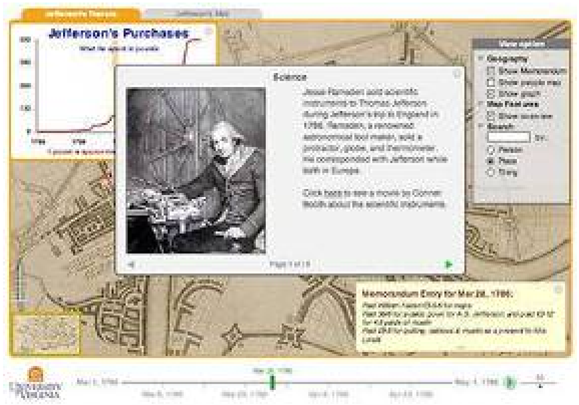
visualeyes

VisualEyes Project Guide

8/30/11



VisualEyes is a web-based authoring tool developed at the University of Virginia to weave images, maps, charts, video and data into highly interactive and compelling dynamic visualizations, and seamless access to other web applications such as Google Earth.

VisualEyes enables users to present selected primary source materials and research findings while encouraging active inquiry and hands-on learning among general and targeted audiences. It communicates through the use of dynamic displays - or “visualizations” - that organize and present meaningful information in both traditional and multimedia formats, such as audio-video, animation, charts, maps, data, and interactive timelines.

Users can view preset collections of events as well as construct their own views of the events based on selected criteria. The effective use of the visualizations can reveal and illuminate relationships between multiple kinds of information across time and space far more effectively than words alone.

About this guide

This project guide is intended to serve as the overall documentation associated with VisualEyes. The main sections of the document will follow the order in which the aspects of VisualEyes are introduced in the following section How VisualEyes Works. It is the intention that through the reading of this manual one will be able to learn how to create a project in VisualEyes. For this reason, the manual will refer frequently to the Project Sampler, a collection within VisualEyes itself of examples of various features that includes instructional screen-casts.

There are two other reference documents: The VisualEyes XML Reference Guide which provides details on the elements available for use in projects and their attributes; and the VisualEyes GLUE Reference Guide, which provides details on using GLUE scripts for advanced interactivity.

Also available is the VisualEyes Tutorial, which is a step-by-step instructional manual designed around a specific project, the may chose to take the tutorial if you prefer that style of learning; however, it is the intention that you will be able to learn how to use VisualEyes by reading this document alone.

We encourage you to look at the various projects on our website to get a feel for the kind of projects that have already been made using VisualEyes.

Table of Contents

How VisualEyes works

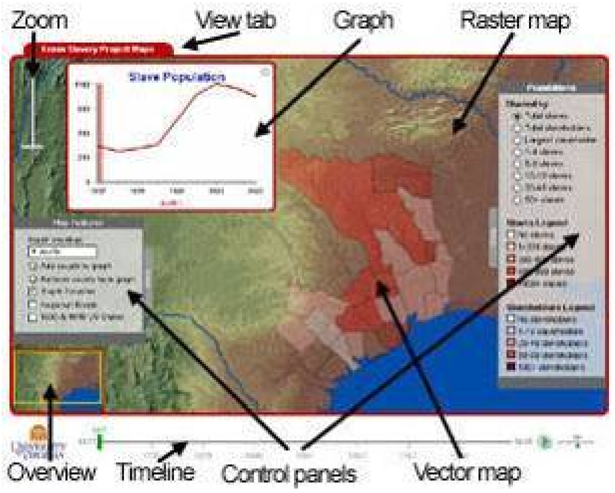
VisualEyes is a Flash-based authoring system that uses the Internet to connect various resources, such as images, maps, video and data together in a seamless interactive presentation. It is a "virtual Lego set" containing an number of features.

A VisualEyes project consists of information and images (**resources**) assembled into what are called *views*. The views are customizable and interactive, enabling users to change how the various *resources* interact (using *controls*). Often these views use a special scripting language, called *GLUE* to control the appearance and disappearance of various elements.

Views

Each VisualEyes project is divided into a number of views, with each view existing in a separate tab, accessible at the top of the window. Clicking on any of the tabs will bring up a different view.

Each view displays the resources, or images and data, that make up the project. The resources in a view can be shown interactively within a time period using the timeline tool. Views can also contain event descriptions, primary-source documents and imagery, maps, digital movies and audio, pop-up information boxes, animations, charts, and graphs of historical data to make the view highly interactive.



Each view can be constructed to show events that match certain criteria based on data. These views can be fixed for demonstration purposes, or left open to allow users to explore various relations between the elements provided allowing for both purposeful and serendipitous discovery of complex interrelations.

Resources

Resources are the sources of information or media to be utilized in a project. The resources could be a map, some media, a table of data, or a graphic. Each resource item contains a URL to link the resource to the project. VisualEyes uses four basic types of resources:

1. **Images**- Digitized images of primary source documents from digital archives can be displayed and integrated into maps, animations and other visualizations. These images can be in JPEG, GIF, or PNG formats, and can be dynamically sized and positioned. The image must be Internet accessible. The Flickr image sharing site is a free and convenient place to store images online.

2. **Movies**- Video files can also be used, including Flash movies and animations. Video file must be available via the Internet or on Google's YouTube site.

3. **Maps-** VisualEyes contains a fully interactive geographic information viewer to display vector-based maps from GIS systems such as ArcGIS, Adobe Illustrator, and Adobe Fireworks.

4. **Data**- A rich array of historical data can be imported into VisualEyes from a database as a table. This data can be supplied as a CSV, XML file, retrieved directly from Google Docs, or embedded directly in the project script.

Controls

Controls provide an opportunity for your users to interact with your project with timelines, animation players, and control panels:

1. **Control Panels-** Views can have multiple pull-out areas docked to a side of the screen that can be expanded or collapsed as needed and contain a number of collapsible check boxes to toggle on and off various features of the map, such as data overlays, roads, town names, etc. Various map features, such as the overview navigation insert and map legend can be turned on and off here as well, but assigning a GLUE script to be activated on clicking. Control panels can contain radio buttons, check-boxes, combo-boxes, sliders, text input boxes, buttons, headers and legends.

2. **Timelines-** Each view can have its own timeline that can control the temporal aspect of the project. Sliding the cursor changes the view's date, which in turn can change the way in which information is displayed if it is time dependent.

3. **Animation Players-** The current time on the timeline can be animated over time, using a player control, allowing the project to animate any time dependent elements from any point on the timeline to another.

4. **Zoomers and Overviews-** The screen can be controlled by a zoom slider and/or a small overview inset that facilitates panning through the screen.

Displays

Displays provide the means of showing data and information within your project to the viewer. They connect the raw data from tables into compelling representations that users can interact with, and include:

- **Text displays-** Text displays can be drawn using dynamically generated data base on data from databases, time from timeline, settings of control panel items, or any combination of them.

- **Paths-** A series of positions on the screen (specified by pixels or latitude, longitude if a map) can be defined to appear at particular times. Each position can be marked by an icon, drawn shape or an image file. Clicking on the position can call up a web page draw graphical elements, or pop-up window showing some information. Lines can be drawn to connect these positions.

- **Graphs-** Various types of graphs (line, bar pie, scatter, etc.) can be drawn using dynamically generated data base on data from data sources, time from timeline, settings of control panel items, or any combination of them.

- **Concept and network maps-** A path can be arrange in a radial concept map format to help visualize relationships between objects shown in radial, hierarchical (i.e. an org chart), or free-form shapes.

- **Timeviews and shelves-** These can be used to display interactive timelines and scrollable collections of images.

- **Widgets-** Widgets are a type of graph that graphically displays a single continuous value on the screen, such as a dial, clock, thermometer, etc.

GLUE

GLUE (The General Language to Unite Events) is a simple scripting language that connects the various resource elements connect with one another and controls how they are displayed. VisualEyes knows how to render a number of types of resource, such as tables, charts, text area, movies, audio clips, vector and raster maps, and the GLUE language contains elements to cause them to display.

GLUE contains elements for linking user-generated actions, such as clicking on the screen with actions. Glue also provides an opportunity to calculate tables and fields in resources based on a simple script in the tag. Many common types of operations can be defined between these elements, so that VisualEyes is able to relate rich data relationships between them and visualize them on a special and temporal basis.

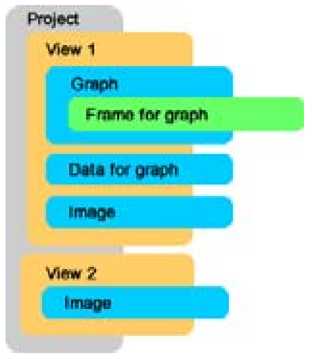
When you use Wizards in VisEdit, necessary pieces of glue are usually automatically created for you. You will use GLUE later to customize your project, but if you are curious now, you can find more info in the VisualEyes GLUE Reference Guide..

Elements and Attributes

VisualEyes uses a script format called XML to represent the projects internally. XML is a simple text format for storing information that a computer _and_ a person can understand. If you use the VisEdit tool to create your project, you do not need to know how to format XML, but an understanding of its building blocks is useful:

Elements

Elements are building blocks in VisualEyes that are connected together to create a project. There is one ***project*** element, the element that contains your entire project. The ***view*** element requires additional elements to be nested within it.

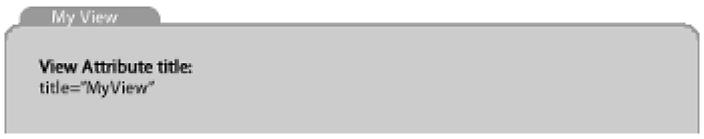


The ***project*** element can contain multiple ***view*** elements, each one displayed as a separate tab in your project, each one containing elements itself, such *resources*, *controls* and *displays* (each box in the drawing represents an element).

Attributes and Values

The attributes control the details of how an element will look or what it will do. If the element was a person, its attributes would be eye color, weight, gender, etc. An attribute is always paired up with a value, for example, *eyeColor* with “blue” and *weight* with “98.”

For example, the ***view*** element, aside from containing other elements, has an attribute called *title*, which causes the value assigned to it (in this case “My View”) to be written in the tab:



View Attribute title:
title="MyView"

Script Text

An element can contain some text that is unstructured. This text is used by VisualEyes to hold GLUE scripts and also for the content in text displays.

Project Sampler

The Project Sampler is a VisualEyes project that demonstrates a number of the displays and techniques. Each is in its own ***view*** and many have a 5-10 minute screen cast walking you through the script. If you are interested in using one of the ideas in the Project Sampler, having working examples to play with makes it easier to create your own.

There is a link to the Project Sampler on the main VisualEyes home page and you can pull in the script into your own VisEdit space by selecting the *Load from ID* option from the File menu and typing "sampler" (no quotes) in the box. The sampler script will be loaded as your own project to explore and edit as you want.

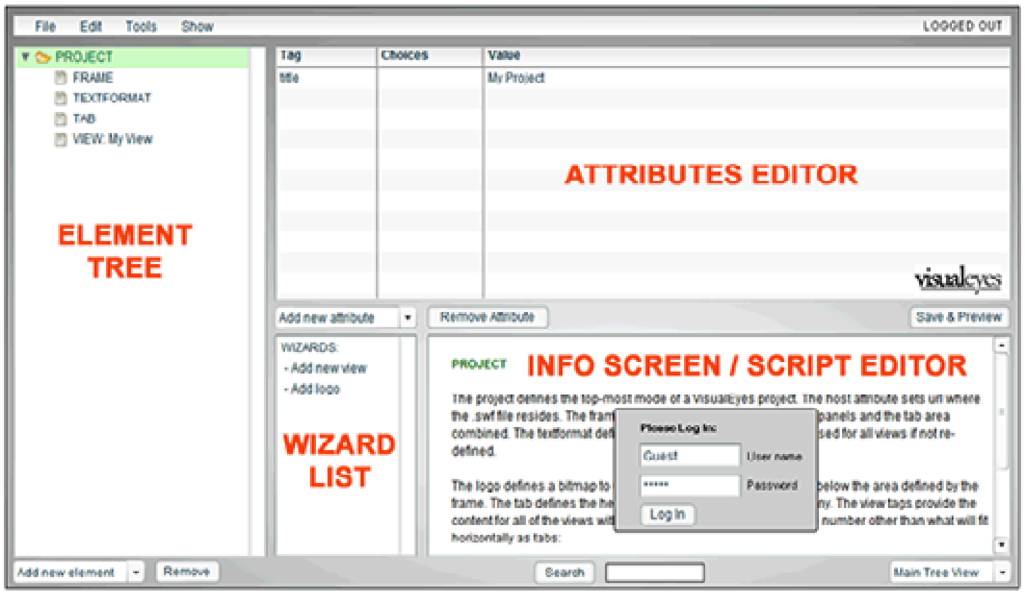
A note on the formatting in this guide

- ***elements*** appear in **bold**/*italic.*
- *attributes* appear in *italic.*
- The red star * next to an attributes description indicates that it is a required attribute.
- If you do not specify an attribute that has a default value, it will retain that default value.
- GLUE methods have parentheses after name().
- WIZARD indicates that a wizard is available to make this element.
- SAMPLER indicates that an example is available in the project sampler.
- **Try it out:** The yellow boxes provide tips on how to experiment with using the elements shown.

Not all of the available attributes available for a given element listed in this guide. Check the VisualEyes XML Reference Guide for the complete list.

Getting started with VisEdit

VisEdit is a browser-based interface that allows you to build VisualEyes projects without typing in all of the XML code that VisualEyes uses internally to make the projects. It can be accessed here. VisEdit 's screen is divided into four frames, each dedicated to a specific purpose:



The project defines the top-most mode of a VisualEyes project. The host attribute sets url where the .swf file resides. The fram panels and the tab area combined. The textformat def sed for all views if not re-defined.

The logo defines a bitmap to below the area defined by the frame. The tab defines the he ny. The view tags provide the content for all of the views wit number other than what will fit horizontally as tabs:

1. The **element tree** shows all the elements that make up your project and provides tools to add or delete them.

2. The **attributes editor** displays the attributes and values that go with the currently selected element, allowing you to add, delete, or modify them.

3. The **info screen/script editor** shows instructional help related to the item you currently have selected and also provide an an area to edit GLUE scripts and text displays.

4. Finally, the **wizards list** provides wizards that walk you through the creation of certain elements, step by step.

Creating an account and logging in

You can use VisualEyes by using the guest account. Without modifying the user name or password, just click 'Log In'. You will be able to edit and save a "communal" account, but if someone else saves something, your work will be lost.

To create your own personal account, where you can permanently save projects, click on the link below the UVa logo and fill in the information requested. Then, instead of using "guest" and "blank", use your new user name and password.

Across from the menus, you will see a user status message. In the screen shot above, this message is 'LOGGED OUT'. When you log in, it will display your user name, and, if you are in a project, the project name and numerical id.

Navigating the element tree

The **element tree** is where you add, select and remove elements from your project and becomes the main way you work with your project. When you click on an element it will turn green. If the element contains other elements within it, you can see show those elements by clicking on the arrow to the left - opening the element's folder. Clicking on the arrow again will close the folder. Clicking on an element selects that element as the current element to:

- Show whatever attributes added to that element in the **attribute editor.**
- Show a description of the element on the **info screen.**
- Show possible attributes and their default values that on the **info screen**.
- Show any wizards for adding new elements in the **wizards list**.
- Add elements that can be added to this element in the **Add new element** button.
- Allow it to be deleted by clicking the **Remove** button.

Navigating the attribute editor

The **attribute editor** shows the attributes already added to the currently selected element. Each attribute appears on its own line, showing the name, possible values, and the actual value set. Depending on the kind of attribute, the options column will display one of three ways:

1. If setting a text or numeric value, it will be blank and the values column is used to type the value directly.
2. If setting a color, the options column will be filled with that color. Clicking on it brings up a color picker dialog. The color's RGB value appears in the values column and can be edited directly.
3. If there is a list of options, you can select one from the drop-down button. The value can also be edited directly in the value column.

The **Add new attribute** drop-down button adds new attributes to the element. The **remove** button removes the currently selected attribute from the element.

