Aperio ImageServer

Programmer's Reference

User Resources

For the latest information on Aperio Technologies products and services, please visit the Aperio Technologies website at: http://www.aperio.com.

Disclaimers

Use normal care in maintaining and using the Spectrum servers. Interrupting network connections or turning off the Spectrum and DSR servers while they are processing data (such as when they are analyzing digital slides or generating an audit report) can result in data loss.

This manual is not a substitute for the detailed operator training provided by Aperio Technologies, Inc., or for other advanced instruction. Aperio Technologies Field Representatives should be contacted immediately for assistance in the event of any instrument malfunction. Installation of hardware should only be performed by a certified Aperio Technologies Service Engineer.

ImageServer is intended for use with the SVS file format (the native format for digital slides created by scanning glass slides with the ScanScope scanner). Educators will use Aperio software to view and modify digital slides in Composite WebSlide (CWS) format.

Aperio products are FDA cleared for specific clinical applications, and are intended for research use for other applications. For clearance updates, visit www.aperio.com.

Aperio grants to those users of Aperio's published APIs (TiffComp, Viewport, ImageNav, and other modules that may be made available) a nonexclusive right to use for your personal/internal use. Commercial redistribution is strictly prohibited. Non-commercial redistribution is permitted provided that its original configuration is not modified without Aperio's express written consent and its origin is not misrepresented.

Trademarks and Patents

ScanScope is a registered trademark and ImageServer, TMALab, ImageScope, and Spectrum are trademarks of Aperio Technologies, Inc. All other trade names and trademarks are the property of their respective holders.

Aperio products are protected by U.S. Patents: 6,711,283; 6,917,696; 7,035,478; 7,116,440; 7,257,268; 7,428,324; 7,457,446; 7,463,761; 7,502,519; 7,518,652; 7.602.524, 7,646,496; 7,738,688 and licensed under one or more of the following U.S. Patents: 6,101,265; 6,272,235; 6,522,774; 6,775,402; 6,396,941; 6,674,881; 6,226,392; 6,404,906; 6,674,884; and 6,466,690.

Contact Information

Headquarters	Europe Office	Asia Office
Aperio Technologies, Inc. 1360 Park Center Drive Vista, CA 92081 United States	Aperio UK Ltd Prama House 167 Banbury Road Summertown Oxford OX2 7HT United Kingdom	Aperio Technologies, KK UZ Building 5F 3-3-17, Surugadai Kanda, Chiyoda-ku Tokyo, Japan 101-0062
Tel: 866-478-4111 (toll free) Fax: 760-539-1116	Tel: +44 (0) 1865 339651 Fax: +44(0) 1865 339301	Tel: +81-3-3259-5255 Fax: +81-3-3259-5256
Customer Service: Tel: 866-478-4111 (toll free)	**Customer Service:** Tel: +44 (0) 1865 339651 Email: europeinfo@aperio.com	**Customer Service:** Email: asiainfo@aperio.com
Technical Support: US/Canada: Tel: 1 (866) 478-3999 (toll free) Direct International Tel: 1 (760) 539-1150 US/Canada/Worldwide: Email: support@aperio.com	**Technical Support:** Direct International Tel: 1 (760) 539-1150 Email: europesupport@aperio.com	**Technical Support:** Direct International Tel: 1 (760) 539-1150 Email: asiasupport@aperio.com

Aperio ImageServer Programmer's Reference

The Aperio ImageServer provides a high-performance interface to Aperio digital slide images stored as SVS files (TIFF files with JPEG2000 compression). Features include:

- Full compatibility with HTTP 1.1 – can be used as a web server.
- Support for returning directory contents and attributes.
- Support for returning file contents and attributes.
- Ability to return designated regions from an image – random access for any coordinates, dimensions, and zoom level.
- Support for large images (up to 250,000 pixels in width and height).
- Directory-level authentication using an authentication database.
- Interface to server-side image database.
- Support for storing and retrieving annotations encoded as XML strings.
- Interface for server-side algorithm processing.
- Integrated APML template processing for formatting HTML pages and XML messages[1].

ImageServer may be used as a web server. HTML and text files are returned "as is" for standard HTTP requests. Support is also provided for GIF, JPEG, and TIFF images—their image contents are returned "as is" with the appropriate MIME types. In addition, an integrated template processing facility supports APML, a simple superset of HTML. (For information on APML, see the *Aperio Markup Language (APML) Programmer's Reference*.)

Aperio grants to those users of Aperio's published APIs (TiffComp, Viewport, ImageNav, and other modules that may be made available) a nonexclusive right to use for your personal/internal use. Commercial redistribution is strictly prohibited. Non-commercial redistribution is permitted provided that its original configuration is not modified without Aperio's express written consent and its origin is not misrepresented.

Contents

[1]The Aperio Markup Language (APML) is documented separately, please see the *Aperio Markup Language (APML) Programmer's Reference*.

ImageServer provides two levels of access to image file information for digital slides (SVS files). First, a low level of "raw" access is provided for a high-speed interface to the Aperio **viewport** (and other client software using the cAIC subroutine). The **viewport** is an ActiveX control (OCX) that can be included in a variety of programs and configurations to display image files[2]. At this level of access, ImageServer functions as the server for a client/server interface to the image information. Second, a higher level of "formatted" access is provided for web interfaces. At this level, image information is returned as JPEG images for compatibility with standard web browsers. The Aperio WebViewer makes use of this level of access.

Installing ImageServer

ImageServer is provided as a self-installing executable. ImageServer may be installed by double-clicking the self-installer and following the displayed instructions. After installation, a program group named "ScanScope" will have been created with an ImageServer icon. Simply double-click the icon to run ImageServer.

The Aperio ImageScope viewer is also provided as a self-installing executable. This program may be freely distributed. ImageScope is installed by double-clicking the self installer and following the displayed instructions. After installation, a program group named "ScanScope" will have been created with an ImageScope icon. Simply double-click the icon to run ImageScope.

Image Directories

ImageServer serves images and other files from a single directory structure. Any directory may be specified as the "root" directory for ImageServer image serving. The URL path / will be associated with this directory. Subdirectories may be used to organize information and/or to control access. A subdirectory named mydir of the root directory would have the URL path /mydir. If desired, shortcuts or links may be used to link images and directories into the ImageServer directory structure. Full support is provided for shortcuts including network paths.

[2]The Aperio **viewport** is documented separately, please see the *Aperio Viewport Active-X (.OCX) Programmer's Reference*.

The ImageServer installer creates a folder named C:\Images which is the default "root" directory for serving images. A small SVS image is placed in this directory by the installer for test purposes. To use another directory as the "root" for serving images, you'll need to change the ImageServer configuration file, see below in the Configuration section.

Running ImageServer

To run ImageServer, select Start | Programs | ScanScope | ImageServer. You may alternatively double-click the ImageServer icon installed on the desktop. While ImageServer is running it appears as a minimized window in your taskbar. You may restore the window to view recent messages from the ImageServer logs.

To stop ImageServer, simply close the window. This can be done by right-clicking on the ImageServer taskbar icon and selecting **close**, or by clicking the close box of the program's window (if you have the window restored).

Aperio WebViewer

The Aperio ImageServer interfaces to a related software product called the Aperio WebViewer. This product consists of APML templates and HTML files and images which enable web pages to be generated displaying the contents of directories and for viewing image files. The WebScope includes a Flash application from Zoomify, Inc., called WebScope, which is specially configured to interface directly to digital slide data served up from ImageServer.
As of Aperio release 6, WebViewer *does not* require an external webserver such as Microsoft IIS and *does not* require Korn shell scripts. Additionally, WebViewer now uses *the same port* as ImageServer, so only one port is required for viewing digital slides on the Internet with a web browser[3].

