ViewONE JavaScript API Manual

Version 3.0

Last Updated: 04th May 2005

Email: info@daeja.com
Web site: http://www.daeja.com

Contents

Introduction

ViewONE is a Java applet that extends your web browser so that you can view, zoom, magnify, scroll, pan, rotate and print your images and image documents quickly and easily.

This document is the ViewONE JavaScript Manual and covers how to use and what options are available for the ViewONE JavaScript API.

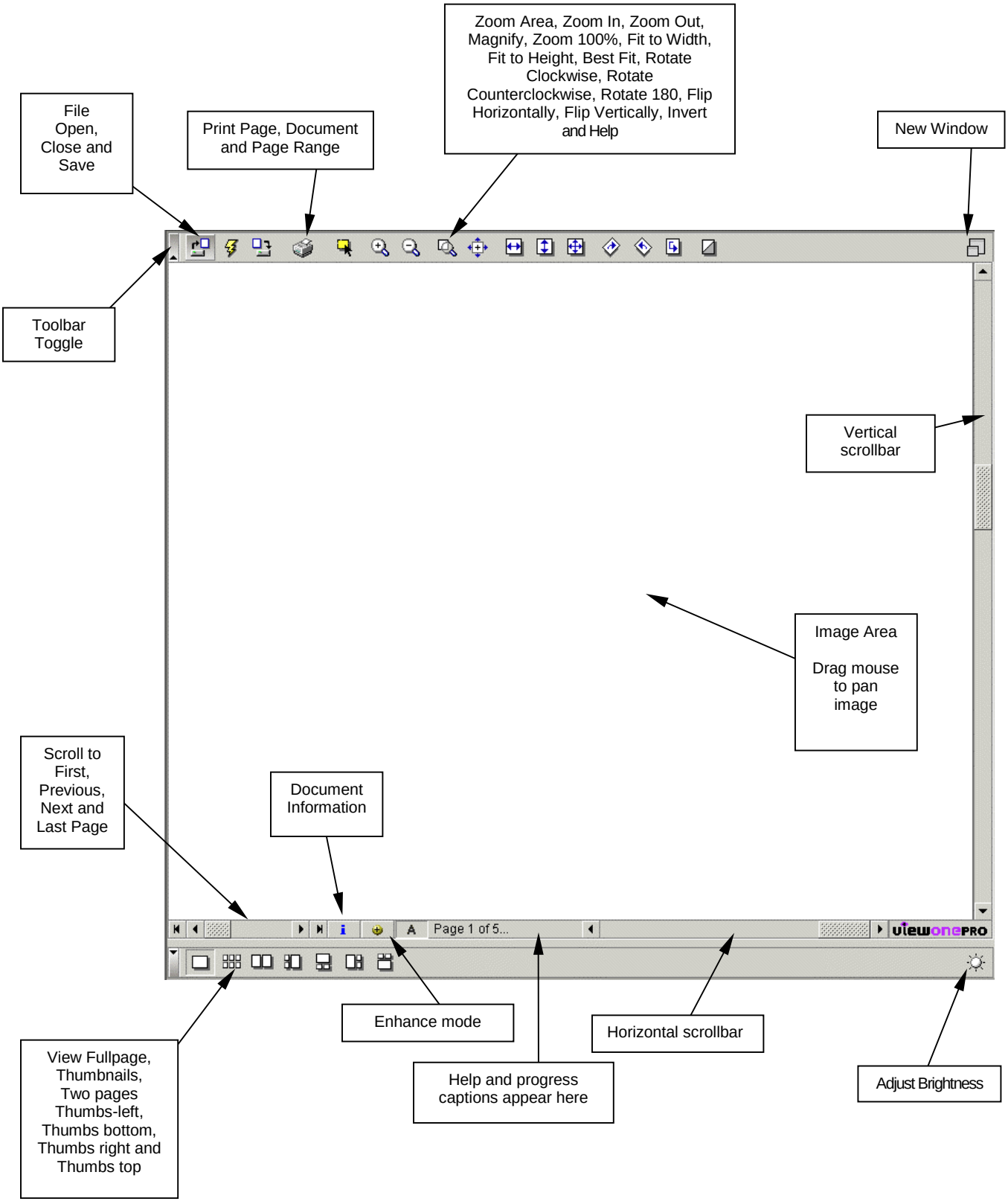
This document is designed to be used in conjunction with the ViewONE HTML and Installation Manual.

(Please note: This manual does not cover the Javascript API for Annotations, please see the Applet Annotations Configuration Manual)

For further information about ViewONE please consult the following documents…

- ViewONE User Manual
- ViewONE HTML and Installation Manual
- ViewONE Annotations User Manual
- ViewONE Annotations Installation HTML and JavaScript Manual
- Various White Papers for in-depth analysis of specific areas
- Latest ViewONE release notes found at http://www.daeja.com/pub/start/release.html

The Applet user interface



Installing ViewONE

HTML:

To install ViewONE on your web site you must first download the 'Update' zip file from the Daeja web site. This can be found by visiting www.daeja.com/pub/start/downloads.html

After unzipping the downloaded file you should have a directory named "**v1files**". This directory contains the essential files required to install and use ViewONE.

Copy **all** files found in the **"v1files"** directory to your web server, preferably in a separate directory similarly named.

ViewONE Standard

If you have downloaded ***ViewONE Standard*** (with or without the Print Accelerator or Annotations optional modules), then to use ViewONE on your web site, setup a web page containing the following HTML…

```
<APPLET CODEBASE = "."
ARCHIVE = "ji.jar"
CODE = "ji.applet.jiApplet.class"
NAME = "ViewONE"
WIDTH = "100%"
HEIGHT = "97%"
HSPACE = "0"
VSPACE = "0"
ALIGN = "middle"
MAYSCRIPT="true">
<PARAM NAME="cabbase" VALUE="ji.cab">
</APPLET>
```

ViewONE Pro

If you have downloaded ***ViewONE Pro*** (with or without the Print Accelerator, Annotations, DjVu, PDF or other optional modules), then to use ViewONE Pro on your web site, setup a web page containing the following HTML…

```
<APPLET CODEBASE = "."
ARCHIVE = "ViewONE.jar"
CODE = "start.jiViewONE.class"
NAME = "ViewONE"
WIDTH = "100%"
HEIGHT = "97%"
HSPACE = "0"
VSPACE = "0"
ALIGN = "middle"
MAYSCRIPT="true">
<PARAM NAME="cabbase" VALUE="ViewONE.cab">
</APPLET>
```

In each case, you must change the "codebase" value to point to the location of the 'v1files' directory that contains ViewONEs resource files (i.e. ji.jar, ji.cab and other .v1 files).

The codebase parameter specifies the location of the ji.jar, ji.cab and other files relative to the location of the HTML page. So in this case it specifies the "current directory" which is "."

(Note: PARAM NAME=cabbase" is required for Microsoft Explorer users).

Our downloadable demos include examples to assist you.

Mayscript parameter

When using JavaScript it is important to include the MAYSCRIPT parameter (see both HTML samples above). This parameter is an enabler for the JavaScript interface. Without it, calls to JavaScript will not work.

Apple Macintosh users (JavaScript)

Depending on the version of the operating system used by Apple Macintosh computers, JavaScript may or may not operate correctly.

Apple MacOS 8 and 9 computers utilize Apples Java engine known as MRJ (Macintosh Runtime Java). This particular Java engine did not have full support for JavaScript to Java Applets and so the JavaScript API will not operate with MRJ.

Apple MacOS X (10.x) computers utilize Apples updated Java 2 engine which does include support for JavaScript to Java API's.

Apple Macintosh users (HTML)

ViewONE comes with a digital certificate which is required to permit the applet to print and provide local caching. ViewONE is also Java 1.1.5 compliant.

Apple Macintosh 9 users will need to use IE4.5 (or later) and Apple Macintosh X (10.x) can use IE5, Netscape 7 or Mozilla.

IE uses Apples run-time Java (called MRJ). Since the release of IE4.5 Apple have released several versions of MRJ. It is recommended that users use MRJ version 2.2 or later. Netscape 7 and Mozilla browsers use Java 1.4.x which is supported by ViewONE.

Using ViewONE Standard on IE for Apple Macintosh

To use IE (MRJ Java) you will have to ensure your HTML specifies the alternative archive file…

ARCHIVE = "jis.jar"

Using ViewONE Pro on IE for Apple Macintosh

To use IE (MRJ Java) you will have to ensure your HTML specifies the alternative archive file…

ARCHIVE = "ViewONEdsa.jar"

We have produced a demo which uses some JavaScript to detect Apple Mac users and make appropriate changes automatically.

Demos are available at http://www.daeja.com/pub/start/downloads.html

Setting up Windows 2003 server

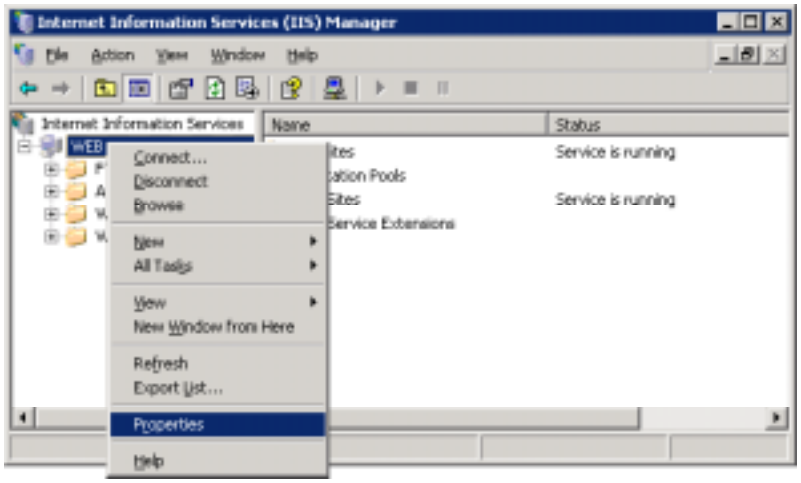
If your server is running Windows 2003 with IIS 6, you will have to add two file extensions to your web site “MIME Types”.

Unlike previous Windows servers, Windows 2003 permits only a limited number of file extensions by default, all others must be enabled manually using the IIS console.