File, Edit and Tool Menus

VisEdit has a menu bar across the top containing options similar to desktop applications to save and load project, undo/redo, and access to tools:

FILE

- **New -** start a new project. When you start a new project the screen will be very bare bones. Creating a new project creates a new project ID number. This ID is how each project is stored.
- **Load Project -** a list of available projects is visible in the Information Box. Click on the project you want it will load, replacing your current one.
- **Load by ID** - a box will pop up, you enter the id of a project and it will load.
- **Save** - saves the current project to the VisualEyes server.
- **Save As -** saves the current project with a new project ID and loads that project.
- **Save version -** saves the current project with a new project version for safe keeping.
- **Show my Data Files** - Shows a list of your data files
- **Preview -** shows you what your project looks like without saving any changes. Click the **Return to Editing** button at the top of the page to get back to VisEdit. You can also reach this screen by clicking the **Save and Preview** button.

EDIT

- **Undo/Redo -** your ability to undo and redo actions is almost unlimited. Every time you undo an action your project returns to its last version. Redo redoes your most recent undo.
- **Copy/Paste -** you can copy and paste any element. This would be useful if, for example, you wanted to create more than one tab with similar attributes. The element and any sub-elements contained within its folder that are copied and pasted will be added on the same level of the tree as the element you copied. The pasted element will sit just below the element that is currently highlighted.
- **Move Up/Down -** click on an element or attribute you wish to move and click either Move Up or Mode Down. The element and any elements contained with in it will move up or down in the tree remaining on its same level of the tree. If you are moving around view folders, the order in which they are shown in the element (top to bottom) tree will be the order they are shown in the project preview (left to right).

TOOLS

- **Convert data/AI file to XML** -Converts/uploads files to the server.
- **Upload CSV file to server** -Uploads CSV files to the server.
- **Upload KML file to server** -Uploads KML files to the server.
- **Upload local XML as a project** -Uploads XML project to the server.

See the Appendix for more information about options in the tools menu.

Using Wizards

Wizards are a feature built into VisEdit that walk you through the creation of certain elements and attributes step by step. Click on an element or attribute that has a wizard feature for example in the main project folder you are given two options for using a wizard. If you click once on the Add a logo wizard the screen changes to the wizard format. In the top section of the screen is the name of the wizard and an option to press the **Quit** button to return back to the Main Tree view. You can quit at any time before the wizard sequence is complete to cancel the action.

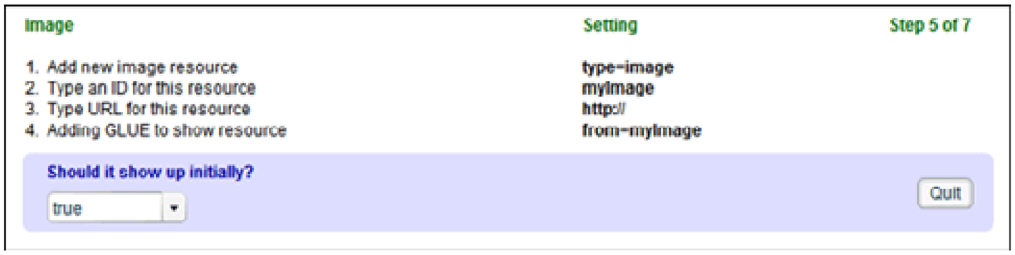


Image	Setting	Step 5 of 7
1. Add new image resource	type=image	
2. Type an ID for this resource	myImage	
3. Type URL for this resource	http://	
4. Adding GLUE to show resource	from=myImage	

Should it show up initially?

true

In the bottom right hand corner are two buttons to help you navigate through the wizard, they are the Previous Step and Next Step buttons. Clicking the **Next Step** button, or hitting the **Down-Arrow** key will start the wizard actions. Each step requires you to make a decision, add a number, add a name, or add a link from a URL or to a data source. Hitting the **Previous Step** button or **Up-Arrow** key lets you go back and make changes.

When the wizard has completed its sequence an option to click the **Add** button will pop-up to bring you back to the Main Tree View. You can now view the changes you made by clicking the Preview option. If you want to make adjustments or remove the logo you just added those changes can be made in the Elements and Attributes Navigators. Wizards are context sensitive and not available for all elements and attributes.

Viewing your project

To view your project at anytime you can click the “Save and Preview” button under the Attributes Editor or you can simply preview without saving by clicking Preview in the File pull-down menu at the top of the screen. Viewing your project will show you how viewers will see and use your project.

A NOTE FOR CHROME USERS: For some reason, known only to Google, your project will not appear sometimes when the "Save and Preview" button is clicked. It is loaded, but you need to click around the screen to have it show up. In addition, the Mac versions of Chrome inhibit the copy function (the CTRL-C key), so as much as Chrome is a great browser, it might be better to used Firefox or Safari when editing projects.

Creating VisualEyes Projects

Making a VisualEyes project involves gathering web accessible resources and creating a script using the VisEdit tool to weave them into a presentation. You need to identify resources you want to use, such as images, movies, and data sets, and put them on the Internet so they can be accessed from anywhere. These can be stored on your own server, or a number of free or inexpensive services can be used:

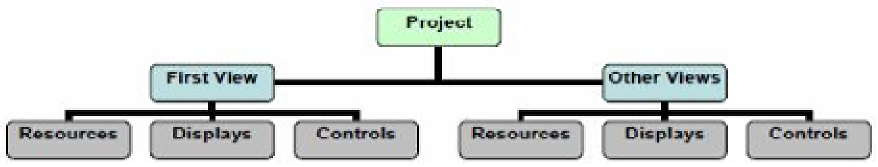
- Images can be easily stored on sites such as Flickr or Picassa
- Videos can be stored on YouTube
- Data files can be stored on Google Docs, or Dropbox.com

You will need to create an account using VisEdit where your project scripts and data files can be stored on our servers. The primary purpose of VisEdit is to guide you through creating a *project script* by allowing you to:

1. Add and/or modify elements to your project script,
2. Save the revised script to the UVa server,
3. Display your project using that script,
4. Rinse and repeat...

The structure of a project

You can think of the ***project*** element as the main folder or tree-trunk, and the elements it holds as sub-folders or tree-branches. The project contains a number of sub-folders called *views*. Each view manifests itself as a tab in your project. Each view can hold elements such as resources, displays and controls:



The most basic elements of a ***project*** are created when you start a new project in VisEdit. These are ***frame***, ***textformat***, ***logo, tab*** and ***view***. These elements are required for all projects and can be customized as needed by adding or changing attributes. Many attributes have default settings, so if you do not explicitly set them, their default values will be used

NOTE: Not all of the available attributes available for a given element listed in this guide. Check VisualEyes XML Reference Guide for the complete list of available attributes.

FRAME ELEMENT

Frames are used to define rectangular areas on the screen or size displayed objects. The ***frame*** element is used to set sizes for many elements. The attributes you set in the ***project*** element's ***frame*** element set the bounds of the *views* for the entire project. Some common attributes are:

Attribute	Description	Default
backCol	*Color of background as an RBG hex number*	*0xffffff*
corner	*Radius of corner of frame for rounded rectangle*	*0*
frameCol	*Color of frame as an RBG hex number*	*0x000000*
frameWid	*Width frame in*	*0*
hgt	*Height of frame in pixels**	
wid	*Width of frame in pixels**	

TEXTFORMAT ELEMENT

The ***textformat*** element controls the basic look of text throughout a project. The attributes of the ***textformat*** element therefore apply to the entire project. This does not mean, though, that all text in a project has to look identical. Rather, the styles are inherited in a cascading effect.

This means, for example, that the view element inherits the ***textformat***'s attributes, but an individual view elements' attributes can also be changed from within its folder to supplement those values specified in the ***textformat*** element. All elements within the edited folder will inherit the edited ***textformat*** element, like a cascade. This cascade continues throughout the folder. The Arial font is anti-aliased and shows fewer "jaggies" than _sans. Some common attributes are:

Attribute	Description	Default
align	*Alignment of text to the screen: left | right | center*	*left*
bold	*Whether or not text is bold: true|false*	*false*
col	*Color of text as an RBG hex number*	*0x000000*
font	*Font face of text: Arial | _sans | _serif | _fixed*	*_serif*
italic	*Whether or not text is italicized: true | false*	*false*
size	*Height of text in pixels*	*12*
underline	*Whether or not text is underlined: true | false*	*false*

LOGO ELEMENT

The ***logo*** element allows you to add an **image** file such as an organizational logo, typically placed below the screen frame that show up in all views.

Attribute	Description	Default
left	*Number of pixels from left of screen**	
source	*File name of logo (including full http:// path and extension)**	
top	*Number of pixels from top of screen**	

WIZARD

TAB ELEMENT

The ***tab*** element controls the look of tabs at the top of each view, including whether you want to have a tab at all (an option if you have only one view in your project). The attributes of the ***tab*** element, should you add it, apply to all the tabs in the project (i.e., one for each view) as a group. When a new project is created, it will contain one ***tab*** element by default. Each project has only one ***tab*** element. If there is no ***tab*** element within the ***project*** element, no tabs will be drawn. Some common attributes are:

Attribute	Description	Default
offCol	*Color of tab when inactive*	*0xcccccc*
offTextCol	*Color of tab text when active*	*0x000000*
onCol	*Color of tab when active*	*0x000000*
onTextCol	*Color of tab text when active*	*0xffffff*
wid	*Width of tab*	*100*

VIEW ELEMENT

The elements above have applied to the project as a whole. A project can contain multiple views, meaning the project will have more than one view element. When a new project is created, it will contain one ***view*** element by default. Each *view* folder becomes its own tab in the project. The view contains other elements that are displayed within that view or tab.

The view element is where the real substance of the project lies. Each view will have resources such as maps, images and data. Displays can also be added as means of showing data and information to the user. Displays can include text displays, paths, graphs, concept maps, timeviews and more. They connect the raw data from tables into compelling representations that users can interact with through controlpanels (timelines, animation players, control panels, and more). The scope of any view is itself. This means each view is an island unto itself. Some common attributes are:

Attribute	Description	Default
pan	*Allow panning of screen: true | false*	*true*
title	*Name of the tab**	

WIZARD

If your image is larger than the ***view***'s ***frame*** and *pan* is set to true, your user will be able to click and drag the image to see parts beyond the frame size. If *pan* is set to false then the image will be fixed in place, unless you are zoomed in using a zoom control.

Try it out: The skeleton project
The four elements nested within the ***project*** element (***textformat, frame, tab, and view***) are used in every VisualEyes project. Experiment with adding and changing various attributes and adding new views. Your work-flow should be:

1. Try something
2. Save and Preview to see how it looks
3. Go back to step 1

Resources

Resources contain information to be used by VisualEyes. This information can be a table of data, a vector map, text, images, animation, movies, audio, charts, and graphs. Resources are the raw material for VisualEyes views. We give the resources names (called *ids*) so we can easily provide access to them by other elements such as GLUE scripts.

All resources must be web-accessible, meaning they can be loaded into a browser by providing the URL address as the *src* attribute. Being web-accessible means you could type that URL into a browser and it would display. The various types of resources fall into four basic categories:

1. **Image resources** that point to pixel-based (aka raster) images such as photographs and drawings in JPEG, PNG, or GIF formats.

2. **Movie Resources** that point to movies and audio, loaded from a website or YouTube.

3. **Map Resources** where vector maps and drawings imported from ArcGIS, Adobe Illustrator and other graphics packages are loaded from a website or the VisualEyes server.

4. **Data Resources**, where numeric and/or string data organized into tables is loaded from a website, Google Docs, or the VisualEyes server.

Common resource attribute tags

There are some attributes that are common to *all* resources:

- The *id* attribute* provides a way to uniquely identify the resource to other elements in a view. The scope of any resource is within the view it is contained by.

  NOTE: IDs are required and must be a single word with no spaces!

- The *type* attribute* defines the type of resource (i.e. map, image, data, etc.).

- The *preload* attribute allows you to control whether or not a your project will start before this resource is loaded. By default, this is set to true, meaning a spinning wheel will appear while that resource is being loaded.

IMAGE RESOURCE

VisualEyes allows you to load an image file (.jpeg, .gif, or .png) from any valid URL. The URL address goes in the *src* attribute. The URL could belong to an image on another web page, or in some sort of online storage database, such as Picassa and Flickr. See the Appendix for information on importing these images from these sites.

By default, the specified image is automatically sized to fit in the window, positioned in the top left corner. If much larger than the frame size, the image can be panned and zoomed. You can also specify the position or size of the image. Any number of images can be layered.

Image resources can be geo-referenced, meaning you can identify points on them using latitude and longitude, rather than pixel coordinates. See the section on geo-referencing in the Appendix for more information.

Attribute	Description	Default
id	*ID of resource**	
src	*Source URL of image file (with http:// part)**	
type	*Type of resource - must be image* *	

WIZARD SAMPLER

Try it out: Adding an image resource

- Click on a view in the Element Tree.
- Use the wizard feature to add an image.
- Give your image a *one-word* id. It can be any name you wish and will be most helpful if the name describes what the image is.
- It must have a source, as noted above the source must be a URL of a .jpeg, .gif, or .png. You can use this image to test: *http://www.viseyes.org/chocolates.jpg*

MOVIE RESOURCE

VisualEyes supports two formats for playing digital movie clips, FLV Flash video clips on the web, and clips from YouTube. You can use the wizard, or a *resource* element with its *type* attribute set to "movie" to create a movie resource in a view. As with an *image* resource, the *src* attribute is the full URL (with http://) where the movie sits on the web.

The *movie* resource requires a *frame* element within it to define the size, position and look of the player that encloses raw movie. Each time you add a resource that needs a frame (using the Wizard), you get a frame element added to that resource. In other words, the frame that controls the project view is separate. The *hgt* and *wid* attributes set the width of the video, and an 8-pixel border is added around it on all sides. Omitting the *wid* attribute will cause movie and player to size itself to match the native resolution of the movie. A slider and play button will be drawn below the video increasing the total height by 20 pixels beyond the borders.

The *autoPlay* attribute determines if the movie will play when it first appears, and *autoRewind* will rewind the movie when it's done if set. *start* and *end* specify the movies bounds to start and stop, in milliseconds. See the Movies section in the Appendix for more advanced features of movies.

Using movies from YouTube

To use a video clip from YouTube, put the YouTube movie id, such as “SzSwnbxb9TY” as the *src* attribute. You can find the movie id by looking at the video’s URL and copying the part after “v=” and before a “&” or the end, http://youtube.com/?v=SzSwnbxb9TY would yield an id of “SzSwnbxb9TY.” **NOTE:** YouTube videos must be made "public" in YouTube to be played in VisualEyes.

Attribute	Description	Default
autoPlay	*Play movie/sound when loaded: true | false*	*false*
autoRewind	*Rewind movie/sound when finished: true | false*	*false*
close	*Has close button: true | false*	*false*
end	*Ending time of movie (in ms)**	
id	*ID of resource**	
src	*Source URL (with http:// part unless its a YouTube clip)**	

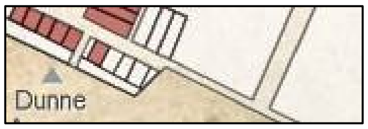
Elements: ***frame****

WIZARD SAMPLER

Try it out: Adding an image resource

- Click on a view in the Element Tree.
- Use the wizard feature to add a movie.
- Play around the ***frame*** options to see how they change the player.

MAP RESOURCE



Maps are an important element in many VisualEyes projects. While images of maps are often used, especially historic maps, it is useful to have some maps that use vector drawing techniques (i.e. drawn using lines and polygons rather than a collection of pixels in an image).

Vector maps have three primary advantages in VisualEyes:

- The lines and shapes do not get blurry, no matter how far you zoom into them like images do, unless those images are very high resolution.
- You can dynamically color the interior and exterior edges of individuals shapes in a map (called features). Individual features can be things like counties or states. If you use an image map, you can only effect the map as whole.
- You can react to clicks on the map's features and cause GLUE elements to perform actions. You would need to set up dots on a path element to by hand to make an image map click-able.

The map resource shares most of the same attributes and capabilities as an image resource, but instead of specifying the URL of a still image file, you provide the URL of an XML file. That file is a list of all the lines and polygons to be drawn. We can import vector data from programs such as arcGIS, Adobe Illustrator, and any program that can output an Illustrator .ai file. See the section on converting AI files in the Appendix for more information.

Individual features of maps can be made interactive and cause other actions when clicked or hovered over. Often Paths and Dots (see below) are used in conjunction with maps to show change and information over time and space. The Paths and Dots can be linked to a timeline (see below). See the section on Interactive Maps in the Appendix. Map resources can be geo-referenced, meaning you can identify points on them using latitude and longitude, rather than pixel coordinates in things like dots. See the section on Geo-referencing in the Appendix.

