.

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>
```

#### 4.4.1 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>.

#### 4.5 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.

##### 4.5.1 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>
```

##### 4.5.2 Double numbers

Doubles follow the [IEEE 754-1985] standard.

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

Example

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

##### 4.5.3 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>
```

#### 4.5.4 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>
```

#### 4.5.5 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>
```

## 4.6 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/Drafts/Format/xml>.

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/Drafts/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).

## 5 Symbol list

Symbols are the main ingredients used to describe a construction. They define how objects are built and their behaviour. Each icon in a DGS palette is roughly associated to one or more symbols; styles like colours, point sizes, line widths are also represented in the file format using symbols. All DGS's share a big set of common features that will be covered by the Intergeo official symbols. However, for additional features it is acceptable to use proprietary symbols that one day might become official.

The list of symbols is divided in three categories: elements, constraints and display depending on the part of intergeo.xml they appear in. Symbols for the elements and constraints categories are not primarily specified using a XML-Schema but with Content Dictionaries, which are part of the OpenMath standard. With some knowledge of how the atoms are expressed in XML, the description of the symbols with Content Dictionaries and their signature with the Small Type System (STS), the XML schema can be generated automatically.

The complete list of official symbols can be found in Appendix A or at <http://svn.activemath.org/intergeo/Drafts/Format/>.

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.



## 6 First (partial) implementations

As soon as version 1 of the file format got more concrete, some software developers started to investigate its practical usage by integrating it (partially) into their software. Ulrich Kortenkamp started by making Cinderella able to store (intersection) points and (parallel or perpendicular) lines. Then René Grothmann added read support into C.a.R. and successfully opened a demo file stored by Cinderella. This demonstrated that the file format is usable to exchange (at least simple) constructions between different applications. Afterwards Yves Kreis, Markus Hohenwarter and Ingo Schandeler investigated both procedures inside GeoGebra.

These implementations were discussed in the first deliverable and later on again by all members of the work package. As a result other software partners started to implement the preliminary version of the file format: Arnault Ioualalen wrote a translator to and from the TracenPoche proprietary format; Alfred Wassermann created an import library for JSXGraph; Ulrich Kortenkamp extended his implementation to be able to store files and the WIRIS team made first investigations of coding the file format.

Some results and implementation decisions will be described in the following subsections to enrich the ongoing discussion. A live view on the implementation compatibility is available at <http://i2geo.net/xwiki/bin/view/About/I2GformatImplementations>.

### 6.1 Saving

Saving requires the translation of the internal format of the application to the common file format. As the internal representation is quite different in each implementation this part needs to be done individually by each software partner.

### 6.2 Opening

To be able to read a construction stored in the common file format it needs to be parsed and then the elements and the constraints need to be handled. The experience shows that any XML parser is able to parse the file format out of the box. An event-driven library might be interesting but the translation from the common file format to the internal representation is most of the work which needs to be done anyway by each partner. Besides the different programming languages used for the different applications were not in favor of such a library so far, but its necessity is unbroken.

No investigations have been made either with random orders of constraints. Using demo files where the constraints were stored in their acyclic represen-

tation (see Sec. 7.10 on p. 30) file exchange between multiple applications have been successfully demonstrated at conferences and during workshops.

### 6.3 Implementation

The projective dynamic geometry systems like Cinderella, GeoGebra and JSX-Graph have easily and successfully implemented (part of) the current file format. Their internal representation of the objects is quite close to the one used in the file format which facilitates the translation.

In WIRIS however creating a construction that behaves like a DGS requires writing a sequence of commands: a program. Thus the implementation is challenging for them which explains why they do currently not use the official test cases and do not yet support saving/opening of the common file format.