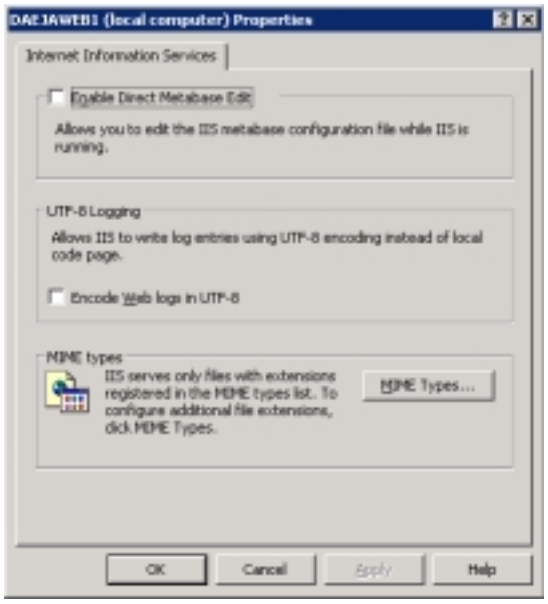
The additional mime types required to operate ViewONE are “.v1” and “.ant”. Note: if they are not added to IIS 6 then ViewONE will display messages during start-up that files with these extensions could not be found).

Both file extensions should be set to the mime type “application/octet-stream”, as follows..

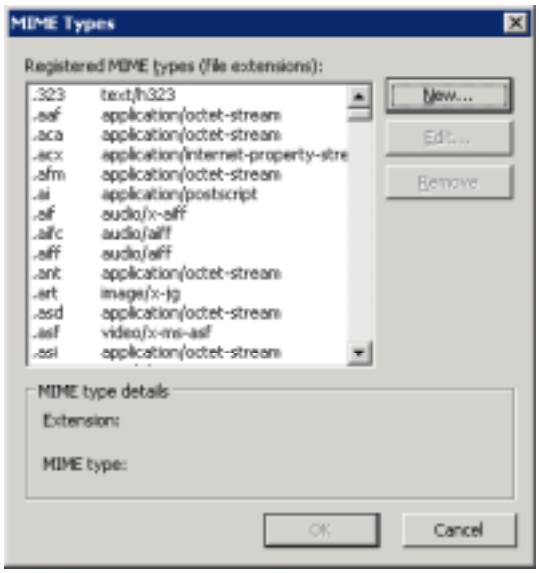
Open up IIS 6 console, then right-click on the top web space entry to highlight “Properties”…



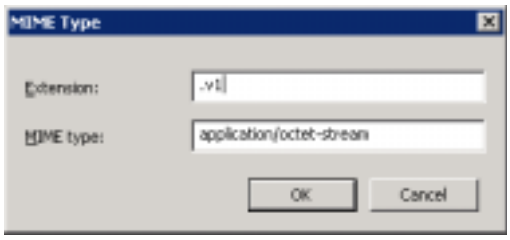
Click on the properties menu to display the properties dialog…



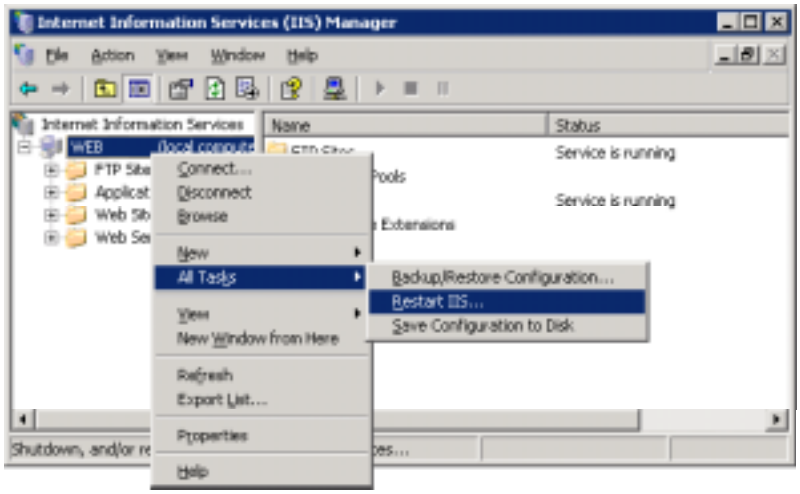
Click on the MIME Type button to display the MIME Type dialog…



Use the New button to add two new MIME types…



Click OK until you return back to the main console dialog. Right click once again on the main web space entry. Click on the Restart IIS menu to activate the new MIME Types.



ViewONE JavaScript Overview

The ViewONE Javascript API offers an alternative mechanism to configure and control the ViewONE applet.

In many cases the use of the Javascript API will not be required simply because most configuration parameters can be implemented using simple HTML (see ViewONE HTML manual).

However, where a more dynamic operation is required, for example, where it is desired to configuration parameters during the operation of ViewONE, perhaps to change a document without reloading a web page, rotate a page, invert etc then the JavaScript API is ideal.

Additionally, traditional web viewing applications habitually reload a web page in order to view or change the currently viewed image/document.

But with ViewONE's extensive Image/Document open/close JavaScript API methods it is possible to avoid web page refreshing entirely. This can result in significant performance improvements since all delays due to web page rehfreshing (and therefore Java Engine reloading and applet restarting) are avoided.

Daeja has produced white paper specifically addressing performance tuning which covers the area of web page refreshing and applet reloading. Please visit Daeja's web site to download white papers www.daeja.com.

Browser Note:

Please note, some browsers are case sensitive and so we advise the use of method names in exactly the same case as specified in this manual. Failure to adhere to this may result in methods not being called correctly.

JavaScript Reference

The JavaScript examples in this manual do not refer to their use in any particular context. The examples could be used within functions of a JavaScript program or directly as event handlers to buttons, hyper-links etc. Our web site illustrates such uses; alternatively refer to an appropriate JavaScript guide.

Filenames and hyperlink addresses are expressed using the Internet URL address format (Uniform Resource Locator), e.g. “http://mysite/myimage.tif”. If any part of the address before “myimage.tif” is not included then the applet will assume a base address that is the same as the applet location (the codebase).

With the exception of filenames and hyperlink addresses, all parameters are case insensitive.

Security note: Change since ViewONE 3.0.99 Standard and 1.0.99 Pro

Some Javascript methods are disabled by default and can only be enabled by setting the **JavaScripExtensions** HTML parameter set to **true**.

e.g. <PARAM NAME=“JavaScriptExtensions” VALUE=“true”>

This is to prevent unauthorized users from attempting to manipulate ViewONE through Javascript methods in order to obtain access to secure information/documents/annotations.

The methods that are restricted in this manner are clearly marked within this document.

General Applet Control

Method:

isReady()

E.g. var ready = ViewONE.isReady();

This method returns true if the applet as completed initialization.

If JavaScript is used to open a document during HTML page initialization then this method is useful in determining when ViewONE has fully initialized (prior to opening the document).

Browsers normally initialize Applets in parallel to HTML initialization and so it may be necessary to use this method to synchronize calls to ViewONE. Also, ViewONE itself runs thru an initialization phase in parallel to the loading of a web page.

Note on “object not found” message…

If you receive an “object not found” message while making this (or any JavaScript calls) then it is because the applet has not been started by the browser (ViewONE has not even had a chance to perform it’s own initialization). This message is produced by the browser (not the applet).

The solution is to add some JS code to the call of isReady() which handles object not found (or any use of JavaScript with ViewONE), as follows…

```
var appletReady = false;
var doc = parent.frames.main.document; //(or just 'document' if it's
not in a frame)

if (doc.ViewONE)
{
   appletReady = doc.ViewONE.isReady();
}

if (appletReady)
{
   //do something
}
```

The "`if (doc.ViewONE)`" line will result in false if the applet has yet to be started.

If you still have problems then you may want to make use of the "OnError" JavaScript statement that allows you to set up your own error handler. Your error handler will then be called when a JavaScript problem is encountered such as above. E.g..

```
onerror = errorHandler;

//you code to do whatever you need to do… then…

function errorHandler()
{
  //if we get here it is probably because a call has been made
  //to the applet before the browser has had time to initialize it
  //it can therefore be ignored
}
```

Method:

getVersion()

E.g. var version = ViewONE.getVersion();

Returns a String value representing the product version.

Opening documents and images

Method:

openFile(filename, page)

E.g. ViewONE.openFile("myimages.tif", 1);

Specifies the filename and initial page of the document to be viewed.

This filename can specify either the filename relative to the code base (as above) or the full URL. The code base is specified in the HTML code for the applet (see previous example). An example of a full URL is as follows…

E.g. ViewONE.openFile("http://mysite/myimages.tif", 1);

Method:

closeDocument()

E.g. ViewONE.closeDocument();

Closes an open document.

NOTE: Closing the open document also resets annotations, annotation templates and background images.

Method Group:

initializePageArray(numPages)
setPageArray(filename, page)
openPageArray(page)

E.g.

ViewONE.initializePageArray(3);

ViewONE.setPageArray("page1.tif", 0);

ViewONE.setPageArray("page2.tif", 1);

ViewONE.setPageArray("page3.tif", 2);

ViewONE.openPageArray(1);

This method group specifies the number files (pages) in a list, then specifies each file (each one representing a successive page of the document), then opens the 'assembled' document at page 1. Note the page array begins at array element zero.

Initializing the page and thumb arrays causes a ‘soft’ close to be performed, i.e. the current document is closed and the annotation source, templates and background images are reset if a document is currently open. If no document is currently open, then the close and reset is not performed.

Method Group:

initializePageAndThumbsArray(numPages)
setPageArray(filename, page)
setThumbsArray(filename, page)
openPageArray(page)

E.g.

ViewONE.initializePageAndThumbsArray(3);

ViewONE.setPageArray("page1.tif", 0);

ViewONE.setThumbsArray("page1-t.tif", 0);

ViewONE.setPageArray("page2.tif", 1);

ViewONE.setThumbsArray("page2-t.tif", 1);

ViewONE.setPageArray("page3.tif", 2);

ViewONE.setThumbsArray("page3-t.tif", 2);

ViewONE.openPageArray(1);

These methods are similar to the previous. They specify the number files (pages) in a list, and then specify a separate file for each page and a thumbnail file for that page. The final method then opens the "assembled" document at page 1. Note, the page array begins at array element zero.

In some instances, it may be advantageous to have separate files for the thumbnails to assist in browsing of thumbnails (smaller files are quicker to download and view).

Initializing the page anf thumb arrays causes a 'soft' close to be performed, i.e. the current document is closed and the annotation source, templates and background images are reset if a document is currently open. If no document is currently open, then the close and reset is not performed.

Method Group:

initializeDocumentArray(numDocs)
setDocumentArray(filename, index)
openDocumentArray()

E.g.

ViewONE.initializeDocumentArray(3);

ViewONE.setDocumentArray(“doc1parameters.txt”, 0);

ViewONE.setDocumentArray (“doc2parameters.txt”, 1);

ViewONE.setDocumentArray (“doc3parameters.txt”, 2);

ViewONE.openDocumentArray(1);

This method group specifies the number documents to open in a multi-document session, then specifies the parameter file for each document (each one representing a successive document), then opens the ‘assembled’ document list at document t1. Note the document array begins at array element zero.