NOTE: A map resource needs to be refreshed in order to be drawn. To do this, add a line to a GLUE script that says: refresh(myMap), where "myMap" is the id of your map resource.

Attribute	Description	Default
id	*ID of resource**	
src	*Source URL of vector shape data file (with http:// part)**	
type	*Type of resource - must be map**	

WIZARD SAMPLER

GMAP RESOURCE (GOOGLE MAP)

Embeds a Google map in VisualEyes. The map can be attached to the screen base, much like an ***image*** resource can be, or as a floating window using the *depth* attribute.

Attribute	Description	Default
depth	*If resource is bound to screen: screen | topMost*	*screen*
dim	*Dimensionality: 2D | orthogonal | 3D*	*2D*
frameCol	*Color of frame as an RBG hex number*	*0x000000*
frameWid	*Width frame in pixels*	*0*
hgt	*Height in pixels*	*500*
id	*ID of resource**	
lat	*Latitude to center map*	*38.14*
left	*Number of pixels from left of screen*	*0*
lon	*Longitude to center map*	*-78.45*
maptype	*Type of map at start (0=map, 1=hyb, 2=sat, 3=ter)*	*0*
overview	*Show overview navigator? true | false*	*true*
top	*Number of pixels from top of screen*	*0*
type	*Type of resource - must be gmap**	
typemenu	*Show map type menu? true | false*	*true*
wid	*Width in pixels*	*800*
zoom	*Show zoom control? true | false*	*true*

XML RESOURCE

In its simplest form, an XML data resource is a list of things: numbers, words, paragraphs, URLs, etc. We call it an XML resource but the raw data can be brought into projects from many formats in addition to XML; comma delimited (.csv) files, tab delimited (.txt), , or directly from a published Google Docs link (see section in the Appendix).

For more information, see the section on Using Structured Information in the Appendix.

Each of these three methods stores the data into an identical format with VisualEyes, as a list containing the elements, referenced from the resource's name. Data brought in this way can be queried using the GLUE query() method or used in GLUE methods that require table data. Data resources that are brought in by these methods share

Attribute	Description	Default
id	*ID of resource**	
src	*Source URL of data file (with http:// part)**	
type	*Type of resource - must be xml**	

WIZARDSAMPLER

Try it out: Adding an XML resource

- Click on a view in the Element Navigator.
- Use the wizard feature to add a XML resource.
- Here is a link to a simple data source: http://www.viseyes.org/SampleTable.csv
- Add a ***GLUE*** element , add a *script* and *init* (set to true) attribute to the ***view***
- Add a status() method to the GLUE's *script* display data from the table: *status(*myData.name)*
- This will show all the names at the bottom of screen when run.
- Change the status() method to*: status(*myData.age.2)*
- This will show all the third age bottom of screen when run. (remember, computers start counting at 0!)

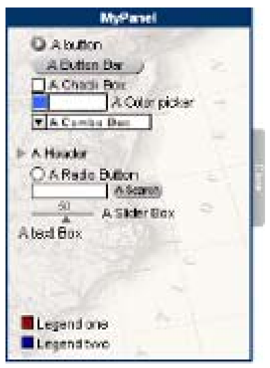
TABLE RESOURCE

The table resource element creates an empty table with whatever fields you want, rather than importing CSV or XML files. This is useful when you want to make a table from a subset of some elements in an imported file. See the Table Resource section the Appendix for more information.

Controls

Controls put the interactive in interactive visualizations and provide opportunities for users to explore the project. These controls include control panels the contain items such as radio buttons and check-boxes to turn on and off project elements, timelines to set the current date, animation players to control animated sequences, and zoomers to zoom into parts of the screen.

CONTROL PANEL



Control panels provide an interactive dialog box-like means for allowing the user to choose what resources are displayed on the screen within parameters set in the project. Control panel items can include: check boxes, radio buttons, combo selection boxes, sliders, text input, and buttons. A user clicks on any of the control panel items to cause some sort of action. For example, clicking on a radio button in your control panel changes a street map resource from 1900 to 1950 and a check box allows the viewer to choose whether they want topographical information layered on top of the street map. Items typically cause some action by adding an id of a GLUE element to call when they are changed or clicked.

A ***controlpanel*** element contains a frame and a series of ***items*** that users click on or drag on to cause some change.

Attribute	Description	Default
title	*Name of the control panel as it appears in header*	
closable	*Control panel has closing button: true | false*	*true*
open	*Control panel is open on start up: true | false*	*true*

WIZARDSAMPLER

Control Panel frame

The ***frame*** specifies the size, position and color of the controlpanel. The *docking* attribute in the ***frame*** can be top, left, bottom, right, or float. The first four cause the ***controlpanel*** to “stick” to that side of the screen, ignoring the positioning parameters (*top* and *left*) set in the frame. The setting *docking* to float will allow the controlpanel to be positioned anywhere on the screen as dictated by the *top* and *left* attributes.

Attribute	Description	Default
backCol	*Color of background as an RBG hex number*	*0xffffff*
docking	*Docking mode: left | right | top | bottom | float*	*float*
frameCol	*Color of frame as an RBG hex number*	*0x000000*
frameWid	*Width frame in*	*0*
hgt	*Height of frame in pixels**	
left	*Left of frame in pixels*	
top	*Top of frame in pixels*	
wid	*Width of frame in pixels**	

Control Panel Items

The ***items*** form the active portion of the ***controlpanel*** as a running list of controllers such as check boxes, radio buttons, combo selection boxes, sliders, text input, and buttons, chosen by setting the *type* attribute to one of the following:

• ***backbuton***	A round button with a **<** that will trigger glue method when clicked
• ***buton***	A round button with a **>** that will trigger glue method when clicked
• ***buttonbar***	A square button with the title written inside that will trigger a glue
• ***checkbox***	A checkbox with the title to the right
• ***color***	A color chip to choose a color from a set of choices,
• ***combobox***	A combo box to choose between several choices
• ***half***	Used to add a half-space vertically (leading) to the list
• ***header***	An arrow control to collapse or expand the items to follow
• ***legend***	Used to put a color choice when drawing legends
• ***line***	Draws a separator line
• ***query***	Adds a query line (if: something equals value)
• ***radio***	A radio button, of which only one is active in a contiguous group
• ***search***	A text input box with a search button bar attached
• ***slider***	A horizontal slider to set the value from 0-100
• ***text***	Displays a line of text
• ***textbox***	A text input box

They are drawn using the current font settings, but the bold and italic can be over-ridden on a per-item basis by setting the ***item***'s *bold* and *italic* attributes to true. The current leading in the ***textformat*** element determines how far they are spaced vertically. If there are more ***items*** than fit vertically, a new column is started.

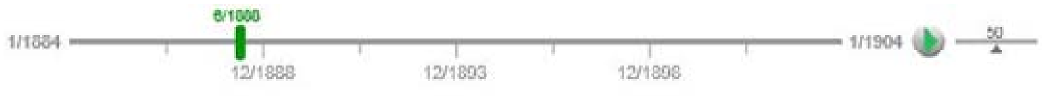
Radio buttons act a a collective group, allowing only one radio on at a time. Header items will collapse items below it (until the next header item). Legends provide a box colored by the def attribute, and are stacked from the bottom of the ***controlpanel*** up. Setting the *glue* attribute of a checkbox item to "legend" will cause the legends to collapse if checked

ComboBoxes are useful items that contain a series of choices in a drop-down menu. When an option is selected, its name is set as the current value, just like a 0 or 1 is set with a checkbox. You set the values in the title attribute, separated by | marks in the *title*, like this: "Yes|No|Maybe so".

Attribute	Description	Default
bold	*Whether or not text is bold: true | false*	*false*
def	*Default value for item on startup: true | false*	*false*
glue	*GLUE to be called by item (with optional data)*	
italic	*Whether or not text is italicized: true | false*	*false*
labCol	*Color of text label*	*0x000000*
title	*Title that appears by control item**	
type	*Type of control (see list below)**	

TIMELINE

Each view can have its own ***timeline*** element that can control the temporal aspect of the project. Sliding the cursor changes the view's date, which in turn can change the way in which information is displayed if it is time dependent.



The timeline must be linked to dates. The *min*, *max* and *start* attributes define the minimum, maximum and starting dates for the timeline, and can be expressed as "year", "month/year" or "day/month/year." Dates can be formatted as years (1976), month/year (3/1856), day/month/year (3/7/1756), or full date (January 6, 1798).

Paths and Dots on your image can be linked to the Timeline. The attributes of the dots must include dates using the same format as the Timeline *max* and *min*. The timespan can be divided by 4 major tick marks, whose length is set by *majorTick* and 3 minor tick marks set by minorTick. The tics can be place above, below and across the main timeline bar by setting tickPos. The *showValues* and *showMinorValues* attributes determine if the tick mark's time values are displayed.

The *speed* attribute controls how fast an animation will play. The default is 100, meaning that it will play 10 seconds with the speed slider in the middle. Setting it lower, say to 20, will make the timeline play in 50 seconds, and 50 will play 20 seconds, etc.

Attribute	Description	Default
dateFormat	*date format: yr | mo/yr | dy/mo/yr | mo/dy/yr | mo,dy,yr*	*yr*
min	*Starting time of the timeline in any time format**	
max	*Ending time of the timeline in any time format**	
minorTick	*Minor tick make length in pixels*	*0*
numTicks	*Number of major ticks*	*4*
play	*Show play button: true | false*	*true*
showValues	*Show values with major tick marks: true|false*	*false*
speed	*Speed of playback from 1-100*	*50*
start	*Initial time of the timeline in any time format on startup*	

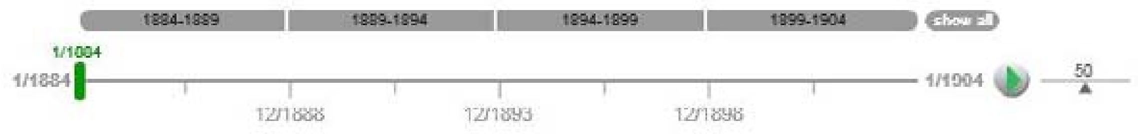
Elements: ***frame*, item*, labels, textformat, timebar***

WIZARD SAMPLER

Try it out: Add a timeline

- Click on a view in the Element Tree.
- Use the wizard to add a timeline.
- Set the values for your starting and ending dates.
- Try playing around with the timeline's attributes to see how it affects the way the timeline works or looks.

Punctuated time bar



You can add a segmented bar that will control the span of the overall timeline (called a punctuated time bar) by adding a timebar to the timeline. The ***timebar*** consists of a series of segments (must be contiguous!) that contain a starting and ending date within the overall *min* and max *set* in the timeline. The ***timebar*** provides a way to make the timeline's span to be a portion of the full time being represented for better control and slowing animation of short events within the larger time-frame.

Attribute	Description	Default
all	*Add a show all segments button: true | false*	*true*
equal	*Make all segments equal widths: true | false*	*false*
glue	*GLUE id to be called when all button is clicked*	
hgt	*Distance of segments from main timeline*	*6*
offCol	*Color of inactive segment as RBG*	*0x999999*
offTextCol	*Color of inactive segment text as RBG*	*0x444444*
onCol	*Color of active segment as RBG*	*0x999999*
onTextCol	*Color of active segment text as RBG*	*0xffffff*

Elements: ***segment****

A *glue* attribute can be specified to cause some GLUE to trigger when any segment is clicked. You can append parameters to the glue's name, which will be available as variables in the GLUE element.

For example, if we wanted to pass the start and end time to the *glue* attribute, we would spec the *glue* as: newTime?2&8, the value of 2 would be set in $$click variable and the value of 8 would appear in the $$param variable within the newTime GLUE element.

Segment

When a segment is clicked, the ***timeline***'s *min* and *max* settings are set to the segment's start and end attributes, making the timeline's span smaller. The screen is updated as if you had dragged the slider to the segment's start date. A button to make the timeline extend from the first segment to the last can be added by setting the all tag to true.

Attribute	Description	Default
end	*Time of the segments end in any time format**	
glue	*GLUE id to be called when segment is clicked*	
start	*Time of the segments start in any time format**	
title	*Text to be displayed in the segment**	

Animation Player

Timelines by default have an animation player button that will cause the timeline to play on its own, as if one was dragging it. The speed of the animation can be controlled by setting the slider bar next to the button faster or slower. This speed can be controlled setting the ***timeline*** *speed* attribute. It defaults to 50. Setting it lower slows the speed, and higher increased the speed. This feature can be turned off by setting the ***timeline*** *play* attribute to "false". You can also show only the animation player without a scroillable timeline to its left by setting the ***frame*** element's *wid* attribute to 0.

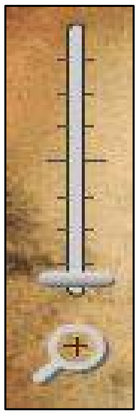
Timeline Labels

Labels signifying specific dates can be added using a labels tag. What direction they emanate from the main bar is set by *pos*, and the distance away from the main bar is set by *offset*. Lines can connect the label to the main bar by setting the lines attribute. The ***textformat*** will use the timeline's text formatting as its basis and elements can be over-ridden. The actual entries are set using the timelinelabels() GLUE method.

Attribute	Description	Default
lines	*Show lines to labels: true | false*	*true*
offset	*Distance from time bar to labels in pixels*	*8*
pos	*Position of labels relative to the main bar: top | bot*	*bot*

Elements: ***textformat***

ZOOM CONTROL



Zoom controls, overviews and magnifier boxes allow you zoom into the screen in much the same way that Google Maps allows you to zoom in on maps. If the screen has a high-resolution or vector-based image, zooming in will yield a narrowed, but sharp view of the screen.

The ***zoomControl*** is a vertical slider for setting the degree of magnification. When the slider is dragged to the bottom, the entire screen will be shown, and when dragged to the top, the screen will zoom in until the maximum set by the *max* attribute. By default, this is set to "3", meaning the screen will zoom in 3 times or 300%, but it can be set to any number from 1 to 10. The screen can be panned by dragging it left or right to bring the desired zoomed-in area into view.

You can attach a magnifier icon that will allow you to draw a box that both sets the magnification factor and pans the area into view by setting the *magnifier* attribute to "true."

Attribute	Description	Default
left	*Horizontal position of zoom control*	*none*
magnifier	*How magnifier control: true | false*	*false*
max	*Maximum zoom allowed (1-10 times)*	*3*
top	*Vertical position of zoom control (in pixels)*	*none*

WIZARDSAMPLER

Try it out: Add a Zoom Control

- Click on a view in the Element Trtee.
- Use the wizard to add a Zoom Control.
- Try playing around with the zoom control's attributes to see how it affects your magnification possibilities.

OVERVIEW

Overviews are small insets to the side of the screen that contain an image with a movable box that represents the screen area that can be dragged to pan the current magnified portion of the screen into view.

The overview can dock itself to any corner of the screen using the *docking* attribute and the *wid* attribute sets its horizontal size. The vertical size is set according to the shape of the background image. The *src* should be set to an image (.jpg, .gif or .png) to use as the background of the overview, and it will be stretched to fit the space. Using the same image as you used for the screen is fine.

Attribute	Description	Default
boxCol	*Control box colo*	*0xffff00*
docking	*Docking location*	*botLeft*
wid	*Width of overview*	*100*
src	*URL of image (full path with http://...)*	*none*

WIZARDSAMPLER

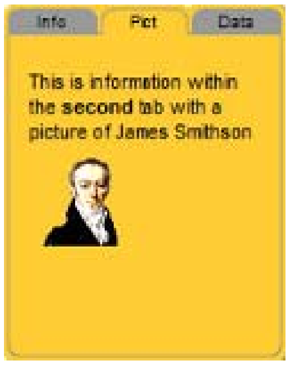
Try it out: Add an Overview

- Click on a view in the Element Tree.
- Use the wizard to add an Overview into an element with an image.
- You must have an URL image source in .jpg, .gif or .png form. This should be the same image as the one used for the background in this folder.
- Try playing around with the overview's attributes to see how it affects your dragging and panning possibilities.

Displays

Displays provide the means of showing data and information within your project in the form of pop-up information boxes, charts, widgets, paths and other displays. Several kinds of displays use dots, discussed in more detail below.

INFOBOX


Info
Pict
Data
This is information within the second tab with a picture of James Smithson

Information boxes are pop-up boxes used to display textual information on demand. They are typically called by clicking on path and graph elements. InfoBoxes can contain a variant of HTML formatting and can be populated using a search and replace variable that can be set using a database. As with a movie resource, the frame of the box is controlled by the frame element.

