I2G

INTERGEO

# Deliverable Nº: D3.3

# Common File Format v1

## The INTERGEO Consortium

## July 2008

**Version:** version of August 1, 2008 9:00 A.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**

| | |
|---|---|
| **Project ref.no.** | IST-507826 |
| **Project title** | INTERGEO - Interoperable Interactive Geometry for Europe |

| | |
|---|---|
| **Deliverable status** | submitted |
| **Contractual date of delivery** | M10 – July 2008 |
| **Actual date of delivery** | July $31^{th}$ 2008 |
| **Deliverable title** | Common File Format v1 |
| **Type** | Presentation |
| **Status & version** | submited version of August 1, 2008 9:00 A.M. |
| **Number of pages** | 34 |
| **WP contributing to the deliverable** | WP3 |
| **WP/Task responsible** | Daniel Marquès |
| **Authors** | 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) |
| **EC Project Officer** | Spyridon Pilos |
| **Keywords** | dynamic geometric system, file format, openmath, standard |
| **License** | This document is available under the license *Creative Commons Attributions Sharealike Germany 2.5* [Cre08] |

# Contents

# 1 Executive Summary

The present document is the specification of the first version of the Intergeo file format. The specification is the result of intensive collaboration between Dynamic Geometric System (DGS) software developers and experts. During a period of 24 months we expect substantial modifications of this specification, provided that some implementations will be done by DGS developers and a lot of practical issues will arise.

The Intergeo file format aims at the interchange of content between DGS's. It is on 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 list is important because choosing correctly its content is crucial to achieve the successful interoperability between all DGS's.

# 2 Introduction

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 the 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 exist. 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

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

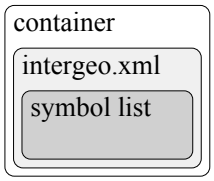


Figure 1: Specification structure in layers

**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 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 set of cohesive 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:*

$$\textit{line } l \textit{ is incident to both points } P \textit{ and } Q.$$

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

$$\begin{gathered} P \textit{ is on } l \\ l \textit{ is tangent to } \Gamma \\ \textit{the distance of } P \textit{ to the center of } \Gamma \textit{ is } 10. \end{gathered}$$

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 dependant 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:*

*free_point(P)*
*line_through_point(l,P)*
*point_on_line(Q,l)*

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 of 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 saves development time. 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 sofar 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 at 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 do not depends on media files might 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 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 definition of the plot area and styles are its objectives.

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 XML, 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.

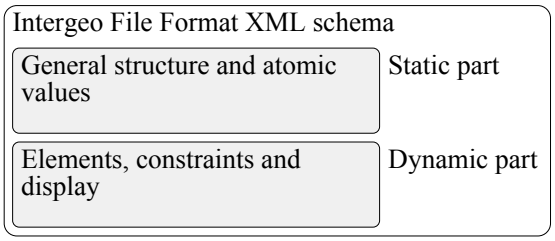


Figure 3: Intergeo file format XML schema

## 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>
    <!-- here goes the list of available elements -->
    <xs:element name="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:choice>
      <xs:element name="homogeneous_coordinates"/>
      <xs:element name="euclidean_coordinates"/>
    </xs:choice>
    <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>
      <!-- here goes the list of available
           constraints -->
```

```
      <xs:element name="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>
```

### 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

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

A display describes general information like axes, grid, viewport, background colour, snap properties, default styles, etc.

And, for each geometric object, it describes its styles. The current status of this document does not specify the display part. This will be achieved in the future.

## 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>
```

# 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 styles depending on the part of intergeo.xml they appear in. They 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 and whether we will admit new Content Dictionaries after the Intergeo project 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. Finally Yves Kreis, Markus Hohenwarter and Ingo Schandeler investigated both procedures inside GeoGebra. Their results and implementation decisions will be described in the following subsections to enrich the ongoing discussion. This is important because their parser/handler will be made available as open source (library) independently of GeoGebra and thus be available to every DGS developer willing to implement the Intergeo File Format. A live view on the implementation compatibility is available at `http://svn.activemath.org/intergeo/Drafts/Format/implementation.txt`.