ImageServer Authorization

ImageServer may use an authorization service for verifying access to directories. ImageServer connects to an [optional] database service to validate authorization requests. The service is identified by a host name and port number at startup.

Authorization credentials are supplied in HTTP requests using the Authorization: header. If no authorization is supplied or incorrect credentials are given, ImageServer will return an HTTP 401 response status ("authorization required"). Note that ImageServer is stateless, so authorization must be supplied on every ImageServer request. An authorization cache is used to mitigate the performance impact of verifying authorization.

The credentials are formatted as **userid:password**, and may be supplied in one of two ways:

[3]Please contact Aperio for more information about the changes to WebViewer in release 6.

```
Authorization: basic base-64-encoded-credentials
Authorization: aws IDEA-encrypted-credentials
```

The **basic** authorization mode is compatible with all standard web browsers. Typically, access is made to a resource without credentials, and a 401 response is returned indicating that authorization is required. The web browser prompts the user to enter a userid and password, and passes the supplied credentials back to ImageServer, where they are validated. Note that base-64 encoding is a light form of encryption and can easily be decoded.

The **aws** authorization mode is supported by the Aperio **viewport** OCX. Again, access is made to a resource without credentials, a 401 response is returned, and the **viewport** passes the status back to the calling application, which prompts the user to enter a userid and password. The credentials are passed back to ImageServer and validated. The **aws** mode uses IDEA 168-bit encryption, which is a strong form of encryption that cannot easily be decoded.

Server-Side Image Database

ImageServer can optionally interface to a server-side database containing digital slide metadata. The server database is interfaced by a service designated by a host name and port number at startup, which identify a database service used to access the database. This service may be the same as the authorization service described above, or may be different. Authorization is recommended but not required when an image database is used.

The server database is accessed via the GETDATA and PUTDATA requests (see "ImageServer HTTP Request/Response API," on page 19 for details). Image information is stored and retrieved using XML strings. All available data for an image are returned on a GETDATA request. When a PUTDATA request is made, only the data supplied in the request are updated; the other data in the database for the digital slide remain unchanged.

If a PUTDATA request is made for a slide *prior* to the slide being scanned, it is associated to slides *as they are scanned* via the barcode value and or physical location (rack/slot).

Following is the XML format used to store image data (closing tags are omitted for brevity):

```
    <IMAGE ID="imageId">
       <CASEID>caseId
       <RACK>rack#
       <SLOT>slot#
       <LOCATION>ACIfilePath
       <SCANDATE>scanDateTime
       <SCANSCOPE>ScanScopeName
       <SCANSTATUS>scanStatus
       <SCANSTATUSDETAILS>scanStatusDetails
       <QUALITYFACTOR>qualityFactor
       <CREATEDBY>operator
       <DESCRIPTION>description
       <COMPRESSDATE>compDateTime
       <COMPRESSEDLOCATION>SVSfilePath
       <BARCODE>barcodeValue
       <ASSAY>assayName
       <USER1>userData
       <USER2>userData
       <USER3>userData
       <USER4>userData
```

imageId	A unique number assigned when an image is created in the database
caseId	Case identifier; set from barcode value or by external application
rack#	Rack# in which the slide is/was located (0 if manual load)
slot#	Slot# within the rack in which the slide is/was located
ACIfilePath	Path to the composite image file for the slide, if any (.ACI file)
scanDateTime	Date/time at which the slide was scanned (blank = not yet scanned). The format is "yyyy-mm-dd hh:mm:ss.ttt"
ScanScopeName	Name of the ScanScope machine, typically "ss####".
scanStatus	String giving success/failure status of scanning operation
scanStatusDetails	String giving error message if *scanStatus* indicates failure
qualityFactor	Integer value 0-100 giving computed quality of image focus
operator	Operator identification, entered through console
description	Description for the slide or batch entered by operator through console
compDateTime	Date/time at which the slide was compressed (blank = not yet done). The format is "yyyy-mm-dd hh:mm:ss.ttt"
SVSfilePath	Path to the digital slide file (.SVS file)
barcodeValue	The value decoded from a slide's barcode, if any (blank if none)
assayName	The name of the assay or preparation for a slide. This value may be used to identify the algorithm required to analyze the slide data.
userData	Any string, including any XML-formatted string, which contains application data to be stored for a digital slide.

If *@imageId* or path (*SVSfilepath*) is specified on a GETDATA or PUTDATA request, it is used to unambiguously identify the digital slide in the database. (Both may be omitted using the request syntax GET @?GETDATA+d or GET @?PUTDATA+d.)

If neither is given, the following data are looked for in the XML parameter (<IMAGE>):

- If *imageId* is specified, it is used (either as an attribute of IMAGE, or a child element).
- If *SVSfilePath* is specified, it is used.
- If *barcodeValue* is specified, it is used. (Note the barcode value must be unique!)
- If *rack#* and *slot#* are specified, they are used to match the most recent digital slide with a blank *scanDateTime*. If *ScanScopeName* is also specified, it is used to match the digital slide as well, otherwise any ScanScope value may match (including blank).
- If *ACIfilePath* is specified, it is used. This is only valid if the **–aci** parameter specified.

If an image cannot be matched, an HTTP 404 (not found) is returned.

Storing and Retrieving Annotations as XML Strings

ImageServer can store and retrieve XML strings representing annotations. Each digital slide served by ImageServer can optionally have an annotation string associated with it. Annotations are authored and rendered on the client side, by rich-clients like Aperio ImageScope, or by web-based viewers such as Aperio WebScope. Annotations are stored and retrieved using the GETANNOTATIONS and PUTANNOTATIONS requests (see "ImageServer HTTP Request/Response API," on page 19 for details).

Annotations may be stored in text files alongside digital slide images, or in the [optional] image database described in the previous section, and designated by a host name and port number at startup. When a GETANNOTATIONS request is received, ImageServer first checks the image database (if present) for annotations associated with the image, and if found, returns the annotation string. If not found, ImageServer checks for a file named **<image_path>.xml**, and if found the contents are returned as the annotation string. When a PUTANNOTATIONS request is received, ImageServer stores the specified string in the location from which it was previously retrieved, if any. That is, if the digital slide has annotations in the database, the new annotations are stored in the database, and if the digital slide has annotations in a file named **<image_path>.xml**, the new annotation is stored there. In all cases the new annotation string completely replaces any previously stored annotation string[4].

Annotations are organized into one or more **layers**. Each layer consists or one or more **regions**. At user option layers may be displayed or hidden, all the regions within a layer are displayed at once. Regions are defined by a sequence of **vertices**. The coordinates of annotation vertices are always relative to the base level (highest resolution) of the digital slide. There are five **region types**, defined by vertices as explained below:

[4]Annotation layers may have a ReadOnly attribute, signifying that they made not be modified. Clients which provide facilities for authoring annotations must enforce this attribute, ImageServer does not do so.

PolyLine	Any number of vertices defines a series of connected lines. The last vertex does *not* reconnect back to the first vertex.
Polygon	Any number of vertices defines a series of connected lines and the last vertex connects back to first vertex.
Ellipse	Two vertices define opposite corners of a bounding box in which an Ellipse is drawn.
Arrow	Two vertices define the endpoints of an arrow. The first is the arrow head, and the second is the tail.
Ruler	Two vertices define the endpoints of a ruler.

Annotation layers and regions may have **attributes** associated with them. Attributes are simply named text, in the form "name=value". The interpretations given to "name" and "value" are application dependent; the annotation facility provided by ImageServer merely stores and retrieves attributes without interpreting them in any way.