The text can be have special tags($$1 through $$99 that can be replaced dynamically using the replaceword() method. Text can contain the standard text formatting macros (see the Appendix).

Attribute	Description	Default
backImg	*Backgound image URL*	
border	*Border amount in pixels*	*24*
depth	*Should it always appear on top: screen:topMost*	*topMost*
id	*ID of resource**	
position	*Position of box following click: abs|north|south|east|west*	*abs*
tail	*Box tail if following mouse click: line|none|solid*	*none*
title	*Title*	

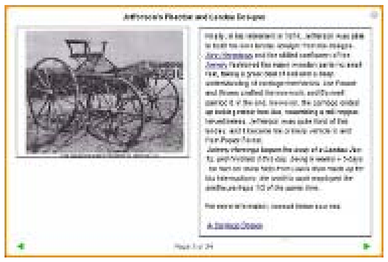
Elements: ***frame****

WIZARDSAMPLER

Tabbed infoboxs

Infobox's can be subdivided into multiple tabbed pages, to make it easier to display larger amounts of information in a smaller space. Each section has its text divided by a header() tag. The name with the tag's parenthesis will show up in the tab.

DOCVIEWER



A ***docviewer*** is a display element very similar an ***infobox*** to that can hold HTML formatted text and a picture side-by-side in series of pages provided by a data source such as an XML file. The data source can have 4 fields: *title, source, desc and caption*. The *title* provides a title at the top and a way to select items from the data source.

The text and picture information is usually supplied by a data resource from a CSV file, and sent to the ***docviewer*** using the filldocviewer() GLUE method, typically as the result of a query.

Items with the same *title* will appear as pages within the document viewer. The *source* gives a url for a picture if desired, and *desc* is an HTML formatted text area. If a *caption* field is defined, it will appear underneath the picture. If both *desc* and *source* are defined, they will appear side by side. If only one is defined, only that one will appear. See the section in the Appendix on “Formatting docviewers” for more information on creating the contents of docviewers. Text can contain the standard HTML formatting macros

Attribute	Description	Default
border	*order amount in pixels*	*24*
close	*as close button: true | false*	*true*
id	*ID of resource**	

Elements: ***page***

WIZARD SAMPLER

Page

You can also add content to the ***docviewer*** with a series of embedded ***page*** elements. Each ***page*** element contains a page, with title, image, and text being provided by the *title, src,* and *desc* attributes in the page element.

Attribute	Description	Default
caption	*Image caption*	
desc	*Text for description page*	
src	*Source URL for image*	

PATH

Paths place dots on the screen and can be connected by lines if desired. The width, color, and alpha can be specified. The position of the dots is set in pixels, relative to the base resource the path is atop, or in lat / lon coordinates based on the base resource.

If *showAllDots* is set true the dots are not time dependent. If false, dots can have times associated with them, so they will appear when the view's timeline date reaches a certain time. The time or date attribute of a dot tells when that dot will be drawn. 0 is at start, .5 is middle, 1 is end, etc. If a date is set, dot will appear when its date matches the date on timeline. Setting the *end* attribute will turn off a dot at a particular date. Time dependent paths can have the line advance between dots as the time changes by setting *tweenLines* to true. The current time of the line can be preceded by an icon by setting the *head* attributes. Paths are useful in showing a trail on a map, but are often used to put buttons, menus and other navigational elements on the screen. Set *showAllDots* to true, so they will always appear

If you have a number of journeys along a set number of path ways, you can define a collection of dots as a pathway. The timing within the path is relative from 0 to 1 start to end, rather than a particular date for better flexibility and accomplished by setting the *pct* attribute in the dots contained in the pathway. That pathway can be drawn multiple occasions and different times by adding route elements to a path that define the start and end times a particular pathway will be drawn.

The front of a path can have an icon or image leading it by setting *headStyle* to an icon or image name. You can color the head icon by setting *headCol* to a color. Clicking on a head will cause a GLUE element to run if one is specified, allowing you to trigger other actions and displays. You can find out which head was clicked by looking at the $$param global list parameter, which will be set to its index view (0 is 1st path, 1 is 2nd, etc.). Alternatively, you can set this value manually by appending it the glue's name with a ? mark, such as: **`glue=myGlue?show me`**. This will cause the words "show me" to be set in the $$param list and available for use in the glue script. The first head in the first path in a ***route*** element will set $$param to 1000, the second to 2000, etc).

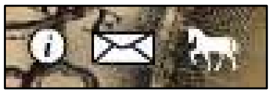
Attribute	Description	Default
col	*Color of line, as an RBG hex number*	*0x00ffff*
headCol	*Color head as an RBG hex number or -1 for no color icon*	*-1*
headEnd	*Leave head icon up at end of path: true | false*	*false*
headSize	*Size of head icon in pixels*	
headStyle	*Image shown at head of path (icon: | .gif | .jpg | .png | .swf)*	
id	*ID of path**	
showAllDots	*Show all dots, regardless of timing: true | false*	*false*
tweenLines	*Animate line between dots based on timing: true | false*	*false*
wid	*Width of line in pixels*	*0*

Elements: ***dot*, textformat***

WIZARD SAMPLER

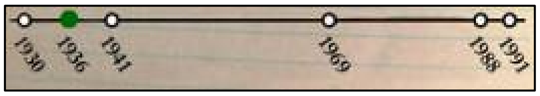
Using paths as selectors and menus

Since each ***dot*** in ***path*** element can have a **GLUE** element called when clicked on, paths are an easy way to create menu bars to select options by clicking on them. By not specifying a *time* or *date* attribute in the dot, it will always be visible.



In this example from the Poplar Forest project, a path containing three icons (the info, letter and an image of a horse) provide an easy way to cause other displays to be drawn when clicked by specifying a **GLUE** element to call:

```
path
        dot  style=icon:info     x=44   y=500   glue=showAbout
        dot  style=icon:letter   x=44   y=500   glue=showLetters
        dot  style=horse.gif     x=44   y=500   glue=showTransport
```



A more sophisticated example is shown to the right, where the dots work as a "radio button", with only one being active at any time, A path containing six year dots are arranged in a line. The default alpha (opacity) for the dots was set to 0 (invisible), but the **GLUE** script causes all the other buttons to be hidden, and the selected one shown:

```
path    id=yearPath
        dot     x=10   y=500   glue=setYear   style=cir    alpha=0
        dot     x=30   y=500
        dot     x=60   y=500
        ...     ...    ...

glue    id=setYear    [script]:   repeat(6)
                                  setdot(yearPath,0,$ix,alpha,0)
                                  repeat(end)
                                  setdot(yearPath,0,$$param,alpha,1)
                                  refresh(yearPath)
```

The path is similar to the first example. Notice that the *glue* and *style* and *alpha* attributes did not need to be repeated in the dots that follow, as they will "inherit" them from the first dot.

The *setYear* **GLUE** element's script does the work here. The setdot() method is used to change the *alpha* attribute of the dot. The repeat() method repeats the setdot() method 6 times, one for each ***dot***, setting the ***dot's*** *alpha* attribute to 0 (invisible), and incrementing the $ix list so each ***dot*** is set in turn. The $$param list is set to the currently clicked dot automatically, so the next line sets the dot's *alpha* is set to 1, making it visible. Finally, the ***path*** element is re-displayed using the refresh() method.

DOT

Dots are an element used by a number of display elements to put information on the screen on top of a resource, such as an image or a map. The basic idea of a dot element is that it:

1. Draws as a graphical element that appears on the screen,
2. In a particular place,
3. At a particular time, and
4. Can call a GLUE method when you click or hover over them with your mouse.

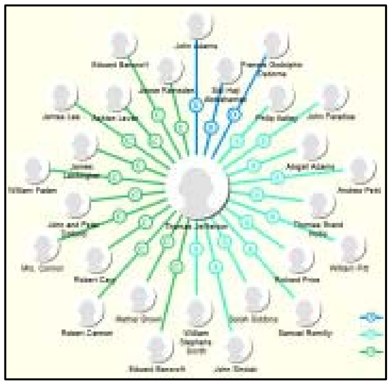
Attribute	Description	Default
alpha	*Opacity as a number from 0-100*	*100*
col	*Color of interior as an RBG hex number*	*0x00ffff*
date	*When dot becomes active (any time format)*	
end	*When dot becomes inactive (any time forma)t*	
frameCol	*Color of frame as an RBG hex number*	*0x000000*
frameWid	*Width frame in pixels*	*0*
glue	*GLUE id to be called if clicked*	
hgt	*Height in pixels*	*0*
icol	*Color of an icon(-1 means leave icon as it was drawn)*	*-1*
id	*ID of path*	
lab	*Labels for dot*	
labelCol	*Color of labels as an RBG hex number*	*0x000000*
labelPos	*Labels position rel. to dot: bot | center | left | right | top*	*bot*
style	*Shape of dot (icon: | | cir | rbar | span | star | triu | trid | tril | trir bar | but ***	
time	*Time dot becomes active from 0-1*	*-1*
wid	*Width in pixels*	*0*
x	*X position of dot*	
y	*Y position of dot*	

** You can also specify graphics images via a url. VisualEyes support images in the JPEG, GIF, SWF, and PNG formats. You must specify the full Internet address: (i.e. style=http://www.mysite.com/mypic.jpg).

Dots will continue using properties set in previous dots to reduce unnecessary repeating of attributes. For example, if you set the style to triu (up-facing triangle), all dots that follow would be rendered as triu until re-specified. See *Persistence of Dots* in the Appendix for more information.

NOTE: Clicking on a dot's graphic will cause a GLUE element to run if there is one specified, allowing you to trigger other actions and displays. You can find out which dot was clicked by looking at the $$param global list parameter, which will be set to the dot’s index in the path. The first dot will set $$param to 0, the second to 1, etc. Alternatively, you can set this value manually by appending it the GLUE’s name with a ? mark, such as glue="myGlue?show me". This will cause the words “show me” to be set in the $$param list and available for use in the GLUE script.

CONCEPT MAP



Concept maps are similar to paths, but the paths can be arranged in a radial manner similar to a hub and spoke shape. The dots are not time dependent, and lines (edges) must be specifically drawn by setting the relationships between the dots (nodes). Labels are automatically drawn if specified underneath the dot. The frame specifies the overall bounds of the concept map.

You can add a legend that identifies the type of lines by including a legend tag. Each tag adds an entry that shows a label associated with each *linestyle*. Setting *backCol* will draw a "wash" of color alpha'd over the background, to help highlight the concept map and separate it from the background.

Attribute	Description	Default
backCol	*Color of wash to blot out background as RBG (-1=off)*	*-1*
col	*Color of line an RBG hex number*	*0x00ffff*
cx	*Center X position in pixels*	
cy	*Center Y position in pixels*	
hgt	*Height in pixels for ovals (omit for perfect circle)*	*0*
id	*ID of map**	
wid	*Width in pixels*	*0*

Elements: ***dot*, line, linestyle, legend***

WIZARD SAMPLER

Lines and linestyles

The lines define the relationship between the dots and determine how they will be placed. Setting the from tag to "" will position the dot pointed by the to tag it at the center of the concept map.

Attribute	Description	Default
from	*ID of dot where line is drawn from**	
style	*ID of linestyle type**	
to	*ID of node where line is drawn to**	

The *linestyle* object defines how the lines will be drawn, and the letter that will be drawn in the middle.

Attribute	Description	Default
alpha	*Opacity as a number from 0-100*	*100*
col	*Color of line an RBG hex number*	*0x00ffff*
id	*ID of linestyle**	
lab	*Labels for line*	
letter	*Letter drawn midway through line in concept maps*	
wid	*Width of line in pixels*	*0*

PICTURE MAP

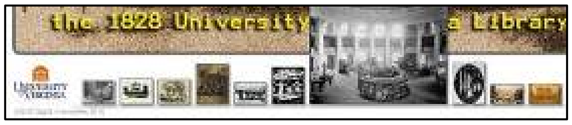
A picture map is kind of concept map that uses pictures that can be rolled over and clicked on to call GLUE scripts. Picture maps are very similar to radial concept maps, but the pictures are not structured around a circle, rather, are placed by setting the dot x and y attributes where you want them. You can add a background image by setting the backImg attribute to an image's url. The frame specifies the extent of the map with the usual background and color options.

With the ***pmap*** element, ***dot*** elements are added that will show up in the frame. As your mouse glides over one of the dots, it will go from the transparency you set in the pmap's alpha attribute to fully visible as you near the center of the image.

Attribute	Description	Default
alpha	*Opacity as a number from 0-100*	*100*
backImg	*Backgound image URL*	
id	*ID of map**	
wid	*Width in pixels*	*0*

Elements: ***dot*, frame*, textformat***

DOCK DISPLAY



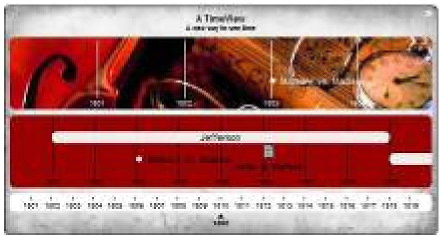
A *dock* display presents a series of dots horizontally across the screen in a similar fashion to the application dock used in the Apple Macintosh. The dots are typically icons or images that are fixed to a base bar. As the mouse hovers over one, it and its neighbors grow by the percentage specified by the growth tag. Setting the *growStyle* to "single" will cause only the dot being hovered on to grow while hovered over, as opposed to the default of "taper", which also grows the two dots on either side of the one being hovered over as well. The dots can have glue attached to cause some action when clicked.. If a *wid* attribute is specified, the number of pictures on the dock will be limited by that number, and green arrows will appear to scroll to show additional pictures.

The frame object sets the bounds of the dock, but since the dock grows and shrinks based on the number of dots within it, the dock will draw from the center of area defined by the frame's *left* and *wid* tags. The frame's *hgt* tag defines the height of the base bar. Setting the hgt to 0 will inhibit the drawing of the base bar.

Attribute	Description	Default
alpha	*Opacity as a number from 0-100*	*100*
id	*ID of map**	
growStyle	*What pictures grown when moused-over: growth | taper*	*growth*
preload	*Load this resource before screen is shown: true | false*	*false*
wid	*Limit number of pix by number*	*0*

Elements: ***dot*, frame****

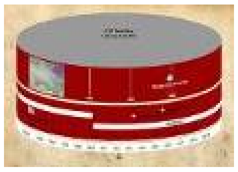
TIMEVIEW



A ***timeview*** resource is a display that shows events that are timed to occur at particular dates. It is similar to a traditional graphic timeline like MIT's Simile. The ***timeview*** can have any number of ***bands***, each one having it's own time scale, allowing you to show images that occur in vastly different scales. All the bands are linked, so scrolling one, scrolls the others. Each image is represented by a ***dot***.

The ***frame*** defines the width, colors, and position of the overall display. The *hgt* attribute is ignored and automatically figured out by adding the heights of the bands in the ***timeview***.

You can set an image background for both the whole display and individual ***band***s by setting the *backImg* attribute to the image's full URL accordingly. The *border* attribute sets the spacing between bands.



Setting the *rot* attribute to something other than "0" will cause the bands to be wrapped around a cylinder in 3D. The cap of the cylinder can be a full oval or cut off at the top with the *capFull* attribute.

The ***band***s are made up of individual ***dot***s, each with a date, a label, an source url, etc, just like ***dot***s are used in the ***path*** and ***cmap*** (concept map) ***resource*** displays, and fully clickable.

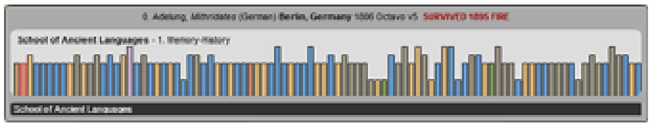
There are two additional types of timeview, controlled by the *style* attribute. The shelf *style* is used to place dots along a scrollable shelf, and the storyline *style* draws a series of lines that vary up and down.