For information about using document paramater files, please refer to the ViewONE HTML Installation Manual; look up the section on the “Doc<N>” HTML parameter.

Initializing the page anf thumb arrays causes a ‘soft’ close to be performed, i.e. the current document is closed and the annotation source, templates and background images are reset if a document is currently open. If no document is currently open, then the close and reset is not performed.

Method:

openList(listFile, page)

E.g. ViewONE.openList(“mylist/list.txt”, 1);

This method offers and alternative option to the page array methods above. It allows a file to be supplied which contains a list of pages.

This is useful for very large documents because it removes the need to deal with an array in JavaScript.

It can also be used to keep the HTML constant, by changing the source list instead of changing the HTML between different documents.

(Separate files for thumbnails are not available for this option).

Method:

reloadList()

E.g. ViewONE.reloadList()

This method forces the list used with the openList() method to be reloaded and the document to be re-opened. It will reload the list file from source (i.e. the web server) each time, so if it has changed then the changes will be picked up.

Method:

openDoc(index)

E.g. ViewONE.openDoc(2)

This method applies only when the “doc<N>” HTML parameter is used (see HTML manual). The value of “index” represents the associated “doc<N>” parameter. Therefore, the above example will cause ViewONE to open the second document in the list (i.e., that specified by the “doc2” HTML parameter).

Method Group:

nextDoc()
prevDoc()
firstDoc()
lastDoc()
getDocIndex()
getNumDocs()

These are convenience methods that can be used in place of the “openDoc” method described above.

Method:

getNumPages()

E.g. var numPages = ViewONE.getNumPages();

Gets the number of pages in the current document (an integer).

Method:

setPage(page)

E.g. ViewONE.setPage(2);

Sets the current page number (an integer).

Method:

getPage()

E.g. var page = ViewONE.getPage();

Returns the current page number as an integer.

Method:

nextPage()

E.g. ViewONE.nextPage();

Convenience method to view the next page (current page + 1)

Method:

previousPage()

E.g. ViewONE.previousPage();

Convenience method to view the previous page (current page - 1).

Method Group:

setDescription(description)
getDescription()

E.g.

var desc = ViewONE.getDescription();

ViewONE.setDescription("myDoc");

Gets or sets the description for the document. The description is used when opening the document in ViewONE, "Opening docXYZ…" or "Opening docXYZ, page 2…." will be displayed in the status bar.

Method:

getDocReference()

E.g.

Var ref = ViewONE.getDocReference();

Gets the document reference for the document.

Method:

getCacheFile()

E.g. var file = ViewONE.getGetCacheFile();

Returns a String value representing the local filename of the displayed image.

If the file was loaded locally then this value will be the actual local image file. If the file was loaded from a web server then this value will be the local 'cached' version of the image file.

*Important: This file must not be locked during read operations (or deleted, renamed. moved etc), as the applet will not be able to continue to use it and unpredictable results may occur.

isCacheAccessFailed ()

Returns true if access to the ViewONE cache fails. ViewONE may attempt to write to more than one location before and will only return true from this call if all attempts to write have failed.

Specifying an Annotations File

Method:

setAnnotationFile(filename)

E.g. ViewONE.setAnnotationFile(“http://mysite/myannotations.ant”);

This method sets the annotation file and must be called before any of the open methods described above.

If the current document is closed after this method has been called, the annotation file will be reset. To avoid this, close the current document before setting the annotation file, as closing of the current document can be a side effect of calling other JavaScript methods, for example, ‘initializePageArray’.

e.g.

viewONE.setAnnotationFile("http://...annotations.ant");

viewONE.openFile(“http://...mydocument.tif”, 1);

or

viewONE.closeDocument();

viewONE.setAnnotationFile("http://...annotations.ant");

viewONE.initializePageArray(2);

viewONE.setPageArray("http://....page1.tiff", 0);

viewONE.setPageArray("http://....page2.tif", 1);

viewONE.openPageArray(1);

Saving documents and images

Method: **save(filename)**

E.g. ViewONE.save("c:/temp/image.tif");

Saves the current document (multi-page tiffs), or current page (multi-file documents) to the specified filename.

If filename is null, either the save dialog is displayed or, if a server side save component has been specified, the image is sent to the server side component.

Note: The save operation does not recreate the source file, it merely copies the source file to the specified destination and renames it to the specified filename.

Method: **savePage(filename)**

E.g. ViewONE.savePage("c:/temp/page.tif");

Saves the current page to the specified filename.

If filename is null, either the save dialog is displayed or, if a server side save component has been specified, the image is sent to the server side component.

If the document is a multi-page tif then this method will extract the current page from the source tif file, and create a new tif file containing only the current page.

Method: **saveSelected(filename)**

E.g. ViewONE.saveSelected("c:/temp/selected.tif");

Saves the current selected pages of a multi-page tif document.

If filename is null, either the save dialog is displayed or, if a server side save component has been specified, the image is sent to the server side component.

Note: this method is for use only with multi-page tif documents. If you are viewing a multi-file document then it is not possible to save selected pages from that document except by individually calling the savePage() method for each page that you want to save. You may find isMultipageTif() and getSelected() methods useful to call before calling this method.

Method:

isMultipageTif()

E.g. vat ismulti = ViewONE.isMultipageTif();

This method is useful if you want to first check whether the current document is a multi-page tif before using the above methods.If it is then this method returns true else it returns false.

Document hyperlinks

Method:

setHyperlink(url, dblClick)

E.g. ViewONE.setHyperlink("newpage.html", false);

or

ViewONE.setHyperlink("http://mysite/newpage.html", false);

Specifies a hyperlink that is activated when the user clicks on the image area. If the dblClick parameter is 'true' then the hyperlink is activated only after the user double-clicks on the image area, otherwise it requires only a single click.

The hyperlink can specify either the filename relative to the code base or the full URL as illustrated.

Method:

clearHyperlink()

E.g. ViewONE.clearHyperlink();

This method clears a hyperlink if one has previously been defined using the setHyperlink() method.

PDF Viewing Options (Pro-Only)

ViewONE Pro introduced an optional PDF module. The module comes with various optional methods as follows...

Method:

(Pro-Only)

setPDFPixelDepth(pixeldepth)

E.g. ViewONE.setPDFPixelDepth(8);

This method will change the default pixel depth when viewing PDF documents. The pixel depth dictates the number of colors that are used for viewing as follows:

PixelDepth

1: View in Monochrome format (black and white)
8: View in Grayscale (256 grays)
24: View in full color (4.2 million colors).

For higher performance (but less colors) choose 1 or 8, for all colors (but slower performance) choose 24. The default value is 8.

Method:

(Pro-Only)

getPDFPixelDepth()

E.g. var depth = ViewONE.getPDFPixelDepth();

Returns the current pixel depth used when viewing PDF documents.

Method:

(Pro-Only)

setPDFResolution(resolution)

E.g. ViewONE.setPDFResolution(200);

This method will change the default resolution when viewing PDF documents. The minimum value is 50 and maximum is 1200.

For higher performance will result when using lower resolutions but less quality (depending on the PDF document), and lower performance but higher quality for higher resolutions.

Method:

(Pro-Only)

getPDFResolution()

E.g. var depth = ViewONE.getPDFResolution();

Returns the current resolution used when viewing PDF documents.

Method:

(Pro-Only)

setAutoLimitPDFResolution(true/false)

E.g. ViewONE.setAutoLimitPDFResolution(true);

When the parameter is set to true, ViewONE will dynamically limit the PDF resolution in order to obtain maximum performance when viewing PDF documents.

ViewONE will attempt to use the default PDF resolution (see above) but if that resolution is likely to result in slow performance, ViewONE will reduce the resolution on a per document and page basis depending on the setAutoLimitPDFMemoryValue() method described below.

Method:

(Pro-Only)

isAutoLimitPDFResolution()

E.g. var limited = ViewONE.isAutoLimitPDFResolution();

Returns true to indicate limiting is switch on and false if not.

Method:

(Pro-Only)

setAutoLimitPDFMemoryValue(value)

E.g. ViewONE.setAutoLimitPDFMemoryValue(10000000);

This methods controls the way ViewONE limits resolution when using the limiter described above.

The parameter dictates the maximum amount of memory that is permitted for a decompressed PDF page. If a page exceeds this amount (e.g. when using high default PDF resolutions) then ViewONE will use a resolution for that page that brings it under this memory limit. The actual resolution used will depend on the page dimensions and color depth. To discover the limited resolution a page must first be viewed, then you can display the Image Information dialog (see Image Information button).

The default value for this parameter is 5Mb (5 * 1024 * 1024 = 5242880 bytes).

If you find the limited resolution is resulting in degraded viewing quality, but you also wish to keep using the auto limiter, then try increasing the default value using this paramater.

The best value for you can only really be discovered by experimentation since it depends upon your documents, PC and network performance and user preferences.

Method:

(Pro-Only)

getAutoLimitPDFMemoryValue()

E.g. var limit = ViewONE.getAutoLimitPDFMemoryValue();

Returns the current limiting value used when viewing PDF documents.

COLD Viewing Options (Pro-Only)

ViewONE Pro introduced an optional COLD module.

The COLD module is responsible for implementing image merging and text formatting processes.

A COLD document (Computer Output to Laser Disk) is traditionally a document that has both text and image content and is usually used with forms packages.

During COLD forms' input, the fields are separated from the actual form (also known as the background or template) and during viewing they need reconstituting so that the user sees the complete form with completed fields.

The ViewONE COLD module allows any of its support files formats to be used either as a background template or foreground content.

For example, it is possible to have a TIFF or JPEG image as the background template and simple text as the foreground.

The COLD module is particularly effective when combined with the PDF module. It then becomes possible to have PDF used as either or both background and foreground.

Although more commonly one would use an image (typically TIFF based) as the background (thereby getting best performance and compression) and utilize the advanced text PDF formatting to handle the foreground (thereby able to create advanced forms for display).

Note: There are many proprietary COLD document formats in circulation. Daeja will look at any sample (with specifications) that you provide to assess the feasibility of adding specific format support to ViewONE. Currently the COLD module supports simple ASCII text and PDF document types.

The module comes with various optional parameters as follows...

Method:

(Pro-Only)

setBackgroundImage(filename, pageNumber)

E.g. ViewONE.setBackgroundImage(“mybackimage.tif“, 1);

This method is used to set the COLD background template for all pages in a document.

The pageNumber paramater is used where the file used for the backround supports multiple pages, such as TIF and PDF files.