### 6.4 Testing

The development of an official test suite for the common file format has begun. A first test case for points is available at <http://i2geo.net/xwiki/bin/view/About/I2GformatTestSuite-Elements> and has been passed by most of the available implementations. More test cases will follow to build a complete test suite for all elements and constraints.

Different levels of compliance are possible for each element/constraint:

The application can

- read the element/constraint and represent it internally with the correct semantics
- read the element/constraint and represent it internally, though maybe with wrong or missing semantics
- read the element/constraint technically - without throwing exceptions etc. - so it can at least be ignored
- not read the element/constraint technically

The result of the test case is currently the lowest compliance level of the different objects. However the final algorithm to combine the different objects and/or the different test cases needs to be discussed and agreed upon quickly to be able to hand out compliance badges as soon as the file format has become stable.

## 7 Future Work and Request For Comments

As for version 1 of the file format we postponed some decisions that should be made with the help of other developers of DGS. We explicitly invite those to join the discussion and propose solutions or give remarks.

We suggest to read [Kor99] for further mathematical details.

Despite the fact that we are listing a lot of yet to be solved issues, we are confident that the first version of the i2g file format is capable of handling any design decisions that result from the following discussion.

### 7.1 More Elements and Polymorphism

Currently, we focused on a restricted subset of possible geometric elements. This restriction enabled us to agree on the structure and basic composition of the i2g format.

Further elements that should be available are purely geometric elements like conics, loci or polygons, and more general ones like functions or numbers, as well as stylistic elements like text objects or images. A task for the next version of the i2g format is to collect and specify all elements that are currently in geometry software.

Certain elements are very similar to others, for example, segments and rays can be used instead of lines in many cases. Other examples are arcs of circles in comparison to plain circles. The next version of the i2g format has to be able to handle this kind of polymorphism.

Some objects can also be replaced by others in certain special cases, for example, a circle might degenerate to a line, or a conic might degenerate to two lines. Although currently no DGS uses these degenerations, this could be desirable for the future. A DGS might construct a parallel line to a degenerate circle through three collinear points – with our current specification and typing mechanism it is not possible to capture this in the file format. However, we request advise from DGS developers on this issue.

### 7.2 Circle Coordinates

Each DGS uses its own representation of coordinates for the basic elements. In the case of points and lines we could easily agree on standard representations, i.e. homogeneous or Cartesian coordinates. For the case of circles it is not as easy: Usually, circles can be specified by a center and a radius. If the center is “at infinity” and/or the radius is not representable by a real number, then it might be easier to represent the circle by a (symmetric) matrix that

specifies the parameters of its quadratic equation, together with the matrix for the dual conic.

It was agreed that the matrix representation of the quadratic equation is to be used, since it is the most general. Because several DGS developers said they would not be able to handle this representation directly, Ulrich Kortenkamp offered to add functions to the Java library that can offer conversions into a more other representations such as (center,radius) whenever possible.

### 7.3 Ambiguity Resolving

Ambiguity Resolving is crucial for finding the correct positions of elements in stored construction after loading, also known as the persistent naming problem [MP02] from parametric CAD. Assume that an intersection point of two circles is used in a construction. If the two circles are moved into a tangent position, and then the construction is stored, then both intersections have the same coordinates and thus cannot be distinguished. If the circles are moved into a position where both intersection points can be distinguished, then it is essential to pick the correct intersection point.

Most DGS solve this problem by having an implicit order of multiple outputs. This order is dependent on the implementation details of the algorithms and cannot be part of a specification. Also, a point might switch branches later due to homotopy-conserving implementations. This means that this approach cannot be used for a cross-software file format.

For the current version v2 of the file format we introduced circles and conics. The order of the inputs can be determined from the coordinates of the intersection points when they exist and differ from each other. For all other cases, the order is still unspecified. This will lead to problems when files are stored and exchanged. However, we decided that we can only solve this problem if we have test files that expose them.

### 7.4 Styling Information

With version 2, the file format includes styling information that is loosely based on SVG.

Some of the DGS support parametrized styles, for example a color that changes according to the position of an element. This cannot be specified in the file format currently. See the next section for a discussion of the problems that are specific to including parametrized values using functions.