Attribute	Description	Default
backImg	*Backgound image URL for full frame*	
border	*Border amount in pixels*	*8*
capCol	*Color of 3D cap as an RBG hex number*	*0x999999*
center	*Start dots in center of band: true | false*	*false*
id	*ID of resource**	
min	*Starting time of the timeview in any time format**	
max	*Ending time of the timeview in any time format**	
rot	*Angle of 3D rotation (in degrees, 0-45)*	*0*
style	*Style of display: shelf | storyline | timeview*	*timeview*
subtitle	*Sub-title*	
timeline	*Sync to timeline in view: true | false*	*false*
title	*Title*	

Elements: ***band*, frame*, textformat***

WIZARD SAMPLER

Shelf Style


School of Ancient Languages - 1. Memory-History
School of Ancient Languages

A ***shelf*** element is a display that shows a collection of images in scrollable frame. The ***shelf*** can have any number of ***band***s, each one having it's own time scale, allowing you to show images that occur in vastly different scales, in a similar fashion as the ***timeview*** resource element.

The ***frame*** defines the width, colors, and position of the overall display. The *hgt* attribute is ignored and automatically figured out by adding the heights of the bands in the ***shelf***.

The ***band***s are made up of individual ***dot***s, each with a date, a label, an source url, etc, just like ***dot***s are used in the ***path*** and ***cmap*** (concept map) ***resource*** displays, and fully clickable.

The ***shelf*** has a notion of a current dot, which indicates the one that has been clicked on last. The dot can be displayed with an edge around it by specifying an *onCol* (to red for example) that will rim the currently active dot. The *offCol* specifies the edge color of all the other inactive dots. You can make the edge's with bigger than the one pixel default be setting the ***dot***'s *frameWid* attribute to some other value than "1".

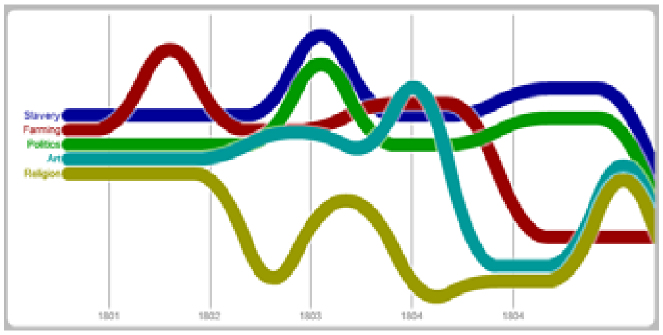
If ***shelf***'s *drag* attribute is "true" (the default), the ***band*** can be scrolled by dragging it directly. If the *drag* attribute is "false", the ***shelf*** cannot be dragged. If a ***dot*** is double-clicked, any GLUE set will be called and the clicked ***dot*** will be highlighted in whatever color you have the *onCol* attribute set to.

The ***shelf*** can be made to synchronize with a traditional ***timeline*** element by setting the *timeline* attribute "true". Any time the ***timeline*** is moving, the ***shelf*** will move and vice versa.

The currently active ***dot*** can be set by looking at the *curDot* attribute in a GLUE script. The refresh() method, can have an optional parameter cause a particular ***dot*** within the ***shelf*** to be highlighted and set as the *curDot*.

Storyline Style

Storylines are a cross between a line chart and timeview displays. The idea is that each line represents a dimension of something you wish to compare over time. This style was inspired by UDC Davis's Michael Ogawa who was in turn inspired by a comic strip on *xkcd.com* that plotted movie themes.



Each line must have a corresponding marker element that defines its color, line width, and whether the lines are smooth or straight. You can add ***dots*** as desired just as in a ***timeview*** display.

Data is added using the dataset() **GLUE** method, which contains a row for each interval being drawn. A 0 continues the line from the center, a 1 goes up one line width, and a -1 goes down, up to +/- 32. The interval is set by the *interval* attribute in the ***band*** element.

Band

If ***timeview***'s *drag* attribute is "true" (the default), the ***band***s can be scrolled by dragging it directly and if a ***dot*** is double-clicked, any GLUE set will be called. If the *drag* attribute is "false", the shelf cannot be dragged and a single-click will call any GLUE element set.

The ***timeview*** can be made to synchronize with a traditional ***timeline*** element by setting the *timeline* attribute "true". Any time the ***timeline*** is moving, the ***timeview*** will move and vice versa.

Each ***band*** can have a *ratio* set, defining how much of the display will be shown without scrolling: Setting ratio to "100" shows all the ***dot***s in a ***band*** at once; Setting ratio to "50" shows half the ***dot***s in a ***band*** and you need to scroll to see the rest, etc.

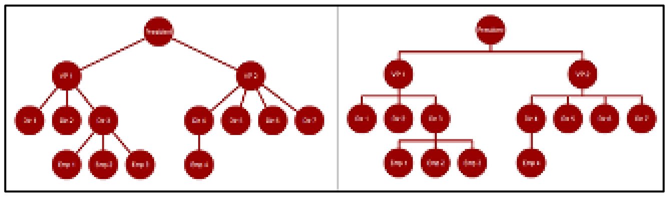
You can show dates within the ***band*** by setting the *tickDatePos* attribute to "top" or "bot". By default, they will be drawn every 365 days, but you can have them drawn any increment by setting the *tickSpan* attribute to the interval (i.e. "3650" for ten years).

Setting a ***band***'s *tickWid* attribute to something other than 0 will cause vertical tick lines to be drawn from top to bottom through the band at point when dates are shown.

Attribute	Description	Default
backImg	*Backgound image URL for band*	
border	*Border amount in pixels*	*8*
dataPos	*Position of data: bot | center | top*	*center*
col	*Color of background as an RBG hex number*	*0xffffff*
corner	*Radius of corner of frame for making rounded rectangles*	*0*
frameCol	*Color of frame edge as RBG (-1 = none)*	*-1*
hgt	*Height of frame in pixels**	
ratio	*Percentage of total time to show in band*	*100*
tickCol	*Color tick lines as an RBG hex number*	*0x999999*
tickDateFormat	*Date format for tick dates: yr|mo/yr|dy/mo/yr|mo/dy/yr|mo,dy,yr*	*yr*
tickDatePos	*Position of tick line date text: top|bot*	*bot*
tickSpan	*Number of days between tick mark lines*	*365*
tickWid	*Width of tick mark lines in pixels*	*0*

Elements: ***dot*, textformat***

NETWORK / ORGANIZATION MAP



Network and organization maps are similar to paths, but the dots are arranged according to the *to* and *from* attributes in the *line* elements. The *dot*s are not time dependent, and *line*s (edges) must be specifically drawn by setting the relationships between the *dot*s (nodes). Labels are automatically drawn if specified underneath the dot. The *frame* specifies the overall bounds of the map.

Setting *shape* to "org" will connect the *dots* in squared off lines as in an organization chart. Setting *shape* to "new" will connect the *dots* directly as in a network chart. The initial *dot*'s *from* attribute should be set to "" (i.e. nothing), to connect it to the screen.

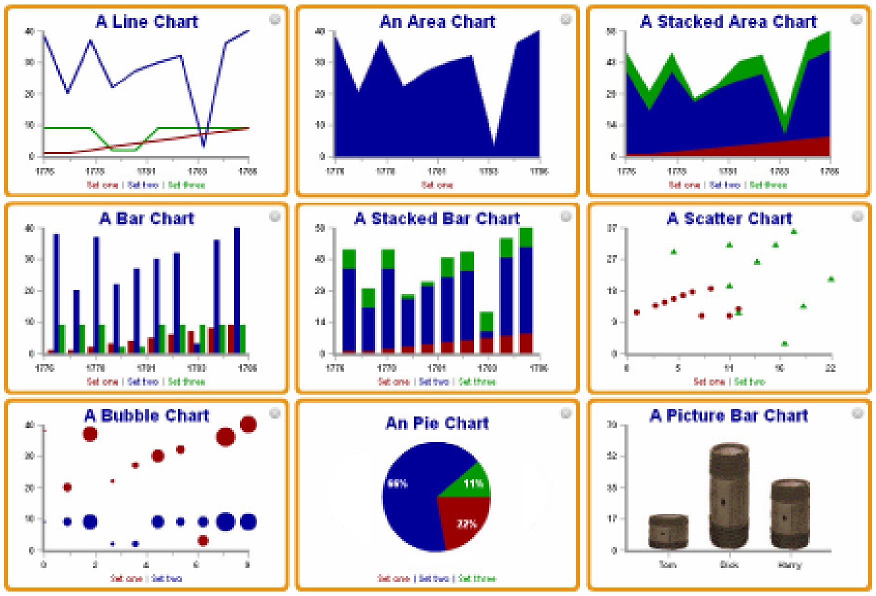
Setting the *shape* to "free" will place the ***dots*** according to the dot's *x* and *y* attributes, allowing for free form placement. Lines will connect between the *from* and *to* attributes set in the ***line*** elements

Attribute	Description	Default
alpha	*Opacity as a number from 0-100*	*100*
backCol	*Color of interior wash to blot out background as RBG (-1=off)*	*-1*
id	*ID of map**	
shape	*Shape of the lines connecting dots: free | org | net*	*net*

Elements: ***dot*, frame*, line*, linestyle, textformat***

WIZARD SAMPLER

Charts and Graphs



The VisualEyes ***graph*** element supports a number of chart types that can be drawn, including line, area, stacked area, bar, stacked bar, scatter, bubble, picture, and pie charts.

Charts

The line, area, and bar *style* charts can have multiple data sets, the color and labels of each as defined by the marker element. Charts can have x and/or y axes lines by adding an ***xaxis*** or ***yaxis*** elements to the resource definition, The *title* and *subtitle* attributes allow you to add titles and subtitles to the graph. The bar and area charts can have their data sets stacked by setting the *stacked* attribute to true. Setting the *legend* attribute to true will show whatever ***marker*** names were set for that dataset at the bottom.

Attribute	Description	Default
backImg	*Backgound image URL*	
border	*Border amount in pixels*	*24*
close	*Has close button: true | false*	*true*
depth	*Should it always appear on top: screen:topMost*	*topMost*
highWid	*Highlight width in pixels*	*0*
id	*ID of resource**	
legend	*Show legend: true | false*	*false*
radialImg	*URL for backgound image in radialbar chart*	
radialWid	*Diameter of backgound image in radialbar chart*	
subtitle	*Sub-title*	
showValues	*Show values on chart (pie only): none | percent | true*	*none*
stacked	*Are data sets stacked atop one another: true | false*	*false*
style	*Style: area | bar | bubble | line | picbar | scatter | area**	*area*
title	*Title displayed on chart*	

Elements: ***frame**, marker, textformat, xaxis, yaxis***

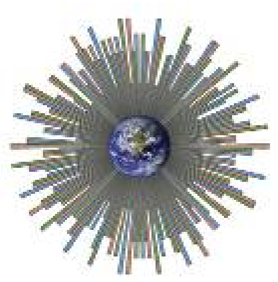
Scatter / Bubble charts

Scatter and bubble charts are bi-variate, requiring 2 datasets for each plotted set. On scatter charts, the first sets the position on the X-axis and the second one sets the position along the Y-axis. On bubble charts, the dots are plotted along the X-axis like a line chart, but the first data set controls the size of the dot drawn at each point. The bubbles are scaled according their value relative to the largest data value in that first set. The dataset's ***marker*** *wid* attribute sets the maximum size of the bubbles when the data value is the highest.

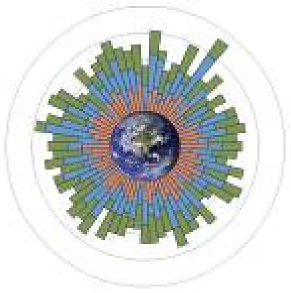
Pie chart

Pie charts get their label names and colors from the marker tags. There should be one marker for each pie slice. You can have the slice values printing inside each slice by setting the showValues attribute to “true”, or “percent” if you want the slice's percentage to the whole.

Radial bar chart



Radial bar charts are regular bar charts wrapped around an image in the center. The image is set using the *radialImg* attribute and should be square, as a circular portion will be automatically used from it. The *radialWid* attribute sets the diameter of the image. The *style* attribute should be set to "bar." This style looks best with a large number of data points.



The *stacked* attribute will stack the data sets as shown in the image on the right. Circular grid lines can be shown by adding an ***xAxis*** element and setting its *grid* attribute to "true." The ***frame*** element will set the overall size for the chart.

There is no wizard for radial bars directly, but use the regular chart wizard and work from there. There is an example of an animated radial bar chart in the SAMPLER.

Xaxis / yaxis

Attribute	Description	Default
autoScale	*Scale y axis maximum automatically: true | false*	*true*
col	*Color of line as RBG hex number*	*0x0000ff*
grid	*Show grid lines: true|false*	*false*
majorTick	*Length of major tick mark in pixels*	*0*
max	*Maximum data value*	*0*
midline	*Draw mid line horizontally : true | false*	*false*
min	*Minimum data value*	*0*
minorTick	*Length of minor tick mark in pixels*	*0*
mod	*Number to round values by*	*1*
pos	*Axis position: left|right*	*left*
showValues	*Show numeric values on axis: true|false*	*true*
title	*Title*	
valueCol	*Color of values as RBG hex number*	*0x0000ff*
valuePrefix	*Prefix for value labels*	
wid	*Length of axis line in pixels*	*0*

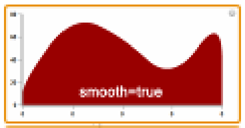
Elements: ***textformat***

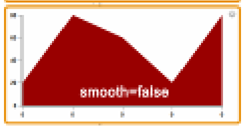
Legend

Attribute	Description	Default
lab	*Labels for legends*	
style	*ID of linestyle type*	

Marker

Markers define the color and other attributes of how the actual data is charted. Each dataset needs to have it's own ***marker*** element within the ***chart*** element. The *col* attribute set the color the data will be drawn. *datawid* sets the line width for line-style charts.





The *name* sets the default name of the dataset, but this is often overwritten when setting the data using a dataset() **GLUE** method. Setting the *style* sets the way individual points are rendered and icons and image files can also be used. The smooth attribute will smooth the lines in a line or area chart.

Attribute	Description	Default
col	*Color of marker an RBG number*	*0x000099*
datawid	*width of data (i.e. line or bar)*	*2*
edgeCol	*Color of marker edge an RBG hex number, or -1 for none*	*-1*
name	*Label of marker*	
smooth	*Are lines/areas curved?*	*false*
style	*Shape of marker: bar | cir | tri [u | d | l | r] | .jpg | .gif*	
wid	*Width in pixels*	*10*

Linestyle

Attribute	Description	Default
alpha	*Opacity as a number from 0-100*	*100*
col	*Color of line an RBG hex number*	*0x00ffff*
id	*ID of linestyle**	
lab	*Labels for line*	
letter	*Letter drawn midway through line in concept maps*	
wid	*Width of line in pixels*	*0*

Adding Data to Charts

Use the dataset() **GLUE** method to set the data sets with values.

glue

[script]

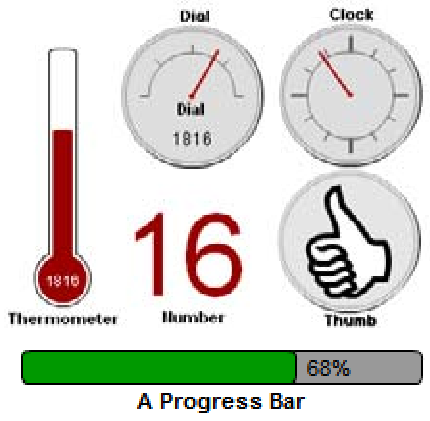
```
list($myData1,1,1,2,3,4,5,6,7,8,9)
list($myData2,38,20,37,22,27,30,32,3,36,40)
list($myData3,9,9,9,2,2,9,9,9,9,9)
dataset(myGraph,0,Set one,$myData1)
dataset(myGraph,1,Set two,$myData2)
dataset(myGraph,2,Set three,$myData3)
```

Animating charts

You can easily animate charts by using the tweenlist() GLUE method to transition between two lists of data.

You can also control what percent of the chart data will draw by seting the graph's *percentDone* attribute from 0-1.

WIDGETS



Widgets are a type of graph that graphically displays a single continuous value on the screen, such as a dial, clock, thermometer, etc. The range of widgets available will grow with time, but they all graphically display the *val* attribute from *min* to *max*.

The data is plotted in the color *col*. The *title* is displayed below the widget except for the dial, where it's within the dial. The numerical value is displayed to 2 decimal places if it is less than 1, or otherwise whole numbers. The size of round widgets like dials only look at the *wid* attribute, where things like thermometer and number use the *hgt* attribute as well. The thermometer widget looks when the *wid* is 1/8 the size of the *hgt* attribute.