Following is the XML format used to store annotations (closing tags are omitted for brevity):

```
<Annotations>
      <Annotation Id=annotationId ReadOnly=readOnly Type=type LineColor=lineColor
            Visible=visible Selected=selected MarkupImagePath=markupPath>
            <ImageId>imageId
            <ModifiedDate>modDateTime
            <Author>authorName
            <Attributes>
                  <Attribute Name=attrName Value=attrValue>
            <Regions>
                  <RegionAttributeHeaders>
                        <AttributeHeader Id=headerId Name=headerName>
                  <Region Id=regionId Type=regType Zoom=regZoom
                        Selected=regSelected ImageLocation=imagePath
                        NegativeROA=negativeROA>
                        <Attributes>
                              <Attribute Name=headerId Value=attrValue>
                        <Vertices>
                              <Vertex X=verX Y=verY>
```

annotationId	[optional] Sequence number assigned by database.
readOnly	[optional] 0=modifiable, 1=read-only (algorithm results)
type	[optional] Annotation type (1=input to algorithm, 2=(unused), 3=algorithm results, 4=manually authored).
lineColor	[optional] The color suggested for outlining the annotation regions; the 32-bit color value 00RRGGBB is formatted as a decimal value.
visible	[optional] Either 0 or 1, to indicate whether this layer is initially visible.
selected	[optional] Either 0 or 1, to indicate whether this layer is "selected." Only one annotation layer may be selected at any time. The selected layer is typically highlighted in some way when displayed.
markupPath	[optional] If the layer has an associated "markup image", this attribute gives the full disk path or ImageServer URL for the markup image.
modDateTime	The date/time of creation or last modification of the annotation layer, given as "yyyy-mm-dd hh:mm:ss.ttt"
authorName	[optional] The person who created the annotation layer, or if created by a program, the name and version of the program.
attrName	Attribute name – any valid string.
attrValue	Attribute value – any valid string.
headerId	Integer giving index to attribute header (typically used as column index)
headerName	Name of attribute header
regionId	[optional] sequence number assigned by Regions object
regType	Region type: 0=PolyLine, 1=Polygon, 2=Ellipse, 3=Arrow, 4=Ruler
regZoom	[optional] A decimal value specifying the zoom level associated with the region. 1.0 = baseline, 0.5 = 50% (zoomed out). Each region may be rendered at any zoom level, this value gives the zoom level used when the region was created.
regSelected	[optional] Either 0 or 1, to indicate whether this region is selected. Only one region may be selected within a layer at any time. The selected region is typically highlighted in some way when displayed.

imagePath	[optional] If the region has an associated markup image, this attribute gives the full disk path or ImageServer URL for the markup image.
negativeROA	[optional] Either 0 or 1, to indicate whether this region is negative. This implies the region is to be excluded from algorithm processing.
verX, verY	The X- and Y- coordinates of the vertex (0,0 at left,top)

Server-Side Algorithm Processing

ImageServer can optionally interface to server-side algorithm processing. The database used for algorithm processing communication is the same as the server-side image database (previous section), and is designated by a host name and port number at startup.

Algorithm processing is accessed via the RUNALGORITHM and ALGORITHMSTATUS requests (see "ImageServer HTTP Request/Response API," on page 19 for details). Annotation layers are used to communicate input regions of analysis to algorithms, and to communicate output result regions from algorithms. Annotations may be accessed via the GETANNOTATIONS and PUTANNOTATIONS requests (previous section, and described in "ImageServer HTTP Request/Response API").

Algorithms are referenced by **macros**. A macro is a named set of algorithms and associated parameters. Algorithm execution is initiated with the RUNALGORITHM request, which creates an entry in a queue of algorithms to be run. The queue is "worked" by one or more computers running Aperio's Algorithm Framework. In the simplest configuration, the server itself is the computer which performs algorithm processing. Along with specifying the macro to be executed, a RUNALGORITHM request may specify an annotation layer for an image to be used as the input region of analysis. If no layer is specified, the entire digital slide is processed. Annotation layers may be created and updated with the PUTANNOTATIONS request.

As algorithms run they periodically update their progress and status in the database. These data may be queried with the ALGORITHMSTATUS request to display progress on the client side. When algorithm processing is complete the results are stored as a new annotation layer for the image. Annotations may be retrieved using the GETANNOTATIONS request.

APML Template Processing

ImageServer includes a simple yet powerful template processing mechanism. Templates are text files which contain a simple superset of HTML called APML (Aperio Markup Language). APML makes use of a *dictionary*. The dictionary is populated with environment information, input parameters, and directory and image file information. When an APML template is requested, it is substituted with values from the dictionary and the resulting HTML pages are returned to the caller. All processing occurs on the server, any standard web browser may be used. Aperio's WebScope is created entirely with APML templates.

APML also supports simple directives of the form <% *directive* %> which may be embedded in the templates. Directives are provided for nesting (<% INCLUDE %>), conditional output generation (<% IF %>, <% ELSE %>, and <% END %>), and automatic formatting of tabular data (<% TABLE %>). The APML template language is documented separately; please see the *Aperio Markup Language (APML) Programmer's Reference*.

APML template processing is initiated from any standard web browser by entering URLs which have the form directory/template.apml. For example:

```
http://www.mysite.com/images/view.apml
```

In this example **www.mysite.com** is the domain name of ImageServer, and **/images** is a directory path (relative to the "base" directory for ImageServer, specified by the **–dir** command-line parameter). The dictionary will be filled with information for the **/images** directory and then the template **view.apml** is processed. The generated HTML will be the output of the request.

Here's another example:

```
http://www.mysite.com:82/images/my_digital_slide.svs/my_viewer.apml
```

In this example **www.mysite.com** is the domain name of the machine running ImageServer, and **82** is the port number[5]. The file **my_digital_slide.svs** is a digital slide in the **/my_images** directory. The dictionary will be filled with information for the **my_digital_slide.svs** image, and then the template **my_viewer.apml** is processed. The generated HTML will be returned as the output of the request.

If an image database is in use, you may reference images by *@imageId* rather than a directory file path. For example:

```
http://www.mysite.com/@4672/viewer.apml
```

In this example, the dictionary will be filled with information for the digital slide with an *imageId* of **4672**, and then the template **viewer.apml** will be processed.

APML templates are stored in a directory designated by the **–apml** command-line parameter. By default, this is <Aperio master installation folder>\ImageServer\apml. Customers may modify the templates provided as part of Aperio's WebViewer and/or may create and use their own APML templates. All files from this directory are cached by ImageServer for performance.

[5]Previous to release 6, the Aperio WebViewer required two port numbers, one for a webserver such as Microsoft IIS, and one for Aperio ImageServer. Customers often used port 82 for ImageServer. As of release 6, Aperio WebViewer uses the APML template facility of ImageServer and only one port is required.

At startup, ImageServer looks for a template named **view.apml** in the APML directory. If this template is present, ImageServer will perform *default* template processing. Requests made from a standard web browser to directories or image files will be assumed to be for the **view.apml** template. For example:

```
http://www.mysite.com/
```

In this example **www.mysite.com** is the domain name of the machine running ImageServer. This request is for the base directory. If the **view.apml** template is present, ImageServer will treat this request as a request for template processing, e.g., **http://www.mysite.com/view.apml**, and the generated HTML output will be returned[6].

As another example:

```
http://www.mysite.com/images/myimage.svs
```

In this example **www.mysite.com** is the domain name of ImageServer and the request is for a file named **myimage.svs** in the **/images** directory. If the **view.apml** template exists, ImageServer will assume this is a request for default template processing, e.g., http://www.mysite.com/images/myimage.svs/view.apml, and the generated HTML output will be returned[7]. Note that the default template **view.apml** can determine whether it was invoked for a directory or an image file by testing the "isdir" value in the dictionary.