## 7.5 Functions and Scripting

Almost all DGS support functions, used for plotting graphs, defining element dependencies, or changing the style of elements dynamically. All of these use a different language to specify the functions, though many aspects are shared. The conformance to standards varies wildly from OpenMath compliance to unspecified.

Right now it seems impossible to homogenize the various dialects. Actually, the translation from one language to the other can be done easily by humans if an automatic conversion fails, so we decided that all functions should be specified in the private sections of the file format. Each DGS may try to interpret the other function specification, of course, and store its own interpretation as well.

For this, we need a notion of “alternatives”, which should be specified in an upcoming version of the i2g format.

Another difficulty in dealing with functions is that some DGS extend the notion of function to a general-purpose functional programming language. This proves that it is impossible to find equivalent functions algorithmically. Nevertheless, in many cases, mostly non recursive algebraic expressions, the translation is straightforward, and so it might be sufficient to use a heuristic approach.

## 7.6 Mathematical Typesetting

The de-facto standard for mathematical typesetting is TeX [Knu84], and the browser-compliant way is to use MathML [CIM08]. DGS software use both approaches, while the TeX implementation used is usually only a subset of the full TeX system as created by Knuth<sup>1</sup>.

A logical solution to storing typesetting information would be to use the existing OpenMath syntax. It has not been decided yet how to handle the situation, and it is unclear whether it has practical relevance at all. We could not agree on a definitive way to typeset formulae. We suggest that a solution for the function specification problem above will be a solution for this problem as well.

## 7.7 Sending Commands to Constructions

Sending commands to constructions is now handled in the Intergeo API specification document [Kor09].

---

<sup>1</sup>Some use the hoteqn library, others use custom implementations

## 7.8 Macro Constructions

So far there is no notion of Macro constructions. We expect that macros are basically sub-constructions, and it is probably sufficient to add an additional `inmacro` attribute to the constraints and elements. This has to be postponed until the first version can be used successfully for saving and loading of more complex constructions.

## 7.9 Number Representation

Currently, the specification of coordinates uses the IEEE standard for doubles (see Sec. 4.5.2 on p. 21). While this is probably sufficient for most purposes, it lacks the ability to describe *real* coordinates, for example the irrational numbers  $\pi$  or  $\sqrt{5}$ . As there are constructions even in elementary geometry that require such numbers, it is desirable to be able to express them. The OpenMath standard supports this, however, the i2g format cannot be based on the full OpenMath specification as most DGS developers cannot afford the necessary implementation work.

For the time being, we will restrict the number representation to the IEEE standard. This should not be the cause of severe problems, because the coordinates of dependent elements can be recalculated up to arbitrary precision by the DGS itself. If there is a need for other number representations, we will extend the mechanism as described in Sec. 4.5.2.

## 7.10 Acyclicity of Construction Dependencies

Most construction-based DGS require that the dependency graph of a construction be acyclic, but there are some systems (both constraint-based as well as construction-based DGS) that allow for certain circular dependencies. Therefore, we do not enforce this property, and we do not impose a special order for the constraints in the constraint part of the i2g format.

This implies that each DGS has to be able to handle cycles. The easiest resolution is to drop those constraints that close a cycle. Another solution is to add all constraints and break the cycles on-the-fly whenever an element moves. If the DGS can handle the additional constraint, then it should add it.

## 7.11 Library Support

In order to make it easier to work with the i2g format we will try to provide an open-source library in Java to read (and possibly write) the format, see also Section 6. We already applied for a corresponding sourceforge project that will host this open source library. The library will consist of a basic XML parser

that offers an API to access the various nodes in the i2g file. In particular, it should be possible to request the coordinates of elements in Cartesian or homogeneous form if possible, regardless of how they are stored.

This library should enable third parties to work with the i2g format, and thus enhance the accessibility of the format as well as the visibility of the Intergeo project and provide sustainability for the future.

## **7.12 Higher Dimensional Geometry**

With the availability of three-dimensional DGS it is necessary to extend the i2g format to 3D. The basic structure should be similar to what we have now, but more elements and constraint symbols are necessary.