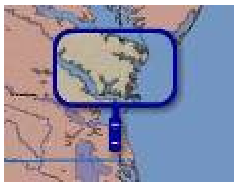
The spinner *style* will allow you to spin any icon, such as the thumb or spinner or arrow icons like the dial *style* widget. The *icol* attribute will color the icon to a different color. The crop and magnifier *style* widgets work in a completely different way than the other widgets, as they are used to work with images rather than values.

You can dynamically set the *val* attribute in GLUE scripts like this: **`setatt(myWidget,val,25)`** which would set the value of the widget with the *id* of "myWidget" to"25". See the GLUE reference for more information.

Attribute	Description	Default
alpha	*Opacity of band background as a number from 0-100*	*100*
back	*Show dial/clock/spinner background: true | false*	*true*
col	*Color of marker as an RBG hex number*	*0x990000*
glue	*GLUE id to be called if timer time is reached*	
hgt	*Height in pixels*	*0*
icol	*Color of spinner style icon*	*-1*
icon	*Shape of spinner-style icon: arrow1 | arrow2 | thumb*	*arrow2*
id	*ID of resource**	
left	*Number of pixels from left of screen*	*0*
max	*Maximum data value*	*100*
min	*Minimum data value*	*0*
style	*clock | crop | dial | number | progressthermometer | spinner | timer**	*dial*
title	*Title of widget to display*	
top	*Number of pixels from top of screen*	*0*
val	*Initial value to display*	*50*
wid	*Width in pixels*	*0*

WIZARD

Magnifier widget



The magnifier style widget will put magnified area of the screen atop a an image resource and let you drag it around as you would a real magnifying glass. Use the same image in the *src* attribute as the base image you want to magnify. See the "Aerials" tab in the Vinegar Hill project to see one in action.

A *frame* element set the size and color of the frame and the *top* and *left* attributes set its initial screen position. Clicking on the "+" and "-" on the handle scale the zoom up and down. Setting magnifier's ***frame*** element's the *corner* attribute the same size as the frame's *wid* attribute will draw a circular shape.

Attribute	Description	Default
id	*ID of resource**	
src	*Source URL of image to show within (full path with: http://)**	
style	*Must be magnifier**	
val	*Initial scale to display (0-10)*	*2*

Elements: ***frame****

WIZARDSAMPLER

Try it out: Add a Magnifier Widget

- Use the wizard feature to create a magnifier.
- Set the *src* as the image you are viewing. Getting fancier you can set the *src* as another map that is overlaid over the current image so that you could view through the magnifying glass how an area has changed through time. Or you could set the *src* as a topographical map that can be viewed through the magnifying glass over a road map. The two maps must be precisely geo-referenced.
- Play around with the attributes to see how they effect your magnifier widget.

Crop widget

The crop widget enables you to create a window on the screen that contains a subsection of an image. The *frame* element set the size and color of the frame and the *src* sets the image to show. Use the move() GLUE method to position the image within the frame (i.e. `move(myCrop,3300,656,400,1650,1680,400,$$now,3).`

Attribute	Description	Default
id	*ID of resource**	
src	*Source URL of image (full pathwith: http://)**	
style	*Must be crop**	

Elements: ***frame****

Timer widget

The timer widget is not a visual display, but is a countdown timer that triggers a **GLUE** element (specified by the *glue* attribute) each time a certain number of milliseconds (set by the *min* attribute) has been reached. For example, 1000 would cause the glue to be called every second The *max* attribute set when the timer will stop (i.e. 8000 would stop after 8 seconds). The timer widget is started and stopped by adding a refresh() **GLUE** element in a ***GLUE*** script: **`refresh(myTimer,start)`** or **`refresh(myTimer,stop).`**

In addition to triggering events on a regular basic, you can load a series of dots that define triggers at particular times to call glue elements. The ***dot*** elements must contain a *time* attribute that specified when, in ms., and a *glue* attribute that will be called. Dots must appear in ascending time. The **`$$param`** of the **GLUE** element will be set with the index of the ***dot*** in the list of dots. You can set the dots using the filltimer() GLUE method.

Attribute	Description	Default
id	*ID of resource**	
glue	*GLUE id to be called if timer time is reached*	
max	*Maximum time in ms*	*100*
min	*Minimum time per glue trigger*	*0*
style	*Must be timer**	

Elements: ***dot***

SAMPLER

Progress bar widget

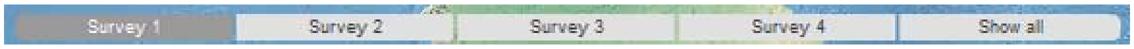
A Progress Bar

The progress style widget will put up a progress bar showing the percentage from 0-100% in a horizontal bar. A ***frame*** element set the size and color of the background frame and the *top* and *left* attributes set its initial screen position. Setting the frame's *backCol* and *frameCol* attributes to -1 will just show the progress bar without a background.

Attribute	Description	Default
col	*Color of bar as an RBG hex number*	*0x990000*
id	*ID of resource**	
style	*Must be progress**	
val	*Initial progress % to display*	*50*

Elements: ***frame****

Menu bar widget



The *menubar* style widget will put up a series of horizontal ***segment*** elements that when clicked on activate a GLUE element. Only one can be active at a time, like a radio-button control. The *onCol* and *onTextCol* determine the segments color and text color when each the segment is active, and the *offCol* and *offTextCol* attributes set the colors when inactive. You can make the segments different widths by setting *equal* to false, otherwise the segments will be same width.

Attribute	Description	Default
equal	*Make all segments equal widths: true | false*	*true*
hgt	*Height in pixels**	
id	*ID of resource**	
left	*Number of pixels from left of screen**	
offCol	*Color of inactive segment as RBG*	*0x999999*
offTextCol	*Color of inactive segment text as RBG*	*0x444444*
onCol	*Color of active segment as RBG*	*0x999999*
onTextCol	*Color of active segment text as RBG*	*0xffffff*
style	*Must be menubar**	
top	*Number of pixels from top of screen**	
val	*Initial segment to activate*	*0*
wid	*Width in pixels**	

Elements: ***segment****

WIZARD

Segment

Each button is defined by adding **segment** elements to the **widget** element. If the menubar's *equal* attribute is false, you can set the *start* attribute to set the width of each ***segment*** independently. The start attribute is the percentage of the total width defined in the menubar's *wid* attribute. Note that all the *start* attributes must add up to 100.

Attribute	Description
glue	*GLUE id to be called when segment is clicked*
start	*Percentage of total the segment takes of whole (0-100)*
title	*Text to be displayed in the segment*

VisualEyes User Guide
Appendix

Table of contents

Formatting infobox and docviewer text

The text displayed in the *infobox* and *docviewer* elements can be easily controlled using a subset of HTML tags shown below. You can use a combination of these tags, or more conveniently, use the macros listed below that simplify the formating.

Macro tags

Tag	Function
b(text)	bolds the text within the parentheses
i(text)	italicizes the text within the parentheses
u(text)	underlines the text within the parentheses
font(face,size,color)	sizes and colors the text that follows
sp(leading,indent,tapstops)	sets the leading, indent & tabstops for text. The tabstops are pixel location, separated by commas (i.e. *sp(0,0,100,200,300).*
t()	adds a tab
br()	adds a line break
*link(url,text,[target])	adds a link to call up a URL when the text in the parentheses is clicked. If the URL starts with http://, a new browser window will open and display the page, otherwise, VisualEyes assumes it is the name of a GLUE element and calls that GLUE element The URL is the link that will open when the text is clicked. A target can be optionally specified as third parameter which sets where the page will open, a frame's name or the preset values of _blank, _self, _parent, or _overlay (which opens an iFrame over the screen area). You can also specify an image file URL (jpg/png/gif) instead of text in the link() macro, so clicking on an image brings up a webpage.
align(side)	sets the alignment for the text that follows. Can be left, right or center
img(url)	will show the image at the specified URL

Special characters

Code	Character
&lt;	< (less than)
&gt;	> (greater than)
&amp;	& (ampersand)
&quot;	" (double quotes)
&apos;	' (apostrophe, single quote)

Raw HTML tags

Tag Name/Function	Tag Code
Anchor	<a href="url">
Bold	<b>
Font	< font [color="#xxxxxx"] [face="Type Face"] [size="Type Size"]>
Italic	<i>
Paragraph	<p [align="left"|"right"|"center"]>
Underline	<u>
Break	<br>
Image	<img src="/images/flash/dogs.jpg
List Item	<li>
Tab	<tab>
Textformat	<textformat > *blockindent* Specifies the block indentation in points *indent* Specifies the indentation from the left to first leading Specifies the amount of leading (vertical space leftmargin Specifies the left margin of the paragraph rightmargin Specifies the rightmargin of the paragraph tabstops Specifies custom tab stops as an array of non- negative integers.

Formatting docviewers

The ***docviewer*** display element is useful for showing collections of text and images in a booklet-like format. It can contain any number of ***page*** elements each one with the following information:

• **title**	The title that appears in bold across the top.
• **caption**	Text that appears underneath the image
• **src**	The URL (including the http:// part) of the image, if any
• **desc**	The formatted text to display, if any

The *desc* text can be formatted by using any of the tags in the previous page

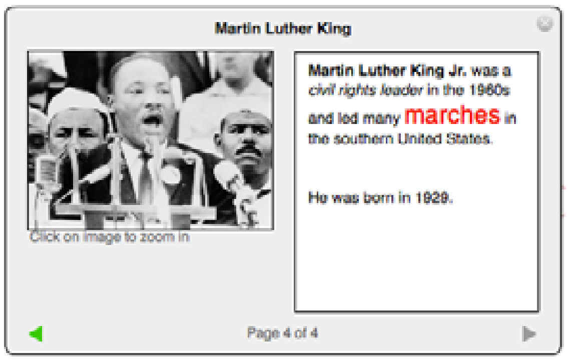
For example, the following page:

• **title**	`Martin Luther King`
• **caption**	`Click on image to zoom in`
• **src**	`http://www.primaryaccess.org/Test.jpg`
• **desc**	`b(Martin Luther King Jr.) was a i(civil rights leader) in the 1960s and led many fon(18,#ff0000,marches) in the southern United States. br()br()He was born in 1929.`

And this page:

• **title**	`Martin Luther King (page 2)`
• **caption**	`Click on image to zoom in`
• **src**	`http://www.primaryaccess.org/Test9.jpg`
• **desc**	`second page`

Would yield a docviewer that looked like this:



Martin Luther King

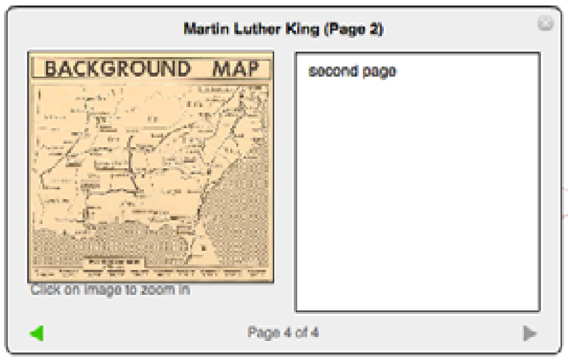
Martin Luther King Jr. was a *civil rights leader* in the 1960s and led many marches in the southern United States.

He was born in 1929.

Click on image to zoom in

Page 4 of 4

Page 1



Martin Luther King (Page 2)

second page

Click on image to zoom in

Page 4 of 4

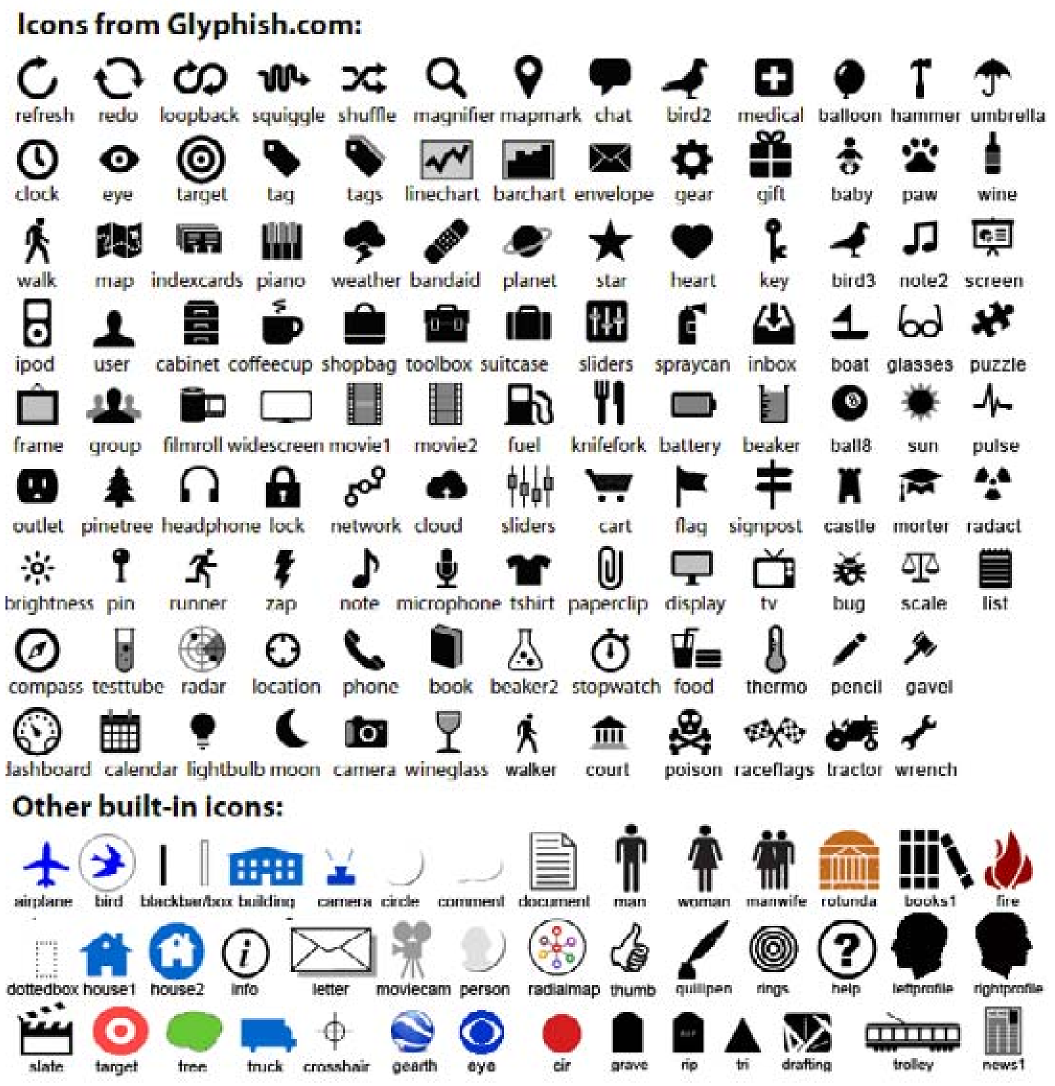
Page 2

Icon types

There are over 100 included in VisualEyes that can be used on ***dots*** and chart ***markers***. They are vector-based, so they do not get pixilated when scaled up. To use one, look up its name in the drawing below and add *icon:* as a prefix to the name (i.e. *icon:comment*) in the ***dot*** *style* attribute.

Icons can be scaled up and down by setting the *wid* and *hgt* attributes to the desired width and height, and rotated any angle using the *rot* attribute. If you specify only the *wid*, or only the *hgt*, the other factor (*wid* or *hgt*) will be automatically scaled to match its original shape. You can distort that shape by setting the *wid* and *hgt* together. By default, icons are colored the way they are in the drawings below, but you can color icons using the *icol* attribute in a ***dot***.

We have been given permission to use 100+ great icons designed by Joseph Wain from Glyphish.com. They are drawn in black, and any gray portion has its transparency set, so when they are colored using the *icol* attribute, the grey areas come out in shades of the *icol*.