Web applications created using APML frequently make use of static HTML pages, images, Flash applets, and other such files. Such files may be referenced with URLs that have a "/html/…" directory prefix. For example:

```
http://www.mysite.com/html/my_file.html
```

In this example, **www.mysite.com** is the domain name of ImageServer, and the file **myfile.html** will be returned "as is". The directory prefix "/html/…" indicates that this file is to be accessed from the HTML directory. All files from this directory are cached by ImageServer for performance.

These files may be stored in a directory designated by the **–html** command-line parameter. By default this is **C:\Program Files\Aperio\ImageServer\html**. Customers may use and modify the files provided as part of Aperio's WebViewer and/or may create and use their own HTML files, images, etc.

> Note: All APML templates and HTML files are cached by ImageServer for performance. To clear the cache, use the RESET command (e.g., while creating and testing templates).

[6]The DIR or DIR3 parameters may be supplied to force data generation, e.g., http://www.mysite.com/?DIR3.

[7]This default template processing only occurs if no parameters are supplied. Any data requests for an image file typically have parameters specified.

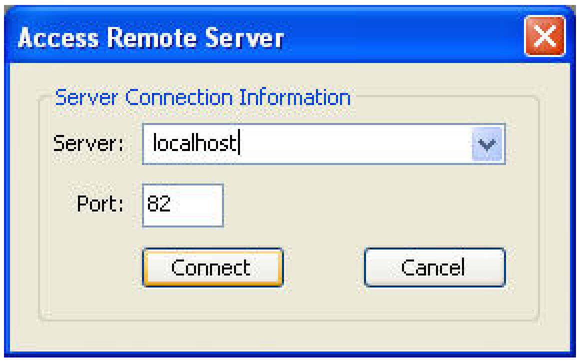
The power and usefulness of APML templates comes from the information which is available in the dictionaries. The contents of the dictionaries used for APML template processing are described in a separate section below on page 40.

Testing and Using ImageServer

After installation of ImageServer it may be tested using the Aperio ImageScope program. This program is provided as a self-installing executable along with ImageServer. ImageScope may be freely distributed and may be used locally (within the local network) or remotely (over a wide-area network).

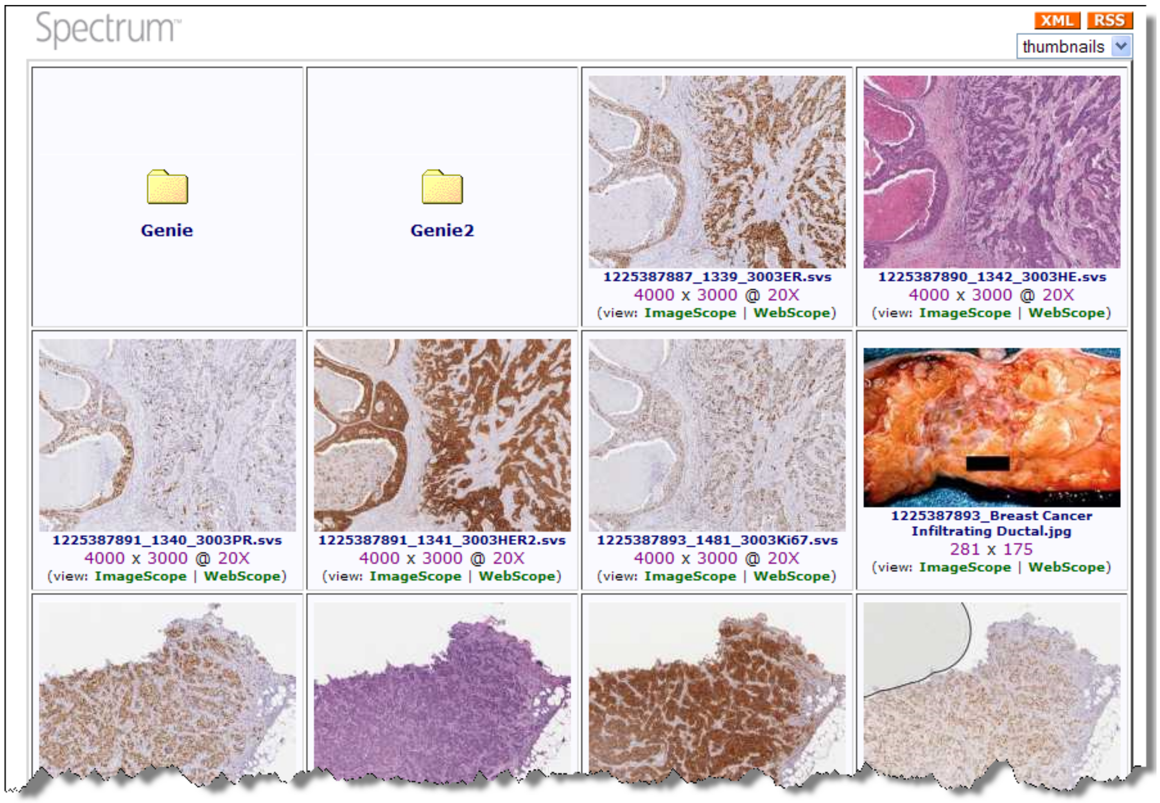
After installing ImageScope, it may be run by selecting Start | Programs | Aperio | ImageScope. Once ImageServer is running, connect to it by selecting File | Open Remote File.

To test ImageServer, enter "localhost" into the Server field and "82" into the Port field, and click **Connect**.



Server: localhost
Port: 82

If ImageServer is running successfully, you will see a page of images like this:


Spectrum
XML
RSS
thumbnails
Genie
Genie2
1225387887_1339_3003ER.svs
4000 x 3000 @ 20X
(view: ImageScope | WebScope)
1225387890_1342_3003HE.svs
4000 x 3000 @ 20X
(view: ImageScope | WebScope)
1225387891_1340_3003PR.svs
4000 x 3000 @ 20X
(view: ImageScope | WebScope)
1225387891_1341_3003HER2.svs
4000 x 3000 @ 20X
(view: ImageScope | WebScope)
1225387893_1481_3003Ki67.svs
4000 x 3000 @ 20X
(view: ImageScope | WebScope)
1225387893_Breast Cancer Infiltrating Ductal.jpg
281 x 175
(view: ImageScope | WebScope)

Clicking **view: ImageScope** will allow you to view the image using ImageScope via the ImageServer.

Other users may connect to ImageServer by specifying your machine's network name in the Server field. It is possible to change the **port** used by ImageServer from the default value of 82, please see "Configuring ImageServer," below.

It is also possible to identify servers by IP address rather than by network name. Simply type the IP address of the server into the Server field, using "dotted" notation (e.g., "192.168.1.1").

Configuring ImageServer

As installed, ImageServer uses default values for a number of operational parameters. These parameters are supplied from a file named:

<Aperio master installation folder>/ImageServer/ImageServer.ini

This is a text file with one line which may contain one or more of the following parameters:

Parameter	Arguments	Description
-ip	<ip_address>	hostname or IP address (default localhost)

Parameter	Arguments	Description
-ipmask	<ip_mask>	IP address mask (see below)
-prt	<port>	port number (default = 80)
-dir	<base_dir>	base directory for files
-apml	<apml_dir>	directory which contains APML templates
-html	<html_dir>	directory with files for "/html/..." prefix
-authip	<host_address>	authorization service hostname / IP address
-authprt	<port>	authorization service port number
-dbip	<host_address>	database service hostname / IP address
-dbprt	<port>	database service port number
-dburi	<uri>	database service URI
-label		enable label images
-mactrim	<left%> <right%>	crop macro images by left% and right%
-aci		support .ACI files, hide TIFF files
-cws		support .CWS files (directories)
-ovly		include overlay images in dir listing
-uttl		enable TITLE updates for dirs / images
-updt		enable RENAME, MOVE and DELETE requests for images
-udb		enable PUTDATA requests for images
-uannot		enable PUTANNOTATIONS requests for images
-ualgo		enable RUNALGORITHM requests for images
-tmot	<req_timeout>	request timeout (in sec, default = 60)
-optmot		image open timeout specified in seconds (suggested value: 5) See "ImageServer Open Image Timeout Parameters" below.
-optmimg	<default_image_URI>	URI for default timeout message image (default: /html/default.jpg). See "ImageServer Open Image Timeout Parameters" below.
-cache	<cache_limit>	cache size limit (MB, default = dynamic)
-log	<log_path>	log file path
-logsz	<max_log_size>	maximum size of logs (MB, default = 1)
-lognm	<max_nbr_logs>	maximum number of logs (def = 5)
-v(1/2)		verbose (enable tracing), v1=more, v2=data dump
-icm		enable server-side Integrated Color Management
-education		prevents ImageServer from serving image data