Note: this method must be called before opening a document.

Method Group:

(Pro-Only)

initializeBackgroundImageArray(pages)
setBackgroundImageArrayItem(filename, index)
useBackgroundImageArray(page)

E.g.

```
ViewONE.initializeBackgroundImageArray(3);
ViewONE.setBackgroundImageArrayItem(“template1.tif”, 0);
ViewONE.setBackgroundImageArrayItem(“template2.tif”, 1);
ViewONE.setBackgroundImageArrayItem(“template3.tif”, 2);
ViewONE.useBackgroundImageArray(1);
```

This method group is used to specify a different COLD background template file for each page. If this technique is used then a template must be specified for all pages of your document.

The first methods specifies the number files (pages) in a list, followed by a series of methods that specify each template (each one representing a successive page of the document), the final method informs ViewONE that the templates are defined and ready to use.

Note: these methods must be called before opening a document.

Method:

(Pro-Only)

setBackgroundImageEnabled(true/false)

E.g. ViewONE.setBackgroundImageEnabled(false);

This method is used to change the visibility of the COLD template image. A value of false will remove the template from view, and a value of true will show the template.

Method:

(Pro-Only)

isBackgroundImageEnabled()

E.g. var inverted = ViewONE.isBackgroundImageEnabled();

Returns a Boolean 'True' if the COLD template is visible

Example JavaScript for opening a document with background templates

```
viewONE.initializeBackgroundImageArray(3)
viewONE.setBackgroundImageArrayItem("http://...backimage1.tif" ,0);
viewONE.setBackgroundImageArrayItem("http://...backimage2.tif" ,1);
viewONE.setBackgroundImageArrayItem("http://...backimage3.tif" ,2);
viewONE.useBackgroundImageArray(1);

viewONE.initializePageArray(3);
viewONE.setPageArray("http://....page1.tiff", 0);
viewONE.setPageArray("http://....page1.tif", 1);
viewONE.setPageArray("http://....page1.tif", 2);

viewONE.setAnnotationFile("http://...annotations.ant");

viewONE.openPageArray(1); //open at page 1
```

Document Indexes (Pro-Only)

ViewONE Version 3 introduced an option to specify a text index for multi-page documents. A text index acts as an alternative to the thumbnail pane and with large documents can offer better performance than thumbnails (due to not having to handle image decompression display).

ViewONE offers a default text index which displays page numbers; however the list may be customized using the following method.

Method:

(Pro-only)

setIndexListFile(URL)

E.g. ViewONE.setIndexListFile("http://mysite/mylist.txt");

The file specified must be a simple text file, with <LF> delimiting successive page indexes.

Image

Method:

invert()

E.g. ViewONE.invert();

Inverts the display colors (black changes to white and visa-versa). This method is also effective on images with more than two colors. A second call to this method will re-establish the original display colors.

If the document is closed when this method is called, the default for all pages is modified. If the document is open when this method is called, the individual page is modified only.

Method:

(V3-Only)

setInverted(true/false)

E.g. ViewONE.setInverted(true);

Sets the invert of the display colors (black changes to white and visa-versa). This method is also effective on images with more than two colors

If the document is closed when this method is called, the default for all pages is modified. If the document is open when this method is called, the individual page is modified only.

Method:

isInverted()

E.g. var inverted = ViewONE.isInverted();

Returns a Boolean ‘True’ if the colors are inverted, ‘False’ if they are not.

Method:

setEnhance(true/false)

E.g. var ViewONE.setEnhance(true);

Specifies whether a monochrome image is displayed with anti-aliasing on or off. A value of ‘true’(default) is on and ‘false’(default) is off.

Method:

isEnhance(true/false)

E.g. var enhance = ViewONE.isEnhance();

Returns a Boolean 'True' if enhance is on, 'False' if it is off.

Method:

setEnhanceMode(mode)

E.g. ViewONE.setEnhanceMode(1);

Specifies which anti-aliasing algorithm to use…

0: Off (all image formats)
1: Weighted averaging (monochrome), averaging (24 bit color)
2: Averaging (monochrome), averaging (24 bit color)
3: Scale-to-black (monochrome), averaging (24 bit color)

Note: This method can be used to turn on/off Anti-Aliasing and so there is no need to use both this and setEnhance methods together.

Method:

(V3-Only)

getEnhanceMode()

E.g. var state = ViewONE.getEnhanceMode();

Returns the current enhance state.

Method:

setRotation(angle)

E.g. ViewONE.setRotation(90);

Specifies the angle at which pages are displayed. Values of 90, 180, or 270 are accepted. The default is 0.

If the document is closed when this method is called, the default for all pages is modified. If the document is open when this method is called, the individual page is modified only.

Method: **initializeRotationArray(int size)**

E.g. ViewONE.initializeRotationArray(4);

Initializes the rotation array, used for setting the rotation of each page of the current image, to the given size.

Method: **setRotationArray(int angle, int page)**

E.g. var ok = ViewONE.setRotationArray(90, 0);

Sets the rotation for given page to the given angle. The pages are indexed from 0, so the first page is zero, the second is 1 etc.

The specified angle should be one of 0, 90, 180 or 270.

The JavaScript method initializeRotationArray should be used to set the size of the rotation array before this method is called.

The method returns true if the page rotation was successfully set or false if the set failed, for instance if the specified page is an invalid index in the rotation array initialized using initializeRotationArray.

Method: **applyRotationArray()**

E.g. ViewONE.applyRotationArray();

Applies the rotation array set up using the initializeRotationArray and setRotationArray methods to the current image.

Method: **getRotation()**

E.g. var angle = ViewONE.getRotation();

Returns the angle of rotation as an integer.

Method: **rotateClockwise()**

E.g. ViewONE.rotateClockwise();

Convenience method to increase the rotation by 90 degrees.

If the document is closed when this method is called, the default for all pages is modified. If the document is open when this method is called, the individual page is modified only.

Method: **rotateCounterclockwise()**

E.g. ViewONE.rotateCounterclockwise();

Convenience method to decrease the rotation by 90 degrees.

If the document is closed when this method is called, the default for all pages is modified. If the document is open when this method is called, the individual page is modified only.

Method: **rotate180()**

E.g. ViewONE.rotate180();

Convenience method to rotate the document to 180 degrees.

If the document is closed when this method is called, the default for all pages is modified. If the document is open when this method is called, the individual page is modified only.

Method: **setFlip(mode)**

E.g. ViewONE.setFlip(1);

Specifies the flip mode of displayed pages. Values of 0 (none), 1 (horizontal or mirror), 2 (vertical) or 3 (both) are accepted. The default is 0.

Flip buttons and menus are not visible to the user by default. To enable these options for the user you must use the flipOptions HTML tag when the applet is loaded.

If the document is closed when this method is called, the default for all pages is modified. If the document is open when this method is called, the individual page is modified only.

Method:

getFlip()

E.g. var angle = ViewONE.getFlip();

Returns the flip mode as an integer.

Method:

setScale(scale)

E.g. ViewONE.setScale(0);

Specifies the scale mode used to display a page. Acceptable integer values are:

0: best fit

The page is scaled to fit into the window area so that the entire page is visible.

1: Fit-to-window-width

The page is scaled so that the width of the page matches the width of the window area. This may result in the visible page height exceeding the available window height in which case a vertical scroll bar appears automatically.

2: Fit-to-window-height

The page is scaled so that the height of the page matches the height of the window area. This may result in the visible page width exceeding the available window width in which case a horizontal scroll bar appears

Method:

getScale()

E.g. var scale = ViewONE.getScale();

Returns the integer scale value.

Method:

getStates()

E.g. states = ViewONE.getStates();

Returns a coded string to be used with setStates(). When this method is called while a document it open, it returns a string containing information about the current zoom, scroll and other states.

Method: **setStates(string states)**

E.g. ViewONE.setStates(states);

Sets the zoom, scroll and other states to the values specified by the coded string. This method should be called before opening a document.

The getStates() and setStates() methods together permit the viewing states to be restored when a document is closed and re-opened.

Method: **zoomIn()**

E.g. ViewONE.zoomIn();

Applies a 25% increase in zoom. Note, at first ViewONE will attempt to use the scale modes (fit-to-width, fit-to-height and best-fit) if they are more appropriate. The zoomIn() function will increase the zoom factor only after the scale modes are no longer suitable.

Method: **zoomOut()**

E.g. ViewONE.zoomOut();

Reverses the effect of zoomIn().

Method: **zoom100()**

E.g. ViewONE.zoom100();

Zooms image to 100% (full resolution).

Method:

setZoom(zoom)

E.g. ViewONE.setZoom(2.0);

Zooms to the value specified. The value is specified as a double value - 1.0 = 100% or 1 image pixel = 1 screen pixel, 2.0 = 200% or 1 image pixel = 2 screen pixels.

If used in conjunction with setXYScroll, it is advisable to use setZoomAndXYScroll instead as problems may occur when both setZoom and setXYScroll are used together.

Method:

setZoomAndXYScroll(zoom, x, y)

E.g. ViewONE.setZoomAndXYScroll(2.0, 100, 100);

Zooms to the value specified. The value is specified as a double value - 1.0 = 100% or 1 image pixel = 1 screen pixel, 2.0 = 200% or 1 image pixel = 2 screen pixels

Method:

zoomArea(x, y, width, height, highlight, seconds)

E.g. ViewONE.zoomArea(400, 500, 100, 150, true, 2);

Zooms to an area starting at “x”, “y” which is “width” across and “height” high. If “highlight” is set “true” then the area zoomed is also highlighted for a time specified by the “seconds” parameter.

If “seconds” is greater than 0 then the highlight is visible for that time. If “seconds” is less than or equal to 0 then the highlight will remain visible until the user clicks or forces a refresh by scrolling, rotating, etc.

x, y, width, height = integers specifying image pixel values
seconds = integer specifying seconds
highlight = boolean

If the area specified is a different aspect ratio from the display area then the viewer will attempt to fit the zoom as best it can.

Method:

setXYScroll(x, y)

E.g. ViewONE.setXYScroll(20,20);

Sets the scroll bar positions to the given values in screen pixels.

If used in conjunction with setXYScroll, it is advisable to use setZoomAndXYScroll instead as problems may occur when both setZoom and setXYScroll are used together.

Method:

setDraggingEnabled(true/false)

E.g. ViewONE.setDraggingEnabled(true);

Specifies whether the dragging of the image is permitted or not (using the mouse). Dragging the image to the right pans the image to the right, dragging the image to the left pans the image to the left etc.