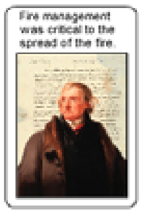

Icons from Glyphish.com:
refresh redo loopback squiggle shuffle magnifier mapmark chat bird2 medical balloon hammer umbrella
clock eye target tag tags linechart barchart envelope gear gift baby paw wine
walk map indexcards piano weather bandaid planet star heart key bird3 note2 screen
ipod user cabinet coffeecup shopbag toolbox suitcase sliders spraycan inbox boat glasses puzzle
frame group filmroll widescreen movie1 movie2 fuel knifefork battery beaker ball8 sun pulse
outlet pinetree headphone lock network cloud sliders cart flag signpost castle morter radact
brightness pin runner zap note microphone tshirt paperclip display tv bug scale list
compass testtube radar location phone book beaker2 stopwatch food thermo pencil gavel
dashboard calendar lightbulb moon camera wineglass walker court poison raceflags tractor wrench
Other built-in icons:
airplane bird blackbar/box building camera circle comment document man woman manwife rotunda books1 fire
dottedbox house1 house2 info letter moviecam person radialmap thumb quillpen rings help leftprofile rightprofile
slate target tree truck crosshair gearth eye cir grave rip tri drafting trolley news1

Dot Drawn Style Types

There are a number of drawn shapes that can be used on dots and markers. They can be scaled up and down by setting the *wid* attribute to the desired width. Setting the *col* attribute will set the color they will be drawn in:

bar	*A filled bar*
but	*A filled bar with circular ends*
cir	*A filled circle*
rbar	*A filled bar with rounded corners*
span	*A two-headed arrow with text beneath*
star	*A filled 5-point star*
trid	*A filled triangle facing down*
tril	*A filled triangle facing left*
trir	*A filled triangle facing right*
triu	*A filled triangle facing up*

Sign style



Signs are a combination between an image ***dot*** and an *rbar* style ***dot.*** The *col*, *frameWid*, and *frameCol* attributes set the background box. The *corner* attribute sets the size of the border (default is 8), and the text wraps to the top, with the alignment controlled by the *labelPos* attribute.The image is set by appending the *style* attribute with the url of the image, like this: s**`ign:http://www.mysite.com/image.jpg.`**

Working with structured information

One of the things that makes VisualEyes particularly useful is its ability to manage, ask questions of, and display elements from large collections of structured information.

What is structured information?

Structured means that instead of the data and information presented like a text document, particular kinds of data are grouped together in meaningful groups. In contrast, unstructured information is like a Word document file. All the information is put together with nothing separating the important elements like this box.

This may work for small amounts of information, but imagine needing to find all the men that passed the test if there were 400 people in the class. It would be very difficult to automate, as the computer would have a hard time telling the ages from the grades.

One solution is to structure the data. That is, if we know an item is a grade, put it in the "grade" group, while an item that is a name would go into the “name” group. We call these collections of structured information tables, and they are really no different than an ordinary Excel spreadsheet.

name	**sex**	**age**	**grade**	**class**
Bob	male	22	100	1
Ted	male	43	40	2
Carol	female	33	90	1
Alice	female	23	75	2

Both a spreadsheet and a table consist of multiple rows of information, sorted into useful groupings in columns. An example of this shown in the box to the left.

Each column is a grouping of related things, called a field. The first row defines the names of the fields, followed by any number of rows that contain the information for the fields. In this table, there are 4 people (Bob, Ted, Carol, and Alice) and 5 fields (name, sex, age, and grade, and class).

Having the data structured makes it easier to make sense out of the table, and ask it better questions. Google is essentially an unstructured table of the web. When we search, it looks to see if any of our search words appear in a web page, and return those pages if it does. An example of an unstructured search would be if you searched for “Tiger” Woods, the golfer. Your search results would include data about the golfer as well as about the feline predator. By comparison, a structured search would involve searching specified fields in a structured table. For example, to conduct a structured search for the occurrences of “Tiger” as a name, you would indicate that you are searching for matches in a structured table’s name field." Structure adds a new level semantic meaning to our searches.

On the web, tables are stored in programs called databases. VisualEyes has a simple database built-in to support you in structuring your data into tables, and then to easily and quickly search for the portions of information you want from your tables.

Putting your table online

The VisEdit webapp has a feature that will take a spreadsheet file, convert it to XML and store it on the VisualEyes server for you. As a result, you can use Excel (or just about any spreadsheet application) to import data to the VisualEyes server so that you can access it online. What follows are the steps to convert your data to XML:

1. Open and format your spreadsheet. To begin converting your data, the first row of your spreadsheet should contain a list of single-word field names by which each column can be referred (i.e. name, sex, age, grade, id in the previous example). The rows that follow within each column can contain any number of items of data sorted across the horizontal fields.

2. Save your spreadsheet using the CSV (comma delimited values) or tab-delimited text file formats available from most spreadsheets and database programs. To do this, use the "Save As…" option in the File menu and set the "Save as type" option to CSV (Comma delimited) or "Text- (Tab delimited)" and save to a file.
3. Go to the VisEdit Editor. In the Tools menu of the VisEdit editor, select the Convert Data File to XML option.

4. A file dialog will prompt you to locate on your hard drive, the spreadsheet you want to convert.

5. Once selected, your spreadsheet will appear in the screen.

6. Make whatever changes you need to the raw text in this view, such as editing the field names on the first line so that they do not contain any spaces. We like to use camelCase (first letter of words in caps, except the first. i.e. myFirstName, myAge, etc.) as it makes for a pronounceable field names.

You can load CSV files directly to the server using the "Upload CSV file to server" option in the Tools menu and it will be saved to server's data folder with your user number its name (i.e. *http://www.viseyes.org/data/1-BobTed.csv*). You can upload revised versions over this at any time if your data changes.

NOTE: This will add 10-20 seconds to the load time when the project is run, so when you have finished making changes to the file for a while, save it as a native XML file for faster loading:

Click on the Convert to XML button to convert your spreadsheet data to XML format. You will be asked to type a name for the table to store it under. Click on the Upload to server button to save that file on VisualEyes server's data folder with your user number and the name you gave it (i.e. *http://www.viseyes.org/data/1-BobTed.csv*).

Querying a table

The process of "asking" a table for certain data is called querying. You do queries all the time on the web when you conduct a search. For example, when you try to find a movie in Netflix, you ask the Netflix server to search its table of movies by matching the words you typed in. Behind the scenes, your search words are sent to the server at Netflix, which "asks" the database to look through the genre you are in and return the titles of any films in which all your search words can be found. After a few seconds, Netflix displays a list of search results. The same process occurs when you search for books at the library website, Google, and even Apple's iTunes, which is no more than a simple database.

The parts of a query

To conduct a query in VisualEyes, you need three basic pieces of information to get the data you want from a given table:

1. The name of the table
2. The conditions
3. The desired fields from the source if the conditions are met

The name of the table

The name of the table that contains the raw information you want to pick and choose from. Since any given project might have many tables, to choose from, you need to specify one of them by giving its name.

The conditions

The conditions that need to be met before any rows are retrieved from the source table. Conditions are statements like, "all the people who scored below 70" but in a form that the computer can understand, such as "grade LT 70". We take advantage of the structured nature of our data and look at the "grade" field to return only people who have grades less than (LT) 70.

A single condition like "grade LT 70" is called a clause. Each clause is said to be true if the condition is met (i.e. the grade is 50) or false if the condition is not met i.e. the grade is 80). Each clause had three parts:

1. The field to look at
2. The conditional (i.e. GT, LT, EQ ...), and
3. The value to compare with, which can be a number, word, sentence, or another field name.

The conditions can get more specific by adding multiple clauses like any Boolean search. In our example, "men who scored over 60 and are under 40" is a condition that translates into three clauses joined by "sex EQ male" AND "grade GT 60" AND "age LT 40." The AND that separates each clause is called an operator and says "return rows if both the clauses it is between are true." Alternatively, we could use the OR operator which says "return rows if either of the clauses it is between are true."

Which fields to return

Your table might have 5 fields, but you may only need to get one, such as the “name”. To do this, you need to specify which fields to include in the results. Specifying "name" will return just the name (i.e. Bob), and "name+age" will return the name and age (i.e. Bob, 22). If you want all the fields, use a star ("*") (i.e. Bob,male,22,100,0).

Glossary

Clause	*A condition that must be met if an item is to be included*
Conditional	*A comparison such as less than, greater than, equals, etc.*
False	*The result of a clause that makes it omitted from in the results*
Field	*A category that an item is separated by type*
Item	*A line of information separated by fields*
Operator	*Used to join clauses together*
Query	*A request for a subset of table items meeting certain conditions*
Results	*A list of items that met the conditions posed*
Structured	*Information is sorted into fields*
Table	*A structured collection of items organized into fields*
True	*The result of a clause that makes it included in the results*

XML Data Format

The easiest way to import data into VisualEyes is using Excel to create a spreadsheet. The top line should contain the field names and the following lines that data for those fields. For example, this format defines 3 fields, name, sex, age and has 4 people's information:

name	**sex**	**age**
bob	male	22
ted	male	43
carol	female	33
alice	female	23

Save this out as a tab-delimited text file in Excel by selecting "Save As…" in the File menu and setting the "Save as type" option to "Text- (Tab delimitated)" and saving to a file. The VisEdit editor has a tool that will allow you to load that file from your computer to the screen area, where it will be formatted into an XML format like this:

```
<TABLE a="name" b="sex" c="age">
<ROW a="bob" b="male" c="22" />
<ROW a="ted" b="male" c="43" />
<ROW a="carol" b="female" c="33" />
<ROW a="alice" b="female" c="23" />
</TABLE>
```

Click on the "Upload to server" button and that file will be saved on VisualEyes server's data folder using your user id and a name you gave it (i.e. /data/1234-MyXMLFile.xml).

Web Table Data Import

Existing websites are a great good of data for projects, for example the Historical Census Browser (http://fisher.lib.virginia.edu/collections/stats/histcensus) is a great way to get county-level census data from 1790 to 1960. Once you have found a table of data you want, select the entire table and copy (CTRL-C) it into your computer's clipboard.

Paste (CTRL-V) this data in an open Excel spreadsheet. The first line should contain a list of single-word field names that each column can be referred by (i.e. name, sex, age in the precious example).

Save this out as a CSV (comma delimited values) or tab-delimited text file in Excel by selecting "Save As…" in the File menu and setting the "Save as type" option to CVS (Comma delimited) or "Text- (Tab delimited)" and saving to a file.

In the Tools section of the VisEdit editor, select the Convert Data File to XML option. Make whatever change you need to the raw text. Click the Convert to XML button. Edit the field names on the first line so that they do not contain any spaces. Click on the "Upload to server" button and that file will be saved on VisualEyes server's data folder using your user id and a name you gave it (i.e. http://www.viseyes.org/data/1234-MyXMLFile.xml).

Sharing Your Project with Others

The adding the project number to the following URL will allow anyone with an Internet connected web browser that has the Flash plug-in installed to see your project:

www.viseyes.org/show?id=xxxx

Replace xxxx with your actual project number, which is show at the top right corner of the VisEdit screen.

Embedding VisualEyes Projects in Web-pages

The Easy Way

While you can always show your VisualEyes project using our website using the www.viseyes.org/show?id=xxxx link, you can also embed it more seamlessly into your own website by adding 2 lines to the page you want to embed the project on your site:

```
<script> var id="xxxx";   var bcol="#ffffff";   var wmode="opaque"; </script>
<script src="http://www.viseyes.org/embed.js"></script>
```

Replace the xxxx with your actual project number, which is show at the top right corner of the VisEdit screen. If you want a different background color than white, replace the ffffff with the color you want behind the tabs and under the timeline. You can make that area transparent by changing opaque to transparent.

The Hard Way

If you want to put a copy of the actual SWF file on your page, the process is more complicated. The reason to do this is to freeze the version of VisualEyes and ensure the version you are embedding will not change. This is important for museums and other institutions that require the utmost of stability.

To do this, you will need to add two files from us, the SWF file called VisualEyes.swf and a JavaScript file called “localembed.js.” Email me at bferster@virginia.edu for them.

Put both files in the same folder as your webpage you want to embed the project in and add the following lines to that file:

```
<script> var id="xxxx";   var bcol="#ffffff";   var wmode="opaque"; </script>
<script src="localembed.js"></script>
```

Replace the xxxx with your actual project number, which is show at the top right corner of the VisEdit screen. If you want a different background color than white, replace the ffffff with the color you want behind the tabs and under the timeline. You can make that area transparent by changing opaque to transparent.

Finding URLS of Flickr Images

An image used in VisualEyes needs to be online to be included in a project. Yahoo's Flickr photo sharing site (www.flickr.com) is free and easy way to put images online. When you upload a picture to Flickr, it loads your original image and stores it, as well as making 5 jpeg copies of varying widths:

- Square (75 pixels)
- Thumbnail (75-100 pixels)
- Small (100-200 pixels)
- Medium (400-640 pixels)
- Large (1024 pixels)

If the image is in landscape mode (i.e. its width is greater than its height), the width will be set to the pixel number listed above and the heights scaled are proportional to the width. If the image is in portrait mode, the height will be set to the pixel number listed above and the width scaled are proportional to the height.

It is best to use an image closest size to how it will be displayed. Too small, and it will appear pixilated, too big, and it will take too long to load and not look as good as one better fitted.

Each size has its own URL. To find it, click on the "All Sizes" item in the picture's "Action" pull-down at the top. Click on the size you want from the "Available sizes" menu, and that size will appear. Right-clicking (Control-click on Mac) on the "Download" will bring up options to copy the link location (called a "shortcut" in IE) to the clipboard, where you can use it as the image's src attribute. You can delete the _d just before the .jpg extension, as it is not needed.

NOTE: Unless you have a paid Flickr "Pro" account, you will not be able to access the original size image. The shanticohort account is Pro but they are inexpensive ($25/yr) if you need your own.

Using Google Earth

You can dynamically show .KML and KMZ Google Earth files in a browser page by using the link() glue method and calling the kml.php file as the url. To call a Google Earth file from a glue script, add a link() method to the script. The link() method has two primary parts, the url and the target.

The url consists of 2 required and one option parts to it: 1) the KML web page: *http://www.viseyes.org/kml.php* 2) the url of the KMZ or KML file: ?url=http:// and optionally 3) display option letter: &show=xxx.

There are a number of options we can set by adding letters in place of the x's :

Option code	Function
r	Shows the roads
b	Shows country/state borders
t	Shows terrain
3	Shows 3D buildings (if any)
s	Shows status bar on bottom with lat/lon
l	Shows scale legend

Target sets where the page will open, which can be set to the frame's name or the preset values of _blank, _self, _parent, or _overlay (which opens an iFrame over the screen area.

For example, if we wanted to show the following KMZ file at

http://www.viseyes.org/map.kmz in an overlay window,the url would be:

link(http://www.viseyes.org/kml.php?url=http://www.viseyes.org/map.kmz,_overlay)

If we wanted to show roads and the scale in a new brower page:

link(http://www.viseyes.org/kml.php?url=http://www.viseyes.org/map.kmz&show=rl,_blank)

NOTE: you will need to have the Google Earth Plug-in installed on your web browser, from http://earth.google.com/plugin

Geo-Referencing Maps and Images

You can specify dot coordinates in longitude and latitude, instead of specifying the coordinates in x and y pixel values. This makes it easier to use locations from Google Maps and other GIS-aware applications. To do this, we need *geo-reference* (a GIS term) the map that the dots will be placed over to correlate the longitude and latitude values to their position on the map or image.

1. Choose two points on the map, one in the upper left corner, and one in the bottom right.

2. Get the pixel position for each point by clicking on the point and pressing the "Alt" key when viewing the project. The screen position will appear in the bottom right of the screen.

3. Find the latitude and longitude and for each point. The longitude (a negative number for US locations)

4. Add 4 new attributes to the image or map element, matching a pixel value to a geo-coordinate, separated by a colon. The gl is the left side, gr the right side, gt the top, and gb the bottom (i.e. gl="49:-78.500488" gt="41:38.041771" gr="759:-78.46872" gb="460:38.027124")

5. You can now specify the dots x and y attribute in longitude and latitude coordinates (i.e. x="-78.500488" y="38.027124). In the path element containing dots add an attribute called res to tell the path to rectify the dot to the particular resource you added the gl/gt/gb/gr attributes to (i.e. res="myMap).

For an example, see the "Geo-reference a map / Google Earth" section in the project SAMPLER

Geo-referencing dot sizes

Dots can be placed according to latitude and longitude, instead of pixels by attaching the <path> to a geo-referenced map using the res attribute. (See section on geo-referencing for more information.) Typically, the width and/or height of the dot's graphics is set in pixels, such as a star, icon, or circle, but sometimes, you want to map an area based on the extent in latitude and longitude. To do this, specify the wid and/or hgt attributes in negative decimal degrees.