The **-ip** parameter gives the address on which the server "listens" for connections. As installed, this is specified as the network name of the machine. The **–ipmask** parameter may be used to specify the particular IP address of the machine to be used, if there is more than one (more than one network interface, and/or more than one IP address assigned to an interface). The mask parameter may contain wildcards (* and/or ?), and may begin with a ~ to indicate the first address which does *not* match the mask should be used. (This is useful when one or more of the IP addresses on a machine are assigned dynamically via DHCP.) The **-prt** parameter gives the port on which the server "listens" for connections. As installed, this is specified as 82. For some network configurations, using port 80 may be more convenient, as it is frequently "open" in network firewalls to enable webserver traffic.

The **–dir** parameter specifies the path to the "root" directory for image serving. As installed, this is specified as the C:\images directory. This may be changed to any drive or directory on the machine, including network shares[8]. The **–apml** parameter specifies the path to a directory containing APML templates, by default <Aperio master installation folder>\ImageServer\apml. Any APML templates referenced are loaded from this directory. The **–html** parameter specifies the path to a directory of HTML files, by default <Aperio master installation folder>\ImageServer\html. Any files referenced with a "/html/…" prefix are loaded from this directory. This directory also contains Flash applets, help files, and auxiliary images.

The **–authip** and **–authprt** parameters specify the hostname / IP address and port for the database authorization service, if any. If these parameters are omitted, no authorization checking is performed.

The **–dbip** and **–dbprt** parameters specify the hostname / IP address and port for the database interface service, if any. If these parameters are omitted, image metadata is not available. The **-dburi** parameter specifies the database service URI; if omitted, the default value /Aperio.Images/Image.asmx is used. If a database service is present, it is used to get and retrieve annotations; otherwise, annotations are assumed to be stored in XML files associated with each digital slide.

The **–label** parameter enables label images to be retrieved. In clinical environments, label images may contain patient names, case numbers, identifying barcodes, and other sensitive information, so consider carefully whether to use this parameter.

The **–mactrim** parameter enables macro images to be cropped at the left and right by specified amounts. This is useful in case portions of the slide label or other operational data are visible in macro images. By trimming the image, no personally identifiable information is revealed. This is typically used in situations where the **–label** parameter is not supplied (to suppress label images).

The **–aci** parameter enables support for ACI files (Aperio Composite Images). These are text files which contain a list of other TIFF files to be "composited" into one digital image (please see page 45 for details). When this parameter is present, TIFF files are hidden in directory views.

The **–ovly** parameter enables overlay images to be displayed in directory listings (these are typically the "markup image" output from server-side algorithm processing, or they may be saved z-stacks from 3D revisit processing). By default, these images are *not* displayed in directories, since they are usually not opened directly.

[8]Using a network share to host images can have performance implications.

The **–uttl** parameter enables TITLE updates for image files and directories. The **–updt** parameter enables RENAME, MOVE and DELETE requests for images (image files may be physically renamed or deleted from disk). The **–udb** parameter enables PUTDATA requests for images (image database information may be updated). The **–uannot** parameter enables PUTANNOTATIONS requests for images (image annotation information may be updated). The **–ualgo** parameter enables RUNALGORITHM requests for images (which in turn update the database and create annotation information).

The **–tmot** parameter gives the time in seconds that connections are left open. HTTP 1.1 supports "pipelining" of requests to optimize performance; this parameter determines how much time may elapse on a connection which is left open before it is closed automatically. The default is 120 seconds.

The **–cache** parameter gives an upper limit to the amount of memory in MB which will be used for caching image and directory data. Image caching is important to reduce disk contention when multiple clients are accessing the same images. The default is 0, which means the limit is determined by the amount of memory in the machine. The actual cache size is adjusted continuously to match the available physical memory of the machine.

The **–log**, **-logsz**, and **–lognm** parameters control the logs created by ImageServer. Log files are named imageserver.<seq#>.log, and are stored in the directory given by the **–log** parameter; the default location is the directory from which ImageServer is run. The **–logsz** parameter gives the maximum size of a log in MB (when this value is reached, a new log is created) and the **–lognm** parameter gives the number of logs which are kept (default is 5).

The **–v** parameter causes tracing information to be written into the log files. Specifying **–v1** causes more tracing to be performed, and **–v2** causes data dumps of message headers.

The **–icm** parameter enables server-side Integrated Color Management to enhance images to be returned from getView() calls. ICM is only performed if the scanner profile has been embedded in the image file.

The **–education** parameter prevents ImageServer from serving image data. It will still serve non-image data related to images, such as annotations.

ImageServer Image Open Timeout Parameters

The **–optmot** and **–optmimg** parameters control image open timeout behavior.

When customers store images on a "slow" storage device, such as a tape drive, the first access to an image may take a considerable amount of time (seconds or even minutes), while subsequent accesses to the image are fast after the file has been loaded.

During the first access to the image in Spectrum, the web browser may time out or the Spectrum user may think the application has hung.

The ImageServer optional timeout feature improves the user experience in this situation. When it is enabled, the first time the user access an image, if the open times out, the user sees a default image which typically contains a text message such as "Your image is being loaded, please retry in X seconds." (This image may be created by the Aperio customer to tailor it to their organization.)

When the user retries, the image will have been loaded and the open will complete successfully. If the user retries too soon, he or she will see the please retry message again.

This feature is disabled by default, and is enabled by supplying the **–optmot** and **–optmimg** parameters as follows:

```
-optmot <open_timeout>
-optmimg <default_image_URI>
```

Where -**optmot** specifies the image open timeout interval in seconds (suggested value: 5), and -**optmimg** is an optional parameter that specifies a URI for the default retry-open image (default: /html/default.jpg).

The open timeout feature is only supported when ImageServer is running in multithreaded mode. In multithreaded mode, every image request is processed under a connection thread. When the open timeout is specified, each time an image is opened a timeout thread is started. If the image open completes successfully within the timeout interval, the timeout thread is cancelled and the image request is completed normally under the connection thread. If the image open does not complete successfully within the timeout interval, the timeout thread cancels the connection thread which is attempting to open the image. It reconfigures itself as the connection thread, and then completes the image request using the open timeout default image. This is usually a small image which contains a text message for the user, such as "your image is being loaded, please retry in X seconds."

ImageServer HTTP Request/Response API

All requests made to ImageServer are via HTTP over TCP/IP. Standard GET / POST syntax for various URLs is used; a standard HTTP response message is returned.

ImageServer supports the following HTTP requests, which are described on the following pages (the page number on which each request is documented is indicated):

`GET directory`	- returns directory contents	23
`GET directory?DIR3`	- returns directory contents (with 3D info)	23
`GET directory?INFO`	- returns directory information	23
`GET directory?TITLE+t`	- sets / clears directory title	24
`GET file`	- returns file contents	24

For all requests that support specification of an *image*, the image may be identified in the request URI by a file path, or [if an image database is present] by an *imageId*, as "@id". If specified by @id, the database is queried for the specified image, and if found, the corresponding file path is used to access the image.