## 6.1 Saving

Few changes were required to store the Intergeo file format from GeoGebra as its own proprietary file format is XML-based. The elements and constraints were simply written in its acyclic representation and the elements contained the display information. In the Intergeo file format these informations are separated into three parts (see page 13). This allows a different parsing of the file (see next section). Besides the `<display>` part allows to have different views of the same construction.

## 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.

GeoGebra uses an event based parser similar to the famous SAX parser, but with much less functionality. No changes to the parser were required to switch from its proprietary XML-based format to the Intergeo file format. Thus any XML parser is probably able to parse the file format out of the box.

A new handler was written to deal with the elements and the constraints. Logically it is event based as well. GeoGebra is able to read the file in one step as long as the constraints are stored in its acyclic representation (see Sec. 7.10 on p. 27). No investigations have been made with random orders of constraints, but probably some ordering needs to be done before processing them.

First all elements are created as free objects and get their coordinates assigned. This creates a first representation of the construction and defines all input/output elements used

later in the constraints. As a consequence we get an increased probability that a construction can be opened even if some constraints cannot be fulfilled as the software might be unable to deal with them. As an example take the constraint `<line_through_point>` which is not available in all DGS. Its output element of type `<line>` is however available in all DGS, will thus be created using the provided coordinates and nevertheless be available as input for further constraints like `<point_on_line>`.

Then all constraints are handled. As mentioned before only ordered constraints can be handled at the moment by GeoGebra. The typed input/output elements provide an easy way to check the corresponding type of the free elements created in the previous step and thus help detect errors in the file. Before handling the constraint, the free element is removed from the construction to free the id, but its coordinates are remembered. Then the constraint is executed and finally the id and coordinates of the output element adjusted to the ones of the removed free one. Using the previously stored coordinates of the output elements is one way to (partially) perform ambiguity resolving (see Sec. 7.3 on p. 25). As an example let's consider the intersection points of two circles. The order of the intersection points is dependant on the implementation details of the algorithms and probably different in the available DGS. By overwriting the output elements using the stored elements the starting order will be the same everywhere. It doesn't however solve all the problems mentioned in the subsection 7.3.

## 6.3 Testing

For testing purposes we provide a simple construction using (intersection) points and (parallel or perpendicular) lines only. The construction in the test file looks like Fig. 4 and can be stored and opened in the latest beta version of GeoGebra[1].

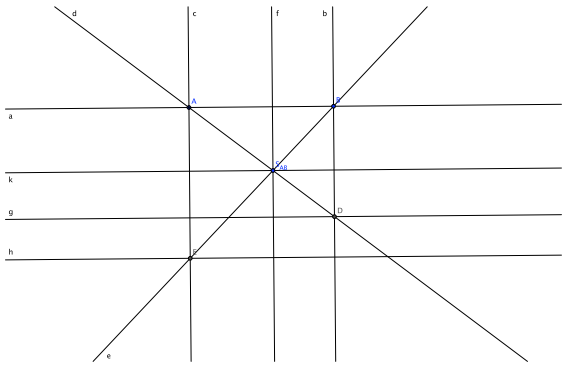


Figure 4: Expected rendering of the test file

The corresponding `intergeo.xml` is provided in Appendix B on page 31.

[1] GeoGebra beta v3.1.10.0, by Yves Kreis, Markus Hohenwarter and Ingo Schandeler, available at `http://www.geogebra.org/webstart/beta/geogebra-beta.jnlp`

# 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.

For the current version, we could not agree on the right way of specifying the coordinates of circles, as the matrix representation was considered to be to general for some, while the point-radius representation was considered to be not general enough by others.

### 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.

As soon as circle (or conic) intersections are introduced, we will have to find a solution to this problem.