This is most often done when geo-referencing an historic map using Google Earth. Once referenced, we need the extent of the map. To find it, right-click on the layer in Google Earth and select the "Properties" item. When you click on the "Location" tab, the extent will show as North, South, East, and West boxes.

The wid is found by subtracting the East side of the area from the West side. The hgt found by subtracting the North side the area from the South side. The fact that these will both be negative number will alert VisualEyes to convert them to pixels from degrees: wid=west-east and hgt=south-north.

The x and y <dot> attributes point to the middle of the area. To find the middle (called a centroid) from the extent, the x is the West + (East-West) /2 and y is South + (North-South)/2.

Click on the "Snapshot View" to in the right click menu to save the current view as the default vantage point when your KML file is launched by Google Earth or VisualEyes. Also, make sure that the layer or folder you snapshot is not in the "Temporary Places" folder, but directly under the "My Places" folder.

Using Google Docs to Store Data

VisualEyes supports a number of formats for storing table data, XML, CSV, ManyEyes, and Google Docs Spreadsheets. Of all the choices, Google Docs is the fastest to use when developing your project.

Google Docs has an online Excel-like spreadsheet web-app that allows you (and collaborators you invite) to edit a spreadsheet hosted by Google. You can use it as you would a CSV file by publishing it and using the link in your *src* attribute for the XML resource element in your project.

There are many advantages to this strategy: The link will automatically be up dated as you change the data, with needing the tedious uploading of the new data to the server; Google Docs maintains a revision history of previous versions and Google keeps your data safe. I would recommend saving a copy on your computer's hard drive just as a precaution. When the data stops being changed, it is probably best to save a CSV file from Google Docs, upload it to the VisualEyes server, because it does take 5-20 seconds longer to load from Docs than a CSV file.

To use Google Docs Spreadsheets in your project

1. Sign in to your Google mail or web account.
2. Select "Documents" from the "more" link at the top of the Google search page.
3. Create or import your data from an existing spreadsheet.
4. Share the spreadsheet by clicking on the "Share" button at the top right corner.
5. Change the permission visibility so "Anyone who has the link can view."
6. Click on the triangle to the right of the "Share" button.
7. Select "Publish as a web page" option from the "Share" pull down menu.
8. Check "Automatically republish when changes are made."
9. Click the publish button.
10. The link will appear in the section called "Get a link to the published data."
11. Copy the link and use it as the *src* attribute in the XML resource element.

Table resource

The table resource element creates an empty table with whatever fields you want, rather than importing CSV or XML files. This is useful when you want to make a table from a subset of some elements in an imported file.

The src attribute provides the list of fields for the new table. The fields are separated by a "|" (pipe), so for example a src attribute of "name|age|sex" would create a table resource with three fields called name, age, and sex.

The table resource is typically filled with data by querying another table with a GLUE script. Suppose we had a table we imported with hundreds of name, but only wanted those who attended "Wilson" in our new table:

```
query(*myTable.name,schoolList,student,school EQ Wilson,0)
query(*myTable.sex,schoolList,age,school EQ Wilson,0)
query(*myTable.age,schoolList,sex,school EQ Wilson,0)
```

The first line fills name field in the new table we created with the results of a query on the schoolList table, extracting a list of students whose school field was Wilson. Note that the name of the field in our new table (name) does not need to match the table in the query (student). The second line adds the sex and the third line adds the ages.

You can also set individual table resource members using the table() GLUE method, which has options to add a new row, or set a row's field value.

Accessing individual data elements in a table

Tables are typically accessed by querying the data with a query() method, but you can access individual elements by specifying them by field. For example, if we had a resource with the id of "myTable" and a field called "name", status(*myTable.name) would print a list of all the rows of the name field on the screen and status(*myTable.name.1) would print the 2nd name (the count starts at zero).

Movies

Setting the glue to some glue object will cause that glue object to be called every n ms specified by timer. This is useful for controlling other screen elements to do something at some time in the movie's playback.You can also control movies from a GLUE script. See the movie() method for more information.

MORE TO COME...

VisEdit Options

Moving Elements in the Main Tree View

The UpArrow and DnArrow keys will allow you to navigate through the Main Tree view of your project. The Home key will bring you to the top. To make editing the XML directly less needed, using the Shift-UpArrow or Shift-DnArrow keys will move the currently selected element up or down in the Tree. Any sub-elements within that element will also be moved. There are options in the Edit menu for doing this as well.

Copy and Paste of Elements in Main Tree View

To make editing the XML directly less needed, selecting the Copy option from the Edit menu (or hitting Ctrl-C) will copy currently the selected element to the clipboard. Any sub-elements within that element will also be copied. Selecting the Paste option from the Edit menu (or hitting Ctrl-V) will copy any elements in the clipboard just beneath currently the selected element in the Tree.

Undo/Redo in Main Tree View

You can click on the Undo option in the Edit menu to go back to a previous step. This works similarly as in programs like Microsoft Word. You can go back to your last 100 actions. Selecting the Redo element in the Edit menu will "undo" the undo. You can also undo the last 32 keyboard actions by hitting the Ctrl-Z key. This key is also available when editing GLUE and <infoBox> script areas.

KML preview

If you click on a or KML file in the Show/My Data files menu option, a new browser window will appear, and that KML file will show in that window using the Google Earth plug-in.

The text in that KML file will be shown in the help area as well. At the end of the KML file, a dot element containing the centroid information of any map overlay in the KML file used for geo-referencing the dot will be displayed (see: Geo-referencing dot sizes).

Data Import from Many-Eyes

IBM's Visual Communication Lab has a great free website (www.many-eyes.com) for visualizing, storing and sharing data sets. You can automatically pull them in as an xml data source by using a link to its data file. You can import these data sets dynamically into History browser by locating a data set or uploading your own to their site. Click on the link called Data File and use that URL when defining an xml resource. This is a simple example of a data set on ManyEyes:

resource
type="xml"
id=myData"
src="http://manyeyes.alphaworks.ibm.com/manyeyes/datasets/things-2/versions/1

Alternatively, you can copy the data on our server by doing the following: Add ".txt" to the url in a web browser, (ie http://manyeyes.alphaworks.ibm.com/manyeyes/datasets/things-2/versions/1.txt) and highlight the data manually (CTRL-A) and copy the data (CTRL-C) onto your computer's clipboard.

In the Tools section of VisEdit, select the Convert Data File to XML option. Instead of loading a file to convert, click "Cancel" and paste (CTRL-V) the data over the instructions in the text box. Click the Convert to XML button. Edit the field names on the first line so that they do not contain any spaces. Click on the "Upload to server" button and that file will be saved on VisualEyes server's data folder using your user id and a name you gave it (i.e. http://www.viseyes.org/data/1234-MyXMLFile.xml).

Upload XML Project File Directly

While it is possible to edit the XML directly using the VisEdit editor, many people making projects will feel more comfortable editing the XML in a text editor such Oxygen or DreamWeaver. These editors have good undo/redo and context coloring the make the process much easier.

To support this work flow, there is an option in the VisEdit File menu called "Upload Local XML File" which will bring up a file box and allow you to select an XML file from your computer's hard drive and upload it to your currently active project. Once uploaded, it will open the same browser window that the "Save and Preview" button uses to preview the project.

The flow goes like this:

1. Edit XML in DreamWeaver
2. Save file to disk in DW
3. Upload Local XML File in VisEdit editor
4. See how it looks
5. Go back to step 1.

Converting AI Files to XML

VisualEyes needs vector graphics images, such as those that come from Adobe Illustrator and arcGIS to be converted to XML format in order to be used as a ***map*** resource. We only support straight lines and polygons and only version 8 files at present.

To convert a file:

1. Save the file in Adobe Illustrator format (version 8) from either arcGIS or any program that output's .ai files, such as Fireworks and Illustrator.

2. Log into VisEdit and select the "Convert data/AI file to XML" option from the Tools menu. Find the .ai file on your computer using the file finder that pops up.

3. In a few seconds, the .ai file's data will appear in the text box.

4. Click on the "Convert to XML" button and the .ai data will be converted to XML displayed. You can modify any attributes, such as col, edgeCol, alpha, or edgeWid attributes.

5. The "Preview" button will show and hide a preview of your graphic on the screen area.

6. Click on the "Upload to Server" button and after giving a file name to store it by, the file will be saved on the VisualEyes server and the name of the file (i.e.http://www.viseyes.org/data/92-test.xml) will be shown on the screen for you to copy and paste into the map resource's src attribute.

7. Select the "Go to View" option in the View pull-down menu and select "Main”

8. Tree View" to exit the Tool menu.

Conversion options

By default, your map will be rescaled to 800 pixels across. To change that, type "width=####" in the textbox next to the "Preview" button, where "####" is the new width (i.e. width=1200 would scale the graphics so they are 1200 pixels across.

Similarly, you can offset the graphic horizontally by adding "left=####" , and vertically by adding the words "top=####" , each offset or width separated by a comma (i.e. (i.e. width=900,left=10,top=100 would scale the graphics so they are 900 pixels across, starting 10 pixels from the left and 100 from the top.

Map XML options

There are some options you can edit in the shapefile XML before saving it to the server. The *offx="xxx"* and *offy="xxx"* attributes will position the map around the screen. The *hair="true"* option will render any lines or edges of polygons as hairlines, meaning they will not get larger when you zoom in. The *smooth="true"* option will render any lines or edges of polygons as curves rather than straight lines.

Feature IDs

By default each new polygon added gets a new unique id number identifying it as a feature. This is the number that will be returned to identify the feature when clicked. These id's can be edited if desired and there are some options for changing the id's using GLUE featureID() method.

Interactive Maps

Each polygon in a can be clicked on and the feature number returned in a list member named $$click in GLUE scripts. By default, each feature is give a number corresponding to its order in the list, but you can set the ids to particular numbers if desired. You can set the individual feature's edge and interior fill color by setting the cols and edges lists (i.e. set(*myMap.cols.4,0x990000) sets the 5th feature to red).

CS2-CS5 AI Files

Loading .ai files created in Adobe's Creative Suite is complicated by the way they encode them internally. We can read them, but the process is convoluted. Open your CS .ai file in a text editor, and copy the contents of the file to your clipboard. Then load a "dummy" version 8 file in VisEdit. Replace the text that was just loaded with the contents of the CS file you copied. Converting and saving will now work.

NOTE: A ***map*** resource needs to be refreshed in order to be drawn. To do this, add a line to a **GLUE** script that says: **refresh(myMap),** where "*myMap*" is the *id* of your ***map*** resource.

Using invisible views

In general, project views appear underneath the tabs when created, but views can also be created invisibly in the background, and explicitly called to the screen when desired. You may have too many views and they won't fit on the screen, or want to dynamically pull up a view based on some actions. Setting the visible view attribute to "false" will allow you to add and edit a new view in the project, but it will not appear unless it is called using the setview() GLUE method. You must also add an id attribute to refer to it by.

Whenever you want that view to appear, add a GLUE element in the view you want to replace with the a script containing the following line: setview(myView), where myView is the id of the hidden view you want to see. Whenever that GLUE element is called, it the view in which it resides will be replaced the one called id. The old view can be replaced by a setview() call to its id attribute, or simply clicking on any tab on the screen.

The persistence of dot attributes

Since the dot elements are very frequently used and many dots share attributes in common, we have tried to minimize the need to redundantly specify most attributes. Most dot attributes, with the exception of x, y, and icol store their last specified values, so if those values are not explicitly changed, they default to the last time they were set.

In the following example, a path has four dots associated in it:

Dot 1	*x=100*	*y=200*	*style=cir*	*wid=20*	*col=0x990000*
Dot 2	*x=110*	*y=210*			
Dot 3	*x=120*	*y=220*			*col=0x000099*
Dot 4	*x=130*	*y=230*	*style=star*		

Each dot specifies the x and y attributes, because each dot has a unique position on the screen, but the other attributes come and go as needed for each of the other four dots:

- Dot 1: Sets the style to a circle, the wid to "20" pixels and the col to "0x990000"
- Dot 2: Draws the same style, wid and col, since they were not changed.
- Dot 3: Uses the same style and wid, but the color is now set to "0x000099".
- Dot 4: Uses the same style and blues col, but the style is now a "star".

This same idea works when loading dots via an XML or CSV file. Assuming you had a list of x, y, or other attributes in a file, any constant attributes can be inherited from a dot element in the path like this:

resource
id=myData *type=xml* *src=mydata.xml*
path
id=myPath *style=cir* *wid=20*
col=0x990000 *glue id=fillDots*
glue *[script]* *dotfill(myPath,myData)*

Using Resource Pointers

Resource pointers allow you to use a data resource, such as a CSV or XML file from another view without needing to reload the file again in the current view. This saves the time needed to reload large data files that have already been loaded in another view.

The resource must be in a previous view to the one you want to add access to, and have an *id* attribute set, so you can identify it using the *src* attribute.

Attribute	Description	Default
id	*ID of resource to point to in other view**	
src	*ID of view where resource was first loaded**	
type	*Must be "pointer"**	

Debugging tips

Loading hang-ups

If you are *preload*-ing an ***image***, ***xml*** or ***map*** resource (the default), VisualEyes will wait to start until the resource has been loaded. Unfortunately, it is hard to know just *which* resource doesn't load if it hangs. Hitting the [esc] key will stop the waiting a place a log of the loading process into your clipboard. To view it, open up a text editor to see which one started loading, but never finished.

Saving Versions

You should periodically save a version using *Save version* option in VisEdit's File menu. It is sometimes easier to go back to a previous version of your project than figuring out what you did wrong. Similarly, you may want to keep a local copy of the XML script on your computer. To do this:

1. Select the *Edit XML View* option from the menu in the bottom-right in VisEdit.
2. The XML for your project will appear in the window.
3. Select all the text using CTRL-A.
4. Copy the text to the clipboard using CTRL-C
5. Open up a text editor (TextEdit, NotePad, Word, etc.)
6. Paste the text into a the text editor using CTRL-V.

If ever you need to use the version, reverse the steps:

1. Open up a text editor containing your script.
2. Select all the text using CTRL-A.
3. Copy the text to the clipboard using CTRL-C
4. Select the *Edit XML View* option from the menu in the bottom-right in VisEdit.
5. Paste the text into a the text editor using CTRL-V.
6. Select the *Main Tree View* option from the menu in the bottom-right in VisEdit.

Controlling an Arduino

VisualEyes can control an Arduino processing card to read and write its various analog and digitals pins. Because Internet security prevents a Flash application to talk directly with your computer's USB port, you need to install a serial proxy server and use the AIR version of VisualEyes. Both parts can be found as a zip file here: ***www.viseyes.org/arduino.zip***.

Install the Arduino software from their site and make sure it is connected and configured properly. Double click on the ***VisualEyes.pde*** file to load the sketch into the Arduino. To verify that it's working, click on the ***Serial monitor*** button, and set the baud rate to 57600, and the state of the pins should stream in the monitor.

Setup for Macintosh

1. Unzip the ***arduino.zip*** file into a folder called ***VisualEyes*** within your ***Applications*** folder.
2. With the Arduino connected and running, run Arduino.app
3. Open the Tools/Serial port option from the menu. There will be a line that starts with /dev/cu
4. Write this name down
5. Double click on the file named ***serproxy.osx.cfg***. It will appear in the TextEdit app.
6. You will see a line that says: ***serial_device1=/dev/cu/usbmodem411***
7. Replace the ***dev/cu/usbmodem411*** portion with the name you wrote down.
8. Save the file
9. Double click on **StartProxy.command** file
10. It will start the Terminal App and should say ***Waiting for clients.***
11. Double click on the **VisualEyes.air** app and install it in the folder.
12. Double click on the ***VisualEyes.app*** in the ***VisualEyes folder*** and enter the project number you want to run.

Setup for Windows

1. Unzip the ***arduino.zip*** file into a folder called ***VisualEyes*** within your ***Progam Files*** folder.
2. With the Arduino connected and running, run Arduino.app
3. Double click on s**etproxy.exe** file
4. It will start the Command monitor and should say ***Waiting for clients.***
5. Double click on the **VisualEyes.air** app and install it in the folder.
6. Double click on ***VisualEyes*** in the ***Start menu*** and enter the project number you want to run.

Getting values from the Arduino

When the Arduino is running, the arduino(read) method will poll the card and retrieve the values on the pins in a global list called ***$$arduino.*** The first 6 items are analog pins 0-5 and the last 5 items are the digital pins 2, 4, 7, 8, and 12.