Manipulation of properties of CGM objects in viewing environments

Name: Dieter Weidenbruck
Job Title: CEO
Affiliation: ITEDO Software
Address Line: Markt 21
City State Country: Siegburg, Germany
ZipCode: 53721
Email: dieter@itedo.com

Dieter Weidenbrück is the founder and CEO of ITEDO Software in Germany, the worldwide market leader for technical illustration solutions. In addition, he is the CEO and President of ITEDO Software LLC in the United States. He is the primary architect of the IsoDraw and IsoView software programs. Dieter Weidenbrück has developed a considerable experience in documentation standards and is actively participating in standardization efforts concerning technical illustration and especially CGM. He currently serves as the Secretary of the CGM Open consortium.

WebSite: www.itedo.com

Background

The increasing use of structured CGM graphics has led to new requirements for accessing and manipulating certain aspects of a graphics file at runtime. More and more illustrations are not only used to display graphic information but to serve for identification and navigation purposes. The first requirement that became obvious was the one for hotspots and links resulting in the definition of the so-called grobjects (graphic objects). Although this step has not completed yet in a satisfactory way this first approach is promising.

This paper discusses the need to access the graphic file and its content to extract and manipulate graphic information at runtime. Examples for usage are the collection of structural data in an external database or application, or simply the change of display properties to improve readability. The purpose of this paper is to document the results of a study regarding the practical requirements for accessing structured graphics.

Additional information related to structured CGM graphics can be found in an article written by da Ponte/Goertz/Henderson for XML Europe 99 called “Implementing a viable architecture for standardized intelligent graphics”.

Structure of a CGM file

When looking at a CGM file there is a file structure dictated by the CGM standard (ISO 8632). A metafile can contain one or multiple pictures that in turn contain the graphical primitives like lines or text elements.

At runtime it is most unlikely that anybody wants to address a single line or circle. So apart from the file structure the logical structure of the content becomes very important. The logical structure is quite similar to the file structure, it just uses additional constructs to identify objects.

A picture can contain layers. Layers can be used to control visibility of different versions of a graphics, e.g. language dependent layers would contain the text for one language each.

A layer can contain graphic objects. A simple way of looking at grobjects is as a group of graphical primitives with an ID attached to them. They also may have other attributes, like a name or a hotspot region. Grobjects may be nested so that a true tree structure is accomplished.

Data types

Within the scope of this paper certain data types have been used to convey information between the CGM file and the outside world:

Data Type	**Description**
String	A character string
Integer	An integer number
Float	A floating point number
Rectangle	A series of four floats, standing for top left bottom right
RGBColor	Three integers standing for red, green, blue
Object	An object like a layer or grobject
List	a list of objects like layers or grobjects. The list may be accessed using indices. It also exposes a list count to the outside.

More data types can be thought of.

Accessing information

Information about the file and its structure

To identify a file at runtime, three different strings may be useful. The **filename** is the name of the file without any path or URI information. The **path** is the file path to the file on disk. The **URI** is the location on the web server.

A file can contain one or multiple pictures, so an important piece of information about a file is the **number of pictures** and a list of **pictures**. By obtaining a list of pictures from the viewer it is possible to access the pictures using an index.

The entire file has graphic attributes as well. The display **scale** and the **offsets** in X and Y determine the position of the file inside the viewer. Further graphical attributes include the extent of the file in millimeters and details like fonts, line weights, or fill colors. Although there may be various line weights used on the illustration it still makes sense to manipulate the line weights at file level, e.g. to increase the line weights to 150%.

Individual objects

There are different properties for different kinds of objects. Some of them are read-only while others are read-/writeable.

Pictures

Pictures are separate entities inside a CGM file, so each picture represents a complete tree structure. There is no other relationship between two pictures other than that they have been stored inside a single file. In practice I do not recommend to store multiple pictures in a single file because of the increased file size. Every time the user wants to see one of the pictures the entire file will be downloaded.

From the user's point of perspective it is also not obvious that he is working with a file containing multiple images.

A picture has a name and a list of layers contained therein. The number of layers can also be obtained from a picture. Graphical attributes can be manipulated at the picture level in the same way described for the entire metafile.

Layers

There have been intense discussions inside the CGM community with regard to layers reflecting the two ways how layers can be used to structure a file. In general layers establish a structure level inside a CGM picture. Layers can be used to switch display of certain portions of an illustration on or off at runtime.

A layer has a name and a list of objects. It is important to note that there may also be other graphical primitives not contained in graphic objects. In general these are not considered to be important at runtime, otherwise they could have been placed inside an object easily.

Once we have a layer we can get the name of its picture. Again the graphical attributes can be controlled for all elements on a certain layer.

Graphic objects

The graphic object (grobject) is the core object inside a CGM file. It provides a way to collect primitives in a group that is identified by means of certain attributes. This is the object we are most interested in. It is identified by a unique ID string and a common name. It may carry additional attributes like the linkURI or screentip attributes defined in WebCGM.