## Appendix A : Intergeo file format OpenMath symbols

The following is a list of OpenMath symbols that we agreed on until 31-07-2009. It will be developed further during the course of the project. At any time, the newest version can be found on <http://svn.activemath.org/intergeo/Drafts/Format/>. In the list, the keyword is listed first, in bold-face. On the next line, the types of its arguments are listed. The order of the arguments is fixed.

### A.1 The elements part

#### **point**

*coordinates*

This will represent a point in the space. It will have coordinates for initialization.

#### **line**

*coordinates*

This will represent 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**

*coordinates, coordinates [,coordinates]*

This will represent 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**

*coordinates, coordinates [,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**

*coordinates, direction | coordinates, coordinates*

This will represent 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.



**conic***matrix [,dualmatrix]*

This will represent a general conic. The coefficients of its associated quadratic form are provided in a  $3 \times 3$  matrix of (possibly complex) numbers. An additional  $3 \times 3$  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 be calculated by the DGS) or a finite set of tracer points (specified together with the corresponding values of the mover point).

**A.1.1 Auxiliary symbols****direction***(homogeneous) 2-vector*

Directions are auxiliary symbols, used mainly to construct rays. Is specified either as  $(a, b)$  or projectively (homogeneously) as  $(a : b : c)$ .

**linear\_equation***coefficient vector, number*

A linear equation can be used to describe a line in the plane.

## A.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.

### **point\_on\_line**

*new point, line*

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*

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. The line can still be rotated around the point by the user.

### **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, the line will be undefined. If two of the points are equal, but not all three, then the line returned will be the one containing these three points.

### **line\_angular\_bisectors\_of\_two\_lines**

*new line, new line, line, line*

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 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\_in\_direction**

*new\_ray, point, direction*

A new ray that starts at the point and moves in the direction provided.

**ray\_from\_point\_via\_point**

*new\_ray, point, point*

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

**line\_parallel\_to\_line**

*new\_line, line*

A line which can be moved with the mouse while it remains parallel to the given one.

**line\_parallel\_to\_line\_through\_point**

*new\_line, line, point*

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

**line\_perpendicular\_to\_line**

*new\_line, line*

A new line which can be moved with the mouse while it remains perpendicular to the given one.

**line\_perpendicular\_to\_line\_through\_point**

*new\_line, line, 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, line*

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.

**direction\_of\_ray**

*new direction, ray*

The direction of a ray.

**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, circle*

The intersection points of two circles. 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, circle*

The new point is an intersection point of the two circles, and is different to the provided point.

**intersection\_points\_of\_circle\_and\_line**

*new point, new point, circle, line*

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.

**other\_intersection\_point\_of\_circle\_and\_line**

*new point, point, circle, line*

The new point is the intersection point of the circle and the line, and is different to the provided point.

**intersection\_points\_of\_two\_conics**

*new point, new point, new point, new point, conic, conic*

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.

**intersection\_points\_of\_conic\_and\_line**

*new point, new point, conic, line*

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 de-

generate and contained a line coincident with the intersecting line, a DGS may return “undefined” or the intersection line.

### **circle\_tangent\_lines\_by\_point**

*new\_line, new\_line, circle, 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*

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*

The new point is the center of the given circle.

### **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.

### A.3 The display part

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

#### **fill**

Fill is used to specify whether the paths have to be filled and which color. The value has to be given as a **paint**.

#### **fixed**

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

#### **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 T<sub>E</sub>X/L<sub>A</sub>T<sub>E</sub>X or MathML.

#### **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**.

#### **stroke**

Stroke is used to specify whether a path has to be drawn and with which color. 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**.

#### A.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.

## Bibliography

- [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.
- [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 T<sub>E</sub>Xbook*. 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.