A value of 'true' (default) indicates dragging is permitted and 'false' indicates that it is not

Method:

isDraggingEnabled()

E.g. var draggingEnabled = ViewONE.isDraggingEnabled();

Returns a Boolean value of 'true' if dragging is allowed else a value 'false' is returned.

Method:

setBrightness(percent)

E.g. ViewONE.setBrighness(60);

This method sets the brightness of the displayed. The value represents a percentage from 0-100, with 50 being the default value. 0 = minimum brightness (dark) and 100 = maximum brightness (light).

If the document is closed when this method is called, the default for all pages is modified. If the document is open when this method is called, the individual page is modified only.

Method:

resetBrightness()

E.g. ViewONE.resetBrighness();

This method resets the brightness level to 50% (the default value).

If the document is closed when this method is called, the default for all pages is modified. If the document is open when this method is called, the individual page is modified only.

Method:

getBrightness()

E.g. var percent = ViewONE.getBrighness();

Returns the current brightness percentage setting (0-100).

Method:

setContrast(percent)

E.g. ViewONE.setContrast(40);

This method sets the contrast of the image displayed. The value represents a percentage from 0-100, with 50 being the default value. 0 = minimum contrast (flat) and 100 = maximum brightness (not flat!).

If the document is closed when this method is called, the default for all pages is modified. If the document is open when this method is called, the individual page is modified only.

Method:

resetContrast()

E.g. ViewONE.resetContrast ();

This method resets the contrast level to 50% (the default value).

If the document is closed when this method is called, the default for all pages is modified. If the document is open when this method is called, the individual page is modified only.

Method:

getContrast()

E.g. var percent = ViewONE.getContrast();

Returns the current contrast percentage setting (0-100).

Method:

setLuminance(percent)

E.g. ViewONE.setLuminance(70);

This method sets the luminance of the image displayed. The value represents a percentage from 0-100, with 50 being the default value. 0 = minimum luminance (dull) and 100 = maximum brightness (bright).

If the document is closed when this method is called, the default for all pages is modified. If the document is open when this method is called, the individual page is modified only.

Brightness increases the brightness of all colors (dark and light) uniformly, whereas luminance causes already bright areas of the image to increase in brightness further and darker areas to increase in brightness also, but by a lesser amount. This simulates a light source shining on the image and can be more effective at making color images clearer to read.

Method:

resetLuminance()

E.g. ViewONE.resetLuminance();

This method resets the luminance level to 50% (the default value).

If the document is closed when this method is called, the default for all pages is modified. If the document is open when this method is called, the individual page is modified only.

Method:

getLuminance()

E.g. var percent = ViewONE.getLuminance();

Returns the current lumance percentage setting (0-100).

Method:

getImageWidth()

E.g. var width = ViewONE.getImageWidth();

Returns an integer value representing the width of the currently displayed image in image pixels.

Method:

getImageHeight()

E.g. var height = ViewONE.getImageHeight();

Returns an integer value representing the height of the currently displayed image in image pixels.

Method:

getXResolution()

E.g. var xRes = ViewONE.getXResolution();

Returns an integer value representing the x-axis resolution of the currently displayed image in dots per inch.

The value is obtained from the image's header information, so if the information is missing or corrupt the returned value will make no sense.

Method:

getYResolution()

E.g. var yRes = ViewONE.getYResolution();

Returns an integer value representing the y-axis resolution of the currently displayed image in dots per image.

The value is obtained from the image's header information, so if the information is missing or corrupt the returned value will make no sense.

Method: **setView(view)**

E.g. ViewONE.setView(0);

Specifies the view mode used to display pages of a document. This method is effective only while a document is open. Acceptable integer values are:

0: Fullpage

(default): A single view of the current page is visible

1: Twopage

Two pages are visible at the same time

2: Thumbsonly

A view of the thumbnails only is visible.

3: Thumbsleft

A view of the current page with thumbnails on the left of the page.

4: Thumbsright

A view of the current page with thumbnails on the right of the page.

5: Thumbstop

A view of the current page with thumbnails at the top of the page.

6: Thumbsbottom

A view of the current page with thumbnails at the bottom of the page.

Method: **getView()**

E.g. var scale = ViewONE.getView();

Returns the integer view mode value.

Method:

setAreaZoom(true/false)

E.g. ViewONE.setAreaZoom (true);

If true, initiates the zoom-area mode. The mouse pointer changes to a cross and the user can drag the mouse (using button one) to select an area for zooming. When the mouse button is released the area selected will be zoomed as large as possible whilst maintaining the image aspect within the available window area.

If the selected area is not greater than the zoom trigger size (currently 20*20 pixels) then zooming will not occur. This allows the user to release the mouse if the mode was initiated accidentally.

If false, mouse functionality returns to drag mode (to pan the image).

Method:

isAreaZoom()

E.g. var areaZoom = ViewONE.isAreaZoom();

Returns a Boolean value indicating the zoom-area status.

Method:

(V3-Only)

toggleAreaZoom()

E.g. ViewONE.toggleAreaZoom();

This toggles ViewONE zoom area mode. When in zoom area mode the cursor will change to a cross hair and the user can select an area to zoom using the cursor. When not in zoom area mode, the cursor can be used to drag/scroll the image.

Method:

setMagnifier(true/false)

E.g. ViewONE.setMagnifier(true);

If true, displays an external magnifier window. A rectangle is visible around the mouse pointer which highlights the area being magnified.

If false, the magnifier window is hidden.

Method:

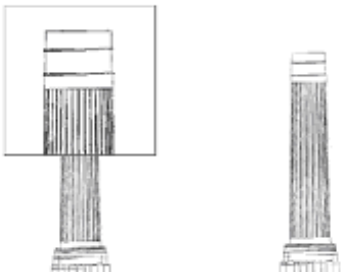
setMagnifierInternal(true/false)

E.g. ViewONE.setMagnifierInternal(true);

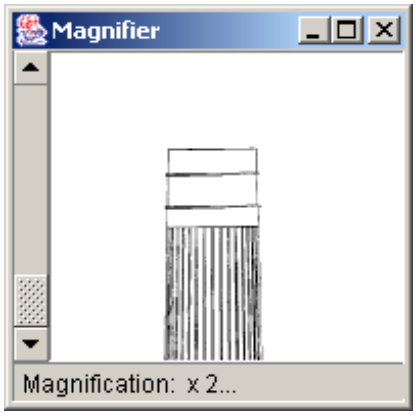
If true, displays a magnifier window internal to the display area.

If false, the magnifier window is hidden.

Internal…



External…



Method:

isMagnifier()

E.g. var magVisible = ViewONE.isMagnifier();

Returns a Boolean value indicating the magnifier visibility status.

Method:

(V3-Only)

toggleMagnifier()

E.g. ViewONE.toggleMagnifier();

Toggles the magnifier on/off.

Method:

setMagFactor()

E.g. ViewONE.setMagFactor(int factor);

Sets the current integer magnification factor (for magnifier window)

Method:

getMagFactor()

E.g. var factor = ViewONE.getMagFactor();

Returns the current integer magnification factor (for magnifier window)

Method:

setMagBounds(int x, int y, int width, int height)

E.g. ViewONE.setMagBounds(10, 10, 300, 300);

Sets the external magnifier's window position and size.

Method:

setNewWindowVisible(true/false)

E.g. ViewONE.setNewWindowVisible(true);

Specifies whether to make the ViewONE new window visible. A value of 'true' makes the window visible and 'false' (default) makes it invisible.

Method:

isNewWindowVisible()

E.g. ViewONE.setNewWindowVisible(!ViewONE.isNewWindowVisible());

Returns a Boolean value of 'true' if the ViewONE new window is visible else a value 'false' is returned.

Method:

(V3-Only)

setImageForeColor(color)

E.g. ViewONE.setImageForeColor(0, 0, 0)

This method sets the default color used for the foreground (text) of monochrome images.

Colors are specified using the standard RGB values.

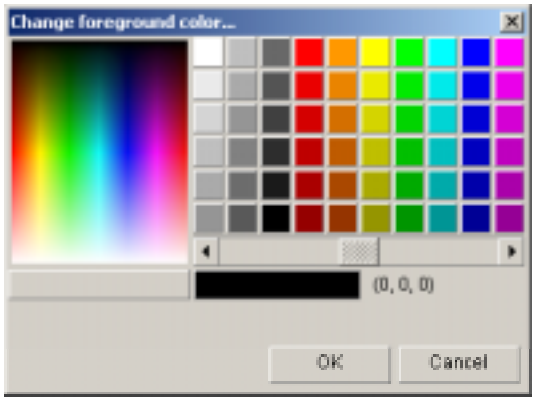
Method:

(V3-Only)

showImageForeColorDialog()

E.g. ViewONE.showImageForeColorDialog();

This method will cause ViewONE to display a dialog to allow the user to change the default color used for the foreground (text) of monochrome images.



Method:

(V3-Only)

setImageBackColor(color)

E.g. ViewONE.setImageBackColor(255, 255, 255)

This method sets the default color used for the background of monochrome images.

Colors are specified using the standard RGB values.

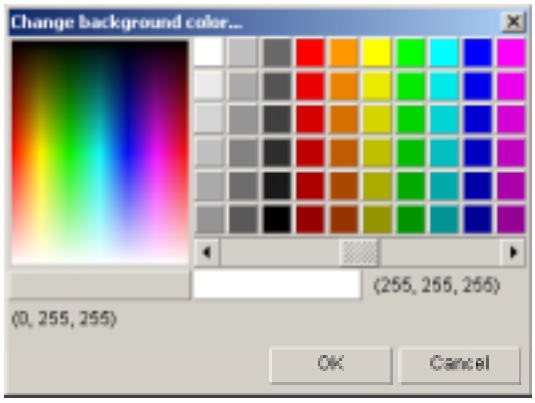
Method:

(V3-Only)

showImageBackColorDialog()

E.g. ViewONE.showImageBackColorDialog();

This method will cause ViewONE to display a dialog to allow the user to change the default color used for the background of monochrome images.



Labels

Method Group:

initializeLabels(numLabels)
setLabel(pageLabel, pageLabelColor, thumbLabel, thumbLabelColor, labelNum)
useLabels()
clearLabels()

E.g.

ViewONE.initializeLabels(3);

ViewONE.setLabel(“Page label 1”, null, “thumb 1”, null, 0);

ViewONE.setLabel(“Page label 2”, null, “thumb 2”, null, 1);

ViewONE.setLabel(“Page label 3”, “223,223,255”, “thumb 3”, “255,223,223”, 2);

ViewONE.useLabels();

ViewONE.openFile(“mydocument.tif”, 1);

This method group specifies the number files (labels) in a list, then specifies each label. Each one representing a successive page of the document and specifying the label to be displayed for the full-page area and the corresponding thumbnail. It then sets the labels in use by calling the useLabels() method.