### 7.4 Styling Information

Currently, the file format does not store any styling information. This problem has been considered easy to solve once we are able to exchange construction data. Proposals for a solution range from very basic styling support specifying colors and thickness up to full SVG-compatibility. The latter seems to be out of reach for many DGS, though.

We reserved the `<display>` section of the file (see page 13) for this kind of information. This is in particular due to the fact that styling information for a single element could be different for each view.

Some of the DGS support parametrized styles, for example a color that changes according to the position of an element. 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 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 uses both approaches, while the TeX implementation used is usually only a subset of the full TeX system as created by Knuth[2].

We could not agree on a definitive way to typeset formulae. Probably it would be a good idea to support MathML, but most of the DGS developers do not want to adopt it, as it seems. So this is currently unspecified and mathematical typesetting has to be specified in the private part of a construction. 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

A common way to interact with constructions is to use Javascript or another scripting language to send commands to the geometry kernel that changes the properties of a construction. We will specify this API in a later revision of the file format. The basic commands will include loading of (parts of) a construction, movement of free elements, and change of stylistic information (for example, show/hide/recolor elements). The specification can be based on the i2g format, as all actions represent changes of the construction.

## 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.

[2]Some use the hoteqn library, others use custom implementations

### 7.9 Number Representation

Currently, the specification of coordinates uses the IEEE standard for doubles (see Sec. 4.5.2 on p. 19). 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.

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 Dependances

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 now, but there are more elements and constraint symbols necessary.

# Appendix A : Intergeo file format OpenMath symbols

The following is a list of OpenMath symbols that we agreed on until 31-07-2008. 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 boldface. On the next line, the types of its arguments are listed.

## 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.

## A.2 The construction part

In the list for the construction keywords, the order of the arguments is fixed. Moreover, the first argument of every keyword is always the new, dependent element of the construction. The other arguments are the existing objects this new object depends on. The arguments will be identifiers of objects declared in the elements part of the file.

**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.

**line_through_point**
*new line, point*
A new line that goes through a given point. The line can still rotate around the point.

**line_through_two_points**
*new line, point, point*
A new line that goes through two given points (being thereby completely determined).

**line_perpendicular_to_line**
*new line, line*
A new line that is perpendicular to a given line.

**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.

**line_parallel_to_line**
*new line, line*
A new line that is parallel to a given line.

**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.

**point_intersection_of_two_lines**
*new point, line, line*
A new point that is the intersection point of two given lines.

**angular_bisector_of_three_points**
*new line, point, point, point*
A new line that is the angular bisector of three given points.

**angular_bisectors_of_two_lines**
*new line, line, line*
A new line that is the angular bisector of two given lines.

## 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. However, for the sake of being able to transmit the labels of objects to other software, this specific keyword has to be present.

**label**
*string*
A label is a string that can be displayed to the user for identification purposes, e.g. a label "P" for a point, or "l" for a line. This should be viewed as a "front-end" name of the object, handy for the user but not necessarily unique, as opposed to the "back-end" name, the unique identifier the software uses.

# Appendix B : Testing (intergeo.xml)

The construction in the test file looks like:

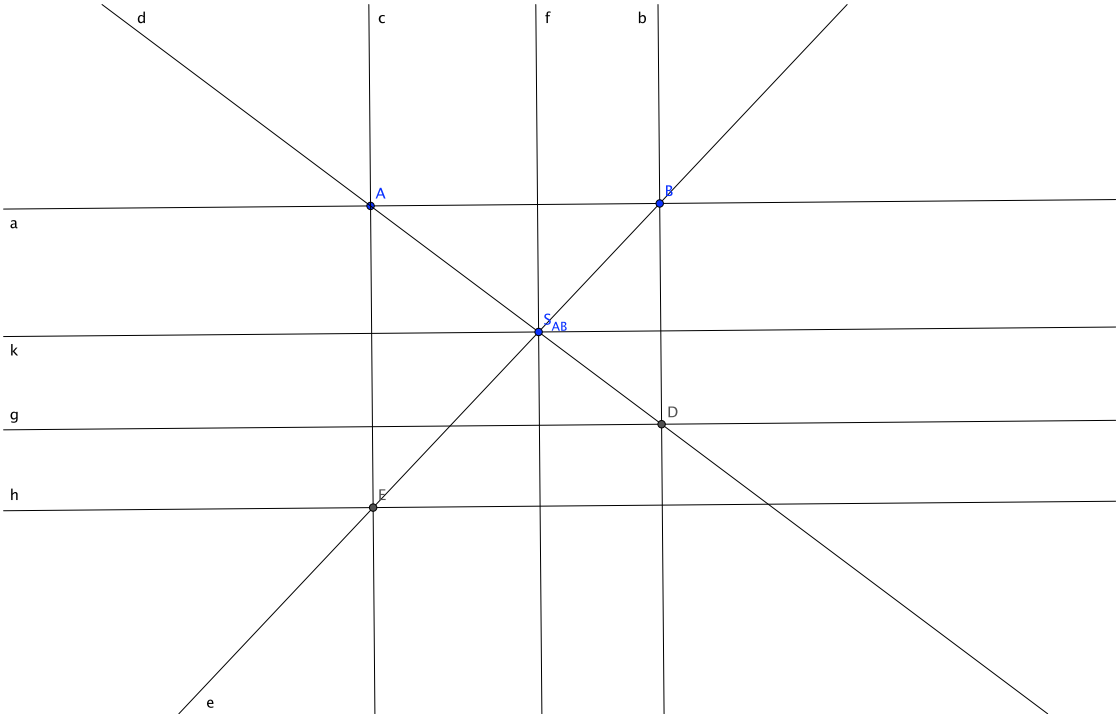


Figure 5: Expected rendering of the test file

The corresponding `intergeo.xml` written by GeoGebra 3.1.10.0 is:

```
<?xml version="1.0"?>
<!--
        Intergeo File Format Version 1.00.20080731
        written by GeoGebra 3.1.10.0 (July 31, 2008)
-->
<construction>
        <elements>
                <point id="A">
                        <homogeneous_coordinates>
                                <double>-2.4</double>
                                <double>4.0</double>
                                <double>1.0</double>
                        </homogeneous_coordinates>
                </point>
                <point id="B">
                        <homogeneous_coordinates>
                                <double>2.28</double>
                                <double>4.04</double>
                                <double>1.0</double>
                        </homogeneous_coordinates>
                </point>
                <line id="a">
                        <homogeneous_coordinates>
                                <double>-0.040000000000000036</double>
                                <double>4.68</double>
                                <double>-18.816</double>
                        </homogeneous_coordinates>
                </line>
                <line id="b">
                        <homogeneous_coordinates>
                                <double>-4.68</double>
```

```
                        <double>-0.040000000000000036</double>
                        <double>10.831999999999999</double>
                </homogeneous_coordinates>
        </line>
        <line id="c">
                <homogeneous_coordinates>
                        <double>-4.68</double>
                        <double>-0.040000000000000036</double>
                        <double>-11.072</double>
                </homogeneous_coordinates>
        </line>
        <point id="C">
                <homogeneous_coordinates>
                        <double>0.32</double>
                        <double>1.96</double>
                        <double>1.0</double>
                </homogeneous_coordinates>
        </point>
        <line id="d">
                <homogeneous_coordinates>
                        <double>2.04</double>
                        <double>2.7199999999999998</double>
                        <double>-5.984</double>
                </homogeneous_coordinates>
        </line>
        <line id="e">
                <homogeneous_coordinates>
                        <double>-2.08</double>
                        <double>1.9599999999999997</double>
                        <double>-3.176</double>
                </homogeneous_coordinates>
        </line>
        <line id="f">
                <homogeneous_coordinates>
                        <double>-4.68</double>
                        <double>-0.040000000000000036</double>
                        <double>1.576</double>
                </homogeneous_coordinates>
        </line>
        <point id="D">
                <homogeneous_coordinates>
                        <double>29.223679999999995</double>
                        <double>5.90784</double>
                        <double>12.647999999999998</double>
                </homogeneous_coordinates>
        </point>
        <line id="g">
                <homogeneous_coordinates>
                        <double>-0.5059200000000004</double>
                        <double>59.19263999999999</double>
                        <double>-26.479743999999997</double>
                </homogeneous_coordinates>
        </line>
        <point id="E">
                <homogeneous_coordinates>
                        <double>-21.828159999999997</double>
                        <double>-8.16608</double>
                        <double>9.255999999999998</double>
                </homogeneous_coordinates>
        </point>
        <line id="h">
                <homogeneous_coordinates>
                        <double>-0.3702400000000003</double>
                        <double>43.31807999999999</double>
                        <double>37.34412799999999</double>