To enable access to the tree structure it is helpful to retrieve the neighbors and the parent of this grobject. The parent can either be another object or a layer. There is also a layer property giving direct access to the layer if the object is deeply nested. A grobject itself can contain other grobjects. As with other structure elements the number of children and the list of children can be obtained.

The grobject allows for manipulation of graphical attributes as described for other structure elements.

Extracting information

Object attributes

You can extract information from an object that has been identified uniquely. In most cases you will only be interested in meta data stored in the CGM file. Examples are the screentip or linkURI attributes defined in WebCGM. If an attribute is not present an error message will be issued.

Graphic attributes

Important graphical attributes are the visibility and the extent of an object. You may also be interested in details like line weight, colors, or text details. If you query an attribute you will receive either a value for this attribute or one of two errors: Attribute not present or No common attribute. The latter one is used in cases where there is no unique answer to your query. Imagine a layer with graphic primitives using different line weights. If you query the line weight there is no common answer.

Manipulating objects

Concept of manipulation at runtime

In web environments the browser can control a lot of attributes. This includes font sizes, fonts, colors and more. In the past it has been impossible to control similar attributes inside graphics. The reasons

for this kind of control include improvement of readability or emphasizing certain areas inside an illustration as desired.

This approach establishes a way to manipulate certain aspects of the illustration without the manual interaction of an illustrator. Was it necessary in the past to work with overlays in different colors or even separate illustrations to just change the color of a line, using this new method it can be done at runtime. Given a single CGM file the web designer can now show an illustration on screen that the illustrator has never prepared that way.

Manipulating attributes

Within the scope of this study only certain attributes have been treated. This list can certainly be expanded over time.

Stroke

The stroke attribute stands for the line weight of the lines of the object(s). It is expressed as a floating number in millimeters or as a percent number. The stroke can be set in an absolute or relative way.

Example:

Stroke = 0.6 mm	absolute setting for all line elements having a stroke
Stroke = 150 %	relative setting, maintains the relationship between line weights

StrokeColor

The strokeColor attribute controls the color of the lines of the object(s). It is expressed as a series of integers for red, green, and blue. The strokeColor can be set in an absolute or relative way:

Example:

StrokeColor = 255,0,0	absolute setting for all line elements having a stroke
StrokeColor = 50 %	relative setting, reduces intensity of the color to 50%

HasFill

A read-only attribute telling you whether the object(s) has/have a fill or not.

FillColor

The fillColor attribute controls the fill color of the object(s). It will be applied to those primitives only that have a fill in the file. It is expressed as a series of integers for red, green, and blue. The fillColor can be set in an absolute or relative way:

Example:

FillColor = 255,0,0	absolute setting for all elements having a fill
FillColor = 50 %	relative setting, reduces intensity of the fill colors to 50%

TxtFont

The txtFont attribute controls the font used on text elements contained in the object(s).

TxtSize

The txtSize attribute controls the font size used on text elements contained in the object(s).

TxtFace

The txtFace attribute controls the font face used on text elements contained in the object(s), e.g. normal, bold, italic etc.

TxtContent

The txtContent attribute controls the content of a text element inside the grobject. It makes most sense to specify grobjects in a way that it is possible to set the text for an individual text element. This can be used for translation purposes or to show different numbers in dimensions based on configurations.

Visible

The visible attribute controls the visibility of the object(s).

Query

A query is a way to extract information from a CGM file. There has not been a standardized approach so far. Most discussions go around text search, however, there is probably more to be found.

Apparently there is the need to search for a particular object based on its ID or name. Apart from that I can think about a searched based on certain attributes of an object based on the definitions above. The result would be a list of items. This list could be processed subsequently item by item.

The research and implementation of query methods has not been concluded yet.

Walking the object tree

Given the information about objects as described above, it is now possible to "walk" the object tree inside a CGM file by looking for neighbors, parents and children. It is important to see that the CGM file contains a true tree structure that can be used to enumerate items.

Example:

Imagine a flow chart of people working in a company. If each entry is a separate grobject you can make the subordinates of a person his "children". At runtime, it would be possible to enumerate them and to show or hide them. This way it would be possible to clean up the visual representation by hiding the information not needed in a certain situation.

Finding objects with specific attributes

A query could specify certain attributes to search for. The result would be a list of objects matching these criteria. Thus it becomes possible to search for all text elements with a text size of 7 points in order to change them to 10 points on the screen.

Text search

A text search is tricky in CGM as this format does not know anything about paragraphs. Even more, it is completely up to the creator of the CGM file to store words as contiguous chunks or to split them wherever he wants. A text element can not span multiple lines in CGM. WebCGM introduced the concept of a "para" object to cope with this issue.

Apart from these restrictions a query could search for all text elements containing a certain string.

Conclusion

The demos shown in the presentation clearly demonstrate the power of manipulations done at runtime. This is the first step away from static graphics to highly dynamic environments used in training or simulation environments.

The demos have been built using a development version of IsoView that has been enhanced to support the additional functions. IsoView is used in both web environments and Visual Basic applications.

No announcement has been made as to if and when a version of IsoView will be released supporting this interface.