If desired or required for data length reasons[9], any supported request may be made via an HTTP POST instead of a GET. In this case, the parameters for the URL, if any, are passed in the body of the POST request, rather than being appended to the URL (and delimited with a ? character). For example:

```
POST image HTTP/1.1
Content-Length: n

PUTANNOTATIONS+d
```

The Content-Length HTTP header is required on POST requests; other HTTP headers (such as Content-Type or User-Agent) are allowed but ignored. Note that unlike data passed as a URL parameter on a GET request, data passed in the request body for a POST *need not* be URL-encoded, although it may be.

ImageServer supports HTTP 1.1 "pipelining" (multiple request/response transactions within one connection). The HTTP Connection: header is used to determine whether connections are automatically closed after a response is sent. Open connections are timed out after 120 seconds.

The following HTTP response types may be returned:

200	- request successful, data returned per response type
400	- invalid request, text string returned with error message
401	- authorization required (see "ImageServer Authorization")
403	- forbidden request (disabled functionality), text string returned with message
404	- directory/file/data not found, text string returned with message
500	- server error, text string returned with error message

ImageServer never returns an HTTP 100 response ("Continue").

[9]ImageServer has a limit of 10K bytes in request URLs, and many HTTP proxies and gateways also have limits on the size of URLs in HTTP headers. For this reason, POST is recommended for all PUTDATA and PUTANNOTATIONS requests.

GET directory, GET directory?DIR3

`GET`	`directory`	- returns directory contents
`GET`	`directory?DIR(+I)(+C)`	- returns directory contents (preferred)
`GET`	`directory?DIR3(+I)(+C)`	- returns directory contents (with 3D info)
`GET`	`directory?DIR(+A)`	- returns directory contents (preferred)
`GET`	`directory?DIR3(+A)`	- returns directory contents (with 3D info)

Returns directory contents for the specified **directory** path. Contents are returned as a text file where each line has the form:

```
file|dir|len|||||title
file|file|len|width|height(|depth)|twidth|theight
      (|TML)(|comptype|compqual)|title|desc
file|type|len
```

The first form is returned for directories, the second for digital slide files (.SVS, .TIFF, etc.), and the third for all other directory entries. Field values:

file	- name of file within directory
type	- the type of file, either dir, link, or file
len	- length of file in bytes
width	- width of image in pixels
height	- height of image in pixels
depth	- depth of image in pixels (only returned on DIR3 requests)
twidth	- tile width in pixels (0 => not tiled)
theight	- tile height in pixels (0 => not tiled)
TML	- TML are each 0/1, whether image has thumbnail, macro, and label (only returned on DIR(3)+I requests)
comptype	- compression/file type (only returned on +C requests)
compqual	- compression quality (only returned on +C requests)
bigtiff	- 0/1 – whether file is BigTIFF
title	- file title, if any
desc	- file description (from TIFF header), if any

The +A parameter causes all file names to be returned, even those that are not in the database, but no other information is returned.
The +I and +C parameters are ignored if +A is specified.

GET directory?INFO

`GET`	`directory?INFO`	- returns directory information

Returns directory information for the specified **directory** path. A single line is returned of the form:

```
||||title
```

where **title** is the directory title, if any.

GET directory?TITLE+t

```
GET   directory?TITLE+t
```
- sets / clears directory title

The directory's title may be set or cleared. The text passed, if any, becomes the title for the directory. Characters in the title must be URL-encoded (e.g. space = %20).

This request is only supported if the **–uttl** parameter is supplied at startup.

GET file

```
GET   file
```
- return file contents

Returns file contents for the specified file path. If the file is a text or HTML file, or a JPEG, GIF, or TIFF image, the file contents are returned "as is" with the appropriate MIME type set. XML files (.xml) and text files (.txt) are always ignored.

If the **–aci** parameter is *not* specified at startup, composite image files are ignored (.aci). If the **-aci** parameter is specified at startup, serving of digital slides from composite image files is supported, and TIFF files (.tif or .tiff) will be ignored.

GET image?x+y+w+h+z+q+m+f+r

```
GET   image?x+y+w+h+z+q+m+f+r
```
- returns region from image file

Returns region from an Aperio image file (SVS) at the specified **image** path. The region is returned as a JPEG image. The following parameters may be supplied:

x	**X-coordinate** of region, in pixels (upper left is [0,0]). Default is 0.
y	**Y-coordinate** of region, in pixels (upper left is [0,0]). Default is 0.
w	**width** of region, in pixels. Maximum width is 2000 pixels. Default is image width or 2000, whichever is less.
h	**height** of region, in pixels. Maximum height is 2000 pixels. Default is image height or 2000, whichever is less.
z	**zoom** level of region (float), .5 = 200%, 2 = 50%, 4 = 25%. Other special values: -1 = thumbnail image -2 = label image (if any) -3 = macro camera image (if any) The default value for zoom is 1 (= 100%)

q	JPEG **quality** level, (1-100). A level of 75 approximately corresponds to the JPEG2000 image quality. Default is 75.
m	JPEG **mode**. If specified as **S** images will be static (non-progressive). Default is progressive (**P**). {Ordinarily progressive mode is "better" however some software cannot handle progressive JPEGs, notably Macromedia Flash.}
f	**focus** value for region (float), in range -1 -> +1 (with 0 at "center"). Default is 0.
r	**random** string value. Not used by ImageServer, but can be used to ensure that the URL is unique. This is useful for defeating browser caching.

If **x** and **y** are specified with a leading 0, *they are relative to the image base*, else they are relative to the zoomed image. Note that smaller zoom values result in higher resolution.

If **focus** is specified, it is in the range -1 to +1, with 0 at "center" of image. Currently focusing is only supported along the Z-axis. Focus values only affect 3D images.

An image thumbnail may be requested by specifying a zoom level of -1. For this case, the **x** and **y** are ignored, and the **w** and **h** give the dimensions of the returned thumbnail (which is scaled appropriately). A width or height value of zero causes the image aspect ratio to be preserved; if both are zero, the "native" size of the thumbnail is used.

The label image associated with the image file may be requested by specifying a zoom of -2. This is enabled by the **–label** parameter. For this case, the **x** and **y** are ignored, and the **w** and **h** give the dimensions of the returned label image (which is scaled appropriately). A width or height value of zero causes the label image aspect ratio to be preserved; if both are zero, the "native" size of the label image is used.

The macro camera image associated with the image file may be requested with a zoom of -3. For this case, the **x** and **y** are ignored, and the **w** and **h** give the dimensions of the returned macro camera image (scaled appropriately). A width or height value of zero causes the macro camera image aspect ratio to be preserved; if both are zero, the "native" size of the macro camera image is used. If the **mactrim** parameter is given, the macro image is cropped by the specified%.

GET image?BLOCK_+x+y+p+l

`GET    image?BLOCK+x+y+p+l` - returns block from image file

Returns the specified block from the image file. The following parameters may be supplied:

x	X-coordinate of block. Default is 0.
y	Y-coordinate of block. Default is 0.
p	image pyramid level. Default is 0 (baseline).
l	image layer. Default is 0 (closest).

The block is returned "as is" without any processing including no color management transformation. The requesting client is responsible for decoding the block data according to the type of compression employed. The FINFO request returns the compression type and codec used for compressing the file. The CINFO request may be required to obtain compression data used for decoding blocks.

If specified the pyramid level designates the image from which the block is retrieved. Pyramid information for an image may be retrieved with the PINFO request. Default is 0 (baseline).

If specified, the image layer designates the Z-layer from which the block is retrieved. Layer information for an image may be retrieved with the LINFO request. Default is 0.