Label colors are specified using the standard RGB values and where no color is specified (i.e. null) then the default color is white.

Notes:

The label array begins at array element zero.

Labels will remain visible until the document is closed or the clearLabels() method is called.

If you do not want to define a label for either the full-page area or a thumbnail then specify the label as a null string e.g…

ViewONE.setLabel(null, null, “thumb 1”, null, 0);

This example sets up a label for the thumbnail for the first page.

Selection and clipboard

Method: **selectPage(int pageNumber)**

E.g. ViewONE.selectPage(4);

Toggles the select property on the page in a document indicated by the “pageNumber” parameter (available for multi-page documents only).

Method: **clearSelections()**

E.g. ViewONE.clearSelections();

Clears all page selections in the document (available for multi-page documents only).

Method: **copyPageToClipboard()**

E.g. ViewONE.copyPageToClipboard();

Produces a print dialog to allow the user to print the current page.

Security note: This method is disabled by default unless the “JavascriptExtensions” parameter is set to “true”.

Method: **getSelection()**

E.g. selection = ViewONE.getSelection();

This method returns a comma-delimited string containing any pages selected by the user.

Printing

Method:

printPage()

E.g. ViewONE.printPage();

Produces a print dialog to allow the user to print the current page.

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

printDocument()

E.g. ViewONE.printDocument();

Produces a print dialog to allow the user to print the current document (available for multi-page documents only).

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

printRange()

E.g. ViewONE.printRange();

Produces a range dialog followed by a print dialog to allow the user to print a range of pages in a document (available for multi-page documents only).

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

printSelected()

E.g. ViewONE.printSelected();

Produces a print dialog to print pages selected using the page-select menu (available for multi-page documents only). Can be used with the "selectPage(pageNumber)" and "clearSelections()" methods to print any page or group of pages within a document.

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

printVisible()

E.g. ViewONE.printVisible();

Produces a print dialog to print the image display (visible).

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

printTransformed()

E.g. ViewONE.printTransformed();

Generates a print with rotate, invert and flip modes applied to the image.

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

setPrintDialog(true/false)

E.g. ViewONE.setPrintDialog(false);

This method applies only when the print-accelerator is used. When this parameter is set to false, printing will take place without showing the standard print-dialog. If the user has not printed using ViewONE with the accelerator previously, then the users' default printer will be used. Otherwise, the last printer used by the user (with ViewONE print accelerator) will be used.

Method: **isPrintDialog()**

E.g. var dialog = ViewONE.isPrintDialog();

Returns a Boolean ‘True’ if the print dialog is enabled, ‘False’ if it is not.

Method: **setPrintCopies(integer)**

E.g. ViewONE.setPrintCopies(2);

This method is effective only when the print dialog is disabled (by calling setPrintDialog(false) or using the equivalent HTML tag). This method sets the number of copies that will be printed when printing a page, pages or the document.

Method: **setPrinter(string)**

E.g. ViewONE.setPrinter(“myprinter”);

This method is effective only when the print dialog is disabled (by calling setPrintDialog(false) or using the equivalent HTML tag). This method sets the ViewONE default printer to the one specified as the parameter. The parameter must be the ‘name’ (or unique part thereof) as seen by the users printer settings. The printers’ default settings will be used for each print (e.g. orientation, resolution etc).

Method: **setPrintHeader(headerString)**

E.g. ViewONE.setPrintHeader("$page # $of ##")

Printouts can include custom text at the top of each page. By default this text is set to the page number followed by the number of pages in the document. The following options are available…

"false" : This value will disable print headers

"any text" : This is the text that will appear at the top of each printed page. For example, it could be your own copyright for the documents being viewed or some other informational text.

"formatted text" : The text can include some limited formatted elements as follows:

$page : This will print the word "page" in the appropriate translation

$of : Similarly for the word "of"

$pages: Similarly for the word "pages"

: This will print the page number of the page being printed

: This will print the number of pages in the document

An example…

"$page # $of ## © Copyright blah 2000"

Note 1: The default value is: "($page # $of ##)"

Note 2: Print headers are available for Internet Explorer 4.01+, Netscape 4.06+ and the Java Plugin 1.3 (The Java Plugin 1.2.2 does not offer sufficient functionality to permit print headers and so they will not be seen if using this version of the plugin).

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method: **setPrintAutoRotate (true/false)**

E.g. ViewONE.setPrintAutoRotate(true)

This parameter is only available when used with the print accelerator and only applicable when the tag multiPrintNum is set to more than 1.

If this parameter is set to true, then ViewONE will attempt to rotate images automatically prior to printing so that as many images can be printed (vertically) on a printed page as possible. The default value for this tag is false.

Toolbars and Buttons

Method: **setScrollbars(true/false)**

E.g. ViewONE.setScrollbars(true);

Specifies whether scrollbars will appear when the image is scaled to a size larger than the display area. A value of 'true' (default) indicates scrollbars are required and 'false' indicates they are not. A change in this setting will be visible after the next refresh of the display (e.g. when a page is zoomed or unzoomed or a page is changed etc.).

Method: **isScrollbars()**

E.g. var scrollbars = ViewONE.isScrollbars();

Returns a Boolean value of 'true' if scrollbars are enabled else a value 'false' is returned.

Method: **setStatusBar(true/false)**

E.g. ViewONE.setStatusBar(true);

Specifies whether the statusbar is visible or not. A value of 'true' (default) indicates the statusbar is visible and 'false' indicates that it is not.

Method: **isStatusBar()**

E.g. var statusBarVisible = ViewONE.isStatusBar();

Returns a Boolean value of 'true' if the statusbar is visible else a value 'false' is returned.

Method:

setFileButtons(true/false)

E.g. ViewONE.setFileButtons(true);

Specifies whether the toolbar includes file buttons. A value of 'true' (default) indicates the buttons are visible and 'false' indicates they are not.

The file buttons are:

Open, Close, Save.

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

isFileButtons()

E.g. var buttonsVisible = ViewONE.isFileButtons();

Returns a Boolean value of 'true' if the file buttons are visible else a value 'false' is returned.

Method:

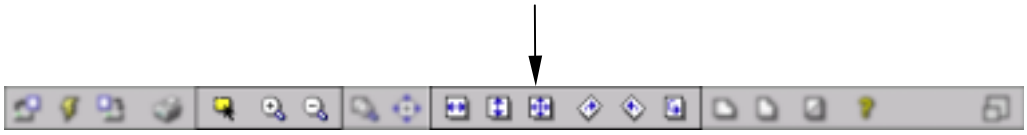
setImageButtons(true/false)

E.g. ViewONE.setImageButtons(true);

Specifies whether the toolbar includes image buttons. A value of 'true' (default) indicates the buttons are visible and 'false' indicates they are not.

The image buttons are:

Zoom area, Zoom in, Zoom out, Fit to width, Fit to height, Best fit, Rotate clockwise, Rotate counterclockwise, Rotate 180.



Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

isImageButtons()

E.g. var buttonsVisible = ViewONE.isImageButtons();

Returns a Boolean value of 'true' if the image buttons are visible else a value 'false' is returned.

Method:

setPrintButtons(true/false)

E.g. ViewONE.setPrintButtons(true);

Specifies whether the toolbar includes a print button. A value of 'true' (default) indicates the button is visible and 'false' indicates it is not.

The print button is:

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

isPrintButtons()

E.g. var buttonVisible = ViewONE.isPrintButtons();

Returns a Boolean value of 'true' if the print button is visible else a value 'false' is returned.

Method:

setInvertButtons(true/false)

E.g. ViewONE.setInvertButtons(true);

Specifies whether the toolbar includes an invert button. A value of 'true' (default) indicates the button is visible and 'false' indicates it is not.

The invert button is:

Method:

isInvertButtons()

E.g. var buttonVisible = ViewONE.isInvertButtons();

Returns a Boolean value of 'true' if the invert button is visible else a value 'false' is returned.

Method:

setNewWindowButtons(true/false)

E.g. ViewONE.setNewWindowButtons(true);

Specifies whether the toolbar includes a new-window button. A value of 'true' (default) indicates the button is visible and 'false' indicates it is not.

The new-window button is:

Method:

isNewWindowButtons()

E.g. var buttonVisible = ViewONE.isNewWindowButtons();

Returns a Boolean value of 'true' if the new-window button is visible else a value 'false' is returned.

Method:

setViewButtons(true/false)

E.g. ViewONE.setViewButtons(true);

Specifies whether the toolbar includes view buttons. A value of 'true' (default) indicates the buttons are visible and 'false' indicates they are not.

The view buttons are:

Fullpage, Thumbnails, Two-page, Thumbs-left, Thumbs-bottom, Thumbs-right, Thumbs-top.

Method:

isViewButtons()

E.g. var buttonsVisible = ViewONE.isViewButtons();

Returns a Boolean value of 'true' if the view buttons are visible else a value 'false' is returned.

Method:

setAllButtons(true/false)

E.g. ViewONE.setAllButtons(true);

Specifies whether all buttons are visible or not (these are file, print, image, new-window and view buttons). A value of 'true' (default) indicates the buttons are visible and 'false' indicates they are not.

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

isAllButtons()

E.g. var buttonsVisible = ViewONE.isAllButtons();

Returns a Boolean value of ‘true’ if the all buttons are visible else a value ‘false’ is returned (these are file, print, image, new-window and view buttons)

Method:

setPageButtons(true/false)

E.g. ViewONE.setPageButtons(true);

Specifies whether the toolbar includes page buttons. A value of ‘true’ (default) indicates the buttons are visible and ‘false’ indicates they are not.

The page buttons are:

First page, previous page, next page, last page and a drag bar for page selection..

Method:

isPageButtons()

E.g. var buttonsVisible = ViewONE.isPageButtons();

Returns a Boolean value of ‘true’ if the page buttons are visible else a value ‘false’ is returned.

Method:

(V3-Only)

toggleAdjustTool()

E.g. ViewONE.toggleAdjustTool();

This method toggles the visibility of the adjust tool (brightness/contrast/luminance).

Method:

(V3-Only)

setAdjustToolVisible(OnOff)

E.g. ViewONE.setAdjustToolVisible(True);

This method sets the visibility of the adjust tool (brightness/contrast/luminance).

Method:

(V3-Only)

isAdjustToolVisible()

E.g. var onOff = ViewONE.isAdjustToolVisible();

This method returns the visibility of the adjust tool (brightness/contrast/luminance).

Menus and keys

Method:

setFileMenus(true/false)

E.g. ViewONE.setFileMenus(true);

Specifies whether the file pop-up menus are available (using mouse button 2/3). A value of 'true' (default) indicates the menus are available and 'false' indicates they are not.