                </homogeneous_coordinates>
        </line>
        <point id="F">
                <homogeneous_coordinates>
                        <double>3357.5491937894385</double>
                        <double>28.697001656320015</double>
                        <double>7.105427357601002E-15</double>
                </homogeneous_coordinates>
        </point>
        <line id="k">
                <homogeneous_coordinates>
                        <double>-28.697001656320015</double>
                        <double>3357.5491937894385</double>
                        <double>-6571.613379297277</double>
                </homogeneous_coordinates>
        </line>
</elements>
<constraints>
```

```
                <free_point>
                        <point out="true">A</point>
                </free_point>
                <free_point>
                        <point out="true">B</point>
                </free_point>
                <line_through_two_points>
                        <line out="true">a</line>
                        <point>A</point>
                        <point>B</point>
                </line_through_two_points>
                <line_parallel_to_line_through_point>
                        <line out="true">b</line>
                        <point>B</point>
                        <line>a</line>
                </line_parallel_to_line_through_point>
                <line_parallel_to_line_through_point>
                        <line out="true">c</line>
                        <point>A</point>
                        <line>a</line>
                </line_parallel_to_line_through_point>
                <free_point>
                        <point out="true">C</point>
                </free_point>
                <line_through_two_points>
                        <line out="true">d</line>
                        <point>A</point>
                        <point>C</point>
                </line_through_two_points>
                <line_through_two_points>
                        <line out="true">e</line>
                        <point>C</point>
                        <point>B</point>
                </line_through_two_points>
                <line_perpendicular_to_line_through_point>
                        <line out="true">f</line>
                        <point>C</point>
                        <line>c</line>
                </line_perpendicular_to_line_through_point>
                <point_intersection_of_two_lines>
                        <point out="true">D</point>
                        <line>b</line>
                        <line>d</line>
                </point_intersection_of_two_lines>
                <line_perpendicular_to_line_through_point>
                        <line out="true">g</line>
                        <point>D</point>
                        <line>a</line>
                </line_perpendicular_to_line_through_point>
                <point_intersection_of_two_lines>
                        <point out="true">E</point>
                        <line>c</line>
                        <line>e</line>
                </point_intersection_of_two_lines>
                <line_perpendicular_to_line_through_point>
                        <line out="true">h</line>
                        <point>E</point>
                        <line>a</line>
                </line_perpendicular_to_line_through_point>
                <point_intersection_of_two_lines>
                        <point out="true">F</point>
                        <line>g</line>
                        <line>h</line>
                </point_intersection_of_two_lines>
                <line_through_two_points>
                        <line out="true">k</line>
                        <point>C</point>
                        <point>F</point>
                </line_through_two_points>
        </constraints>
        <display>
                <point id="C">
                        <label>S_{AB}</label>
                </point>
        </display>
</construction>
```

# 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.

[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.

[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.