GET image?INFO, GET image?INFO3

`GET`	`image?INFO(+I)`	- returns file information
`GET`	`image?INFO3(+I)`	- returns file information (with 3D info)

Returns image file information for the specified **image** as a single line with the format:

```
width|height(|depth)|twidth|theight(|TML)|title|desc
```

Where the fields mean:

width	- width of image in pixels
height	- height of image in pixels
depth	- depth of image in pixels (only returned for INFO3 request)
twidth	- tile width in pixels (0 => not tiled)
theight	- tile height in pixels (0 => not tiled)
TML	- TML are each 0/1, whether image has thumbnail, macro, and label (only returned on INFO(3)+I requests)
title	- file title, if any
desc	- file description (from TIFF header), if any

INFO(3) is supported for backward compatibility; new applications should use XINFO instead.

GET image?FINFO

`GET`	`image?FINFO`	- returns image attributes

Returns image file information for the specified **image**, as a single line with the format:

```
size|comptype|compqual|compcodec
```

Where the fields mean:

size	- file size of image (after compression)
comptype	- compression type, specified as 0=none, 1=LZW, 2=JPEG, 3=JPEG2000, 7=YUYV, 8=JPEG (.jpg), 10=CWS

compqual	- compression quality (1-100) for JPEG/JPEG2000
compcodec	- description of codec used for compression

FINFO is supported for backward compatibility; new applications should use XINFO instead.

GET image?LINFO

```
GET   image?LINFO
```
- returns image file layer information

Returns image file layer information for the specified **image**, as a single line with the format:

```
layers|(lzoff|ldepth|lfocus)
```

Where **layers** is the number of Z-layers in the file. The layer parameters are repeated for each layer present in the file, and have the meaning:

lzoff	- z-offset to layer, in pixels (from "closest")
ldepth	- depth of layer, in pixels
lfocus	- focus value for layer (center of depth), in range -1 -> +1

If an image is purely 2D, it will have one layer at z-offset 0 of depth 1 with focus value 0.

LINFO is supported for backward compatibility; new applications should use XINFO instead.

GET image?PINFO+l

```
GET   image?PINFO+l
```
- returns image file pyramid information

Returns image file pyramid information for the specified **image** (and for the specified layer **l**, if any), as a single line with the format:

```
levels|(lwidth|lheight|lzoom)
```

Where **levels** is the number of pyramid levels in the file/layer. (If the layer is omitted, 0 is assumed.) Level data are returned for each level present in the file/layer, including the thumbnail (as the last level for each layer), and have the meaning:

lwidth	- width of pyramid level image, in pixels
lheight	- height of pyramid level image, in pixels
lzoom	- zoom level of pyramid image, 2 => 50%, 5 => 20%

If the image file has a label image (and the **–label** parameter was supplied), following the thumbnail will be dimensions for the label image (for layer 0). This can be determined from the zoom for this image which will be 1.0.

If the image file has a macro camera image, following the thumbnail or label will be dimensions for the macro camera image (for layer 0). The zoom level for this image will be 1.01.

PINFO is supported for backward compatibility; new applications should use XINFO instead.

GET image?CINFO+p+l

`GET    image?CINFO+p+l` - returns image file compression data

Returns image file compression data from the specified **image**, for the specified pyramid level (if omitted, 0 = baseline is used), and Z-layer (if omitted, 0 is used). These data are passed back "as is" for use in decoding block information when images are accessed at the block level (see BLOCK parameter).

An example is images which are compressed with JPEG compression. JPEG uses large quantization tables which are identical for each block, and which are stored separate from the block data themselves. Other compression technologies may also have level-dependent data which are common to all blocks in the level. JPEG2000 *does not* make use of such tables.

CINFO is supported for backward compatibility; new applications should use XINFO instead.

GET image?XINFO+d

`GET    image?XINFO+data` - returns image data as XML block

Returns a variety of image data for the specified **image**, based on the [optional] data parameter. The **data** parameter may have the following nodes:

`<commonlevels>`	- specifies that layers are reported within levels (if omitted, levels are reported within layers)
`<getthumb>`	- thumbnail image data will be returned (base64-encoded)
`<getmacro>`	- macro image data will be returned (base64-encoded)
`<getlabel>`	- label image data will be returned (base64-encoded)
`<getprofile>`	- ICC profile will be returned (base64-encoded)

The data parameter and the returned response data are XML-formatted, with an `<XINFO>` node as the outer XML tag. The following data elements are returned:

`<width>`	- image width in pixels
`<height>`	- image height in pixels
`<depth>`	- image depth in pixels
`<bitsPerSampl>`	- number of bits per sample
`<samplPerPix>`	- number of samples per pixel
`<tilewidth>`	- image tile width in pixels (0 => not tiled)
`<tileheight>`	- image tile height in pixels (0 => not tiled)
`<title>`	- file title, if any
`<description>`	- file description string (from TIFF header), if any

```
<filesize>        - file size in bytes (compressed)
<imagesize>       - image size in bytes (uncompressed)
<comptype>        - compression type (0=none, 1=LZW, 2=JPEG, 3=JPEG2000)
<compquality>     - compression quality (1-100) for JPEG/JPEG2000
<compcodec>       - compression codec description
<layers>          - number of Z-layers in file
<layer id=n>      - node describing layer n, one for each layer
      <zoff>         - layer Z-offset, in pixels
      <depth>        - depth of layer, in pixels
      <focus>        - [optional, float] focus value for layer, in range -1 -> +1
      <levels>       - number of levels for layer (including base, but not thumbnail)
      <level id=n>   - node describing level n, one for each level (not thumbnail)
            <width>      - width of image, in pixels
            <height>     - height of image, in pixels
            <zoom>       - [float] zoom level of image, 2 => 50%, 5 => 20%
            <xoff>       - [optional] x-offset to data in this level
            <yoff>       - [optional] y-offset to data in this level
            <tilewidth>  - [optional] tile width for this level (default = main)
            <tileheight> - [optional] tile height for this level (default = main)
            <cdata>      - [optional] compression data for image, base64-encoded
<thumb>           - present => image has thumbnail
      <width>        - width of thumbnail, in pixels
      <height>       - height of thumbnail, in pixels
      <zoom>         - [float] zoom level of thumbnail, 2 => 50%, 5 => 20%
      <image>        - [optional] thumbnail image, JPEG-compressed, base64-encoded
                       (returned if <getthumb [quality=q] [crypt=e]> specified)
<macro>           - present => image has macro image
      <width>        - width of macro image, in pixels
      <height>       - height of macro image, in pixels
      <image>        - [optional] macro image, JPEG-compressed, base64-encoded
                       (returned if <getmacro [quality=q] [crypt=e]> specified)
<label>           - present => image has label image
      <width>        - width of thumbnail, in pixels
      <height>       - height of thumbnail, in pixels
      <image>        - [optional] label image, JPEG-compressed, base64-encoded
                       (returned if <getlabel [quality=q] [crypt=e]> specified)
<profile>         - present => image has embedded icc profile
      <length>       - byte count of profile
      <data>         - [optional] profile byte stream, base64-encoded
                       (returned only if <getprofile>  specified)
```

For the <thumb>, <macro>, and <label> data, which have an optional <image> sub-node, the data parameter determines whether the image data are returned, and the quality= attribute (if present) determines the JPEG compression quality. The crypt= attribute (if present) determines the encryption applied to the image; currently the only valid value is "IDEA" (specifying 128-bit IDEA encryption). If the <image> data are encrypted, a crypt= attribute is returned on the response node.

XINFO is recommended for obtaining file information instead of INFO, FINFO, LINFO, PINFO, and CINFO.

GET image?PROFILE

```
GET   image?PROFILE
```
- returns ICC profile embedded in the image

Returns the base64-encoded ICC profile of the scanner that captured the image.

GET image?PRESCAN

```
GET   image?PRESCAN
```
- returns prescan data for image file

Returns the prescan data for the specified **image**, if available. The "prescan" is a scan performed on a small area of clear glass which enables variations in pixel sensors, illumination, optical rolloff, and glass opacity to be subtracted out.

The data are returned as a series of text lines with the form:

```
pixel#:redMult,greenMult,blueMult
```

where:	pixel#	is the index of the pixel location across the scan (0-based)
	redMult	is the "multiplier" for red channel values at this pixel location
	greenMult	is the "multiplier" for green channel values at this pixel location
	blueMult	is the "multiplier" for blue channel values at this pixel location

The multiplier values are expressed as "times 1000" to enable prescan correction to be performed with fixed-point arithmetic. Each pixel in the image is corrected as follows:

color_value = sensor_value * multiplier / 1000

GET image?TITLE+t

```
GET   image?TITLE+t
```
- sets / clears image title

The image's title may be set or cleared. The text passed, if any, becomes the title for the image. Characters in the title must be URL-encoded (e.g. space = %20).

This request is only supported if the **–uttl** parameter is supplied at startup.

GET image?COPY+newpath

```
GET    image?COPY+newpath
```
- copy image file

This request causes the specified **image** to be copied to the path specified by newpath. Newpath Specifies the destination folder, not the destination file name. If newpath begins with a slash, then the path is relative to the ImageServer base folder. If newpath does not begin with a slash, then the path is relative to the path containing the source image.
The copy request returns immediately and the copy operation is continued asynchronously. The progress of the copy operation can be queried by a TRANSFERPROGRESS request.

The request returns an index that can be used with TRANSFERPROGRESS to get the percent complete of the copy.

GET image?TRANSFERPROGRESS+transferindex

```
GET    image?TRANSFERPROGRES+transferindex
```
- get image file transfer progress

This request returns the percentage complete (0 – 100) of a previously initiated copy or move request. Once a TRANSFERPROGRESS request returns 100, indicating the copy or move has completed, further TRANSFERPROGRESS requests for the same image and transferindex will return an error indicating that the file is not being copied or moved.

The method will return an error string if an error occurred during the copy or move transfer.

GET image?RENAME+n

```
GET    image?RENAME+n
```
- rename image file(s)

This request causes the specified **image** to be renamed. If the image is in digital slide form, the SVS or TIFF file is renamed. If the image is in composite image form (ACI file), the ACI file is renamed. All associated files are renamed correspondingly: title files, annotations, macro image file, label image file, thumbnail image file, etc. If the image is a composite webslide (CWS file), the CWS directory is renamed. Note this function only supports *renaming*, not *moving*, so the new path must be relative or on the same device as the old path.

This request is only supported if the **–updt** parameter is supplied at startup.

GET image?MOVE+newpath

`GET    image?MOVE+newpath` - move image file(s)

This request causes the specified **image** to be moved to the folder specified by newpath. If the image is in digital slide form, the SVS or TIFF file is moved. If the image is in composite image form (ACI file), the ACI file is moved. All associated files are renamed correspondingly: title files, annotations, macro image file, label image file, thumbnail image file, etc. If the image is a composite webslide (CWS file), the CWS directory is moved.

The move request returns immediately and the move operation is continued asynchronously. The progress of the move operation can be queried by a TRANSFERPROGRESS request.

The MOVE request returns an index that can be used with TRANSFERPROGRESS to get the percent complete of the move.

This request is only supported if the **–updt** parameter is supplied at startup.

GET image?DELETE

`GET    image?DELETE+NOEXIST` - delete image file(s)

This request causes the specified **image** *to be deleted*. If the image is in digital slide form, the SVS or TIFF file is removed. If the image is in composite image form (ACI file), the ACI file and all associated TIFF files are removed. All associated files are deleted as well: title files, annotations, macro image file, label image file, thumbnail image file, etc. If the image is a composite webslide (CWS file), the CWS directory and all files in the directory are deleted.

If the +NOEXIST parameter is specified, then ImageServer will return a success code (200) even if the file does not exist. If the +NOEXIST parameter is not specified and an attempt is made to delete a non-existent file, ImageServer will return an error.

This request is only supported if the **–updt** parameter is supplied at startup.

GET image?GETID

`GET    image?GETID` - returns imageId for image

This request causes the *imageId* associated with the image (typically specified as URL) to be returned, as plain text. This request requires an image database be present (**-dbip** parameter).

GET image?GETURI

```
GET    image?GETURI
```
- returns URI for image

This request causes the URI associated with the image (typically specified as *@imageId*) to be returned, as plain text. Resolution of *@imageId* requires an image database be present (**-dbip** parameter).

GET image?GETPATH

```
GET    image?GETPATH
```
- returns file path for image

This request causes the file path associated with the image to be returned, as plain text. Typically this is not useful to requesting programs, since the image is probably not addressable via this path; the GETURI request (above) returns the URI which *is* addressable. Resolution of *@imageId* requests an image database be present (**-dbip** parameter).

GET image?GETDATA+d

```
GET    image?GETDATA+d
```
- returns image data as XML string

This request causes the image metadata associated with the specified **image** to be returned (if any), formatted as an XML string. The XML data are retrieved from the image database. The optional *d* parameter may be used to supply an XML string with data identifying the image to be updated. Please see the section "Server-Side Image Database" on page 7 for more information and the XML string format. Note that characters in the input string (if given) must be URL-encoded (e.g. space = %20, left-bracket = %3C, equal-sign = %3D, right-bracket = %3E).

GET image?PUTDATA+d

```
GET    image?PUTDATA+d
```
- stores image data passed as XML string

This request causes the image metadata for the specified **image** to be updated. The *d* parameter is an XML string with the data to be updated; please see the section "Server-Side Image Database" on page 7 for more information and the XML string format. If there are already data in the database for the image, new values supplied on this request update existing values, but otherwise existing data are preserved. The image metadata are stored in the image database. Note that characters in the string must be URL-encoded (e.g. space = %20, left-bracket = %3C, equal-sign = %3D, right-bracket = %3E).

In many cases, the image data XML string can be quite large, so using the HTTP POST mechanism for PUTDATA requests is recommended.

This request is only supported if the **–udb** parameter is supplied at startup.

GET image?GETANNOTATIONS

`GET    image?GETANNOTATIONS` - returns annotations as XML string

This request causes the annotations for the specified **image** to be returned (if any), formatted as an XML string. The XML data are retrieved from the image database or from a text file. Please see the section "Storing and Retrieving Annotations as XML Strings" on page 9 for more information and the annotation XML string format.

GET image?GETANNOTATIONS2

`GET    image?GETANNOTATIONS2` - returns annotations as XML string

This request is identical to the GETANNOTATIONS request with the exception that the region attribute headers 'Region', 'Length', 'Area' and 'Text' return the real database id's for these attributes and not the pseudo-ids of 9999, 9998, 9997 and 9996. This request is valid only with DataServer version 11.0 and greater.

GET image?PUTANNOTATIONS+a

`GET    image?PUTANNOTATIONS+a` - stores annotations passed as XML string

This request causes the annotations for the specified **image** to be added or replaced. All annotation layers for the image are set or updated by this request. Annotations are stored in the image database (if any), or in a text file named **<image>.xml**, where <image> is the path to the digital slide file. Please see the section "Storing and Retrieving Annotations as XML Strings" on page 9 for more information and the annotation XML string format. Note that characters in the string must be URL-encoded (e.g. space = %20, left-bracket = %3C, equal-sign = %3D, right-bracket = %3E).

In many cases, the annotations XML string can be quite large, so using the HTTP POST mechanism for PUTANNOTATIONS requests is recommended.

This request is only supported if the **–uannot** parameter is supplied at startup.

GET image?GETVIEWSET

`GET