The menu is as follows:

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

IsFileMenus()

E.g. var menusEnabled = ViewONE.isFileMenus();

Returns a Boolean value of 'true' if the menus are enabled else a value 'false' is returned.

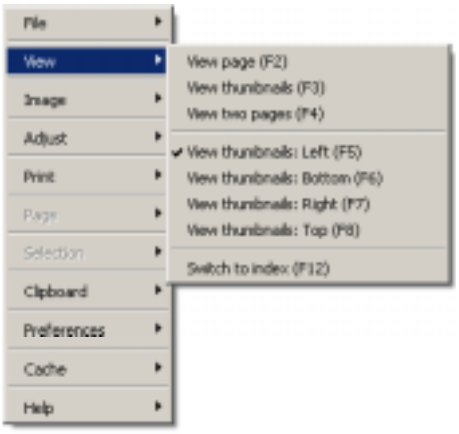
Method:

setViewMenus(true/false)

E.g. ViewONE.setViewMenus(true);

Specifies whether the view pop-up menus are available (using mouse button 2/3). A value of 'true' (default) indicates the menus are available and 'false' indicates they are not.

The menu is as follows:



Method:

isViewMenus()

E.g. var menusEnabled = ViewONE.isViewMenus();

Returns a Boolean value of 'true' if the menus are enabled else a value 'false' is returned.

Method:

setImageMenus(true/false)

E.g. ViewONE.setImageMenus(true);

Specifies whether the image pop-up menus are available (using mouse button 2/3). A value of 'true' (default) indicates the menus are available and 'false' indicates they are not.

The menu is as follows:

Method:

isImageMenus()

E.g. var menusEnabled = ViewONE.isImageMenus();

Returns a Boolean value of 'true' if the menus are enabled else a value 'false' is returned.

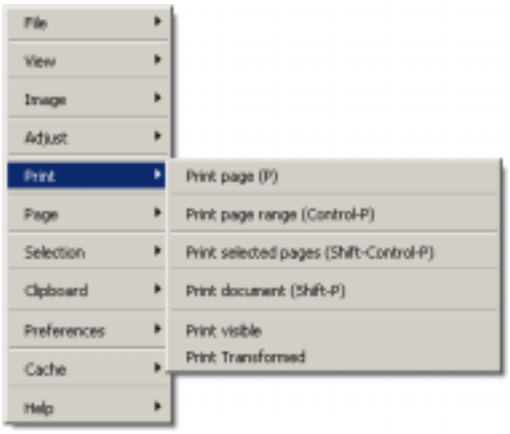
Method:

setPrintMenus(true/false)

E.g. ViewONE.setPrintMenus(true);

Specifies whether the print pop-up menus are available (using mouse button 2/3). A value of 'true' (default) indicates the menus are available and 'false' indicates they are not.

The menu is as follows:



Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

isPrintMenus()

E.g. var menusEnabled = ViewONE.isPrintMenus();

Returns a Boolean value of 'true' if the menus are enabled else a value 'false' is returned.

Method:

setPageMenus(true/false)

E.g. ViewONE.setPageMenus(true);

Specifies whether the page pop-up menus are available (using mouse button 2/3). A value of 'true' (default) indicates the menus are available and 'false' indicates they are not.

The menu is as follows:

Method:

isPageMenus()

E.g. var menusEnabled = ViewONE.isPageMenus();

Returns a Boolean value of 'true' if the menus are enabled else a value 'false' is returned.

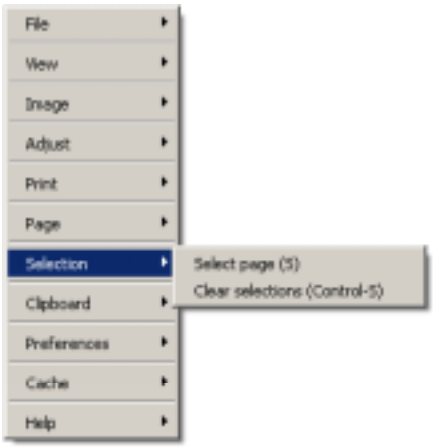
Method:

setSelectMenus(true/false)

E.g. ViewONE.setSelectMenus(true);

Specifies whether the select pop-up menus are available (using mouse button 2/3). A value of 'true' (default) indicates the menus are available and 'false' indicates they are not.

The menu is as follows:



Method:

isSelectMenus()

E.g. var menusEnabled = ViewONE.isSelectMenus();

Returns a Boolean value of 'true' if the menus are enabled else a value 'false' is returned.

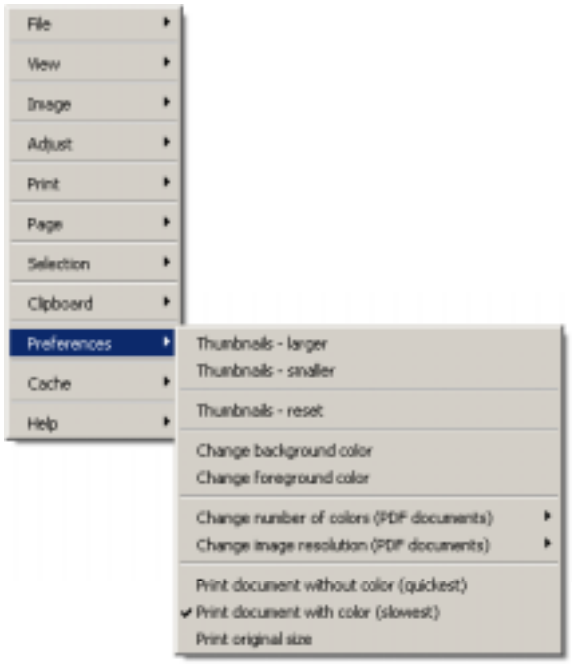
Method:

setPreferenceMenus(true/false)

E.g. ViewONE.setPreferenceMenus(true);

Specifies whether the preference pop-up menus are available (using mouse button 2/3). A value of 'true' (default) indicates the menus are available and 'false' indicates they are not.

The menu is as follows:



Method:

isPreferenceMenus()

E.g. var menusEnabled = ViewONE.isPreferenceMenus();

Returns a Boolean value of 'true' if the menus are enabled else a value 'false' is returned.

Method:

setAllMenus(true/false)

E.g. ViewONE.setAllMenus(true);

Specifies whether the all pop-up menus are available. A value of 'true' (default) indicates the menus are available and 'false' indicates they are not.

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

isAllMenus()

E.g. var menusEnabled = ViewONE.isAllMenus();

Returns a Boolean value of 'true' if the menus are enabled else a value 'false' is returned.

menus are available and 'false' indicates they are not.

Method:

setFileKeys(true/false)

E.g. ViewONE.setFileKeys(true);

Specifies whether the hot keys for file operations are enabled. A value of 'true' (default) indicates the keys are enabled and 'false' indicates they are not.

The keys are as follows:

Open File	O
Open URL	Shift-O
Close	C
Save	F
New window	W

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

isFileKeys()

E.g. var keysEnabled = ViewONE.isFileKeys ();

Returns a Boolean value of 'true' if the keys are enabled else a value 'false' is returned.

Method:

setImageKeys(true/false)

E.g. ViewONE.setImageKeys(true);

Specifies whether the hot keys for image operations are enabled. A value of 'true' (default) indicates the keys are enabled and 'false' indicates they are not.

The keys are as follows:

Fit to window width	F9
Fit to window height	F10
Fit to window	F11
Zoom in	Add
Zoom out	Subtract
Magnifier	M
Zoom to 100%	Z
Zoom area	A
Rotate clockwise	R
Rotate counterclockwise	L
Enhance	E
Invert	I

Method:

isImageKeys()

E.g. var keysEnabled = ViewONE.isImageKeys ();

Returns a Boolean value of 'true' if the keys are enabled else a value 'false' is returned.

Method:

setPrintKeys(true/false)

E.g. ViewONE.setPrintKeys(true);

Specifies whether the hot keys for print operations are enabled. A value of 'true' (default) indicates the keys are enabled and 'false' indicates they are not.

The keys are as follows:

Print page	P
Print document	Shift-P
Print page range	Ctrl-P
Print selected pages	Ctrl-Shift-P

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method:

isPrintKeys()

E.g. var keysEnabled = ViewONE.isPrintKeys ();

Returns a Boolean value of 'true' if the keys are enabled else a value 'false' is returned.

Method:

setViewKeys(true/false)

E.g. ViewONE.setViewKeys(true);

Specifies whether the hot keys for view operations are enabled. A value of 'true' (default) indicates the keys are enabled and 'false' indicates they are not.

The keys are as follows:

View page	F2
View thumbnails	F3
View two page	F4
View thumbnails: Left	F5
View thumbnails: Bottom	F6
View thumbnails: Right	F7
View thumbnails: Top	F8

Method: **isViewKeys()**

E.g. var keysEnabled = ViewONE.isViewKeys ();

Returns a Boolean value of ‘true’ if the keys are enabled else a value ‘false’ is returned.

Method: **setPageKeys(true/false)**

E.g. ViewONE.setPageKeys(true);

Specifies whether the hot keys for page operations are enabled. A value of ‘true’ (default) indicates the keys are enabled and ‘false’ indicates they are not.

The keys are as follows:

Next page	Page Down
Previous page	Page Up
First page	Home
Last page	End

Method: **isPageKeys()**

E.g. var keysEnabled = ViewONE.isPageKeys ();

Returns a Boolean value of ‘true’ if the keys are enabled else a value ‘false’ is returned.

Method: **setSelectKeys(true/false)**

E.g. ViewONE.setSelectKeys(true);

Specifies whether the hot keys for select operations are enabled. A value of ‘true’ (default) indicates the keys are enabled and ‘false’ indicates they are not.

The keys are as follows:

Select page	S
Clear Selections	Shift-S

Method:

isSelectKeys()

E.g. var keysEnabled = ViewONE.isSelectKeys ();

Returns a Boolean value of ‘true’ if the keys are enabled else a value ‘false’ is returned.

Method:

setAllKeys(true/false)

E.g. ViewONE.setAllKeys(true);

Specifies whether the all hot keys are enabled or not. A value of ‘true’ (default) indicates the keys are enabled and ‘false’ indicates they are not.

Security note: This method is disabled by default unless the “JavascriptExtensions” parameter is set to “true”.

Method:

isAllKeys()

E.g. var keysEnabled = ViewONE.isAllKeys ();

Returns a Boolean value of ‘true’ if the keys are enabled else a value ‘false’ is returned.

Timeout/User Idle Control

Method: **setTimeout(seconds)**

E.g. var ready = ViewONE.setTimeout(30);

This method sets and starts a usage timer. If the user does not use the applet for the number of seconds specified then the applet will automatically be disabled. It can be re-enabled by calling one of the timeout JavaScript methods (see below), opening a document using one of the JavaScript open methods, by revisiting the page containing the applet (Netscape) or reloading the page (Internet Explorer).

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method: **getTimeout()**

E.g. var seconds = ViewONE.getTimeout();

This method returns the timeout value (integer seconds) set using either the setTimeout() method or the HTML tag "timeout".

Method: **stopTimeout()**

E.g. ViewONE.stopTimeout();

This method will disable the timer set using either the setTimeout() method or the HTML tag "timeout" and if the applet had timed-out then it will wake-up (i.e. be re-enabled).

Security note: This method is disabled by default unless the "JavascriptExtensions" parameter is set to "true".

Method: **isTimedOut()**

E.g. var timedout = ViewONE.isTimedOut();

This method returns a value of true if the applet has timed-out as a result of the user not using the applet for the time specified by the setTimeout() method or the HTML tag "timeout". It otherwise returns false.

Method:

getTimeLeft()

E.g. var timeleft = ViewONE.getTimeLeft();

This method returns the time in seconds left before the applet times-out. but only if the setTimeout() method or HTML tag “timeout” has been used to set the time in the first place. It otherwise returns 0. The time left is automatically reset each time the user performs any action, such as scrolling, changing pages, zooming etc.

Method:

wakeUp()

E.g. ViewONE.wakeUp();

This method will wake-up the applet if it has timed out (see setTimeout()). The user will then be able to use the applet as normal.

Security note: This method is disabled by default unless the “JavascriptExtensions” parameter is set to “true”.

The Event Handler and Event Handling

ViewONE introduces the concept of applet JavaScript event handling. This is a mechanism by which it is possible to use JavaScript to monitor user activity and other selected actions performed by the ViewONE applet.

This can be a powerful facility, for example, it allows user activity and usage to be logged and/or actions to be performed dependent on what the user is doing/has done (e.g. charge for specific services like printing, perform actions on double-clicks, monitor documents opened etc).

The event handling is first enabled by the use of the HTML tag “eventHandler” and the MayScript tag (see example below). The eventHandler tag must be set to the name of a JavaScript function conforming to the following specification.

When the applet is initialized, it will call this function, passing it parameters relevant to the action performed as described below.

The JavaScript ‘event handler’ must be specified with two parameters; id (integer) and text (String), but can be any name of your choice, as follows…

```
Function myEventHandler(id, text)
{
  alert(“Event received, id=“+id+”, text=“+text);
}
```

In this example, the event handler just displays a notice to the user that an event had been received. You are however free to do whatever you need, for example testing the value of the id and text fields then taking further actions as necessary.

Events handler change in ViewONE Version 3

Prior to ViewONE Version 3, all events were signaled irrespective of whether the event was required. However, Version 3 introduced an event filter so that only those events that are desired are signaled to the event handler.

Version 3 requires an additional HTML parameter which specifies the event Id's that are desired (see below for full list of Id's).

* Note: Without this parameter no events will be signaled.

The parameter takes the form of a simple list, e.g…

<param name="eventInterest" value="9, 37, 39">

In this example, events 9, 37 and 39 are signaled.

Events ids and descriptions

ViewONE can signal many different events to the event handler.

The following list describes the events that will be received by this function (over page)…and the next page after provides a full example.

Id	Event Text	Description
1		Reserved.
2		Reserved.
3	printpage: page *n* of *n*	User has printed a page.
4	printvisible: page *n* of *n*	User has printed the visible part of a page
5	opened: *url*	User has opened a document
6	saved: page *n* of *n*	User has saved a page
7	click: page *n* of *n*	User has clicked on a page
8	dblclick: page *n* of *n*	User has double-clicked on a page
9	page: page *n* of *n*	User has changed page
10	timeout: page *n* of *n*	Indicates the applet has timed-out. See JavaScript method setTimeout() and HTML tag “timeout”
11		Reserved.
12	select page: page *n* of *n*	User has selected a page
13	unselect page: page *n* of *n*	User has unselected a page
14	mouse down: page *n* of *n*	User has pressed a mouse button
15	mouse up: page *n* of *n*	User has released a mouse button
16		Reserved.
17		Reserved.
18		Reserved.
19		Reserved.
20	set document: doc *n* of *n*	User has selected the next document in the list (this applies only when the “doc<N>“ HTML tag is used – see HTML manual)
21	end tab	The user has used the tab key to change focus while the last focusable component in ViewONE already had focus. ViewONE will assign focus back to the first in its list, however you may override ViewONE by using this

		event to switch focus to any alternative component on your web page.
22	ready	The applet has just been started and is ready to accept JavaScript calls.
23	annotation hyperlink	The user has activated an annotation (JavaScript) hyperlink (see annotations configuration manual).
24	annotations save ok	Annotations have just been saved.
25	annotations save failed	The annotations save operation has just failed.
26	print cancelled	A print dialog or print job has been cancelled.
27	print ended	A print job has been successfully sent to the printer.
28		Reserved.
29		Reserved.
30	annotation created	The user has added an annotation through the user interface.
31	SaveDocument Failed(*n*)	This event can fire with two different *n* values… - if *n* is 1, it means that ViewONE could not create the save files in its own cache (prior to sending them to the destination). One way this might happen is if the disk was full. - if *n* is 2, it means that ViewONE could not copy the save files to the specified destination.
32	annotations updated: page *n* of *n*	Gets fired whenever a document's annotations set is modified. It can, therefore, be used to indicate that the document's annotations now require saving.
33	annotations restored: page *n* of *n*	Gets fired whenever a document's annotations set is restored. It can, therefore, be used to indicate that the document's annotations no longer require saving.
34	Page rendered	The image for the current page has been rendered/updated. This event occurs on every redraw/refresh. See event 38/39 for filtered versions.
35	Area Selected	An area of the current page has been selected by the user (using the zoom area tool).
36		Reserved
37	Key pressed	A key has been pressed (key defined in the event). Notes: It is strongly recommended that you turn off key processing by ViewONE by using the HTML parameter ProcessKeys set to False, otherwise ViewONE will process keys AND generate events. It is also strongly recommended that you set up a test to display the event text before assuming that value of that text because some keyboards may differ in the text and may also differ according to the localization setup of your machine. The following keys do not generate events.. - Keys processed when editing a text annotation - SHIFT, CTRL and ALT Keys (instead, when a key is pressed a marker is included to indicate when these keys are pressed with other keys –see examples below)

<table>
<tr><td></td><td></td><td>- Windows main menu key
- Popup menu key

The following keys will generate events but will always also be handled by ViewONE event when ProcessKeys is set to False…

- Scroll bar keys (Page up/down, home, end, arrow keys)
- Windows menu key

Modifier and cursor keys are defined using names (see below)

Examples:

<table><tr><th>Event Text</th><th>Descriptio</th></tr><tr><td>“A”</td><td>A key pressed</td></tr><tr><td>Control A</td><td>Ctrl + A keys pressed</td></tr><tr><td>Shift A</td><td>Shift + A keys pressed</td></tr><tr><td>Alt A</td><td>Alt + A keys pressed</td></tr><tr><td>Alt Shift Control A</td><td>Alt Shift + Ctrl + A keys pressed</td></tr><tr><td>Space</td><td>Space key pressed</td></tr><tr><td>Ctrl Space</td><td>Ctrl + Space keys pressed</td></tr><tr><td>Caps Lock</td><td>Caps Lock pressed</td></tr><tr><td>Up</td><td>Up arrow key pressed</td></tr><tr><td>Numpad-9</td><td>9 key on a number pad is pressed (note some numpad keys have a ‘-‘ separator rather than a space)</td></tr></table></td></tr>
<tr><td>38</td><td>Full page rendered</td><td>A page has been viewed in the full-page panel. This event only occurs the first time the page is viewed.</td></tr>
<tr><td>39</td><td>All full pages rendered</td><td>All pages in the document have been viewed in the full-page panel. This event occurs only once per document.</td></tr>
<tr><td>40</td><td></td><td>Reserved</td></tr>
<tr><td>41</td><td>Keep Alive</td><td>This event is generated at regular intervals (specified by the KeepAliveTime HTML paramater). The text accompanying the event is specified by the KeepAlive HTML paramater.</td></tr>
<tr><td>42</td><td>Save Complete</td><td>Generated when a document is successfully saved.</td></tr>
<tr><td>43</td><td>Cache Access Denied</td><td>This event is generated if all attempts to write to a ViewONE cache have failed.</td></tr>
</table>

Event Handler example:

```
<html>
<head>
<title>ViewONE Event Handler Demo</title>
</head>
<body bgcolor="#C0C0C0" text="#000000" topmargin="0" leftmargin="0">

<script LANGUAGE="JavaScript">
<!--

function myhandler(id, text)
{
   alert(id + ", " + text);
}

//-->
</script>

<applet CODEBASE="../v1files"
ARCHIVE="ji.jar"
CODE="ji/applet/jiApplet.class"
NAME="ViewONE"
ID="ViewONE"
WIDTH="100%"
HEIGHT="97%"
HSPACE="0"
VSPACE="0"
ALIGN="middle"
MAYSCRIPT="true">

<param name="cabbase" value="ji.cab">
<param name="eventhandler" value="myhandler">
<param name="eventInterest" value="9, 37, 39">
<param name="backcolor" value="192,192,192">
<param name="filename" value="mydocument.tif">

</applet>
</body>
</html>
```

Testing your Event Handler

There may be cases where 'LiveConnect' is not implemented (older browsers or some browser implementations on non-MS Windows platforms) which will mean JavaScript cannot call applet methods and the event handler functionality will not work. In all other cases the following methods may help in debugging potential event handler problems…

You need to make sure the MayScript and eventHandler tags are specified. If you have made a mistake with either then the following methods will help indicate where the problem lies…

isEventHandlerOK()

getEventHandlerError()

And can be used as follows…

```
If (!document.ViewONE.isEventHandlerOK())
{
  alert(document.ViewONE.getEventHandlerError());
}
```

If you receive an error message when the applet attempts to call your event handler, then assuming LiveConnect is enabled, the most likely cause is that the incorrect name for your event handler was supply to ViewONE. Note that JavaScript function names are case sensitive.

The MayScript tag

The MayScript tag is required if any applet is to call JavaScript methods. This tag was introduced by Netscape and is also implemented in Internet Explorer. We have however noticed that the use of this tag can reduce start-up performance for Netscape 4 because it appears to force all applets to be recompiled with each invocation. Netscape 7 however does not suffer from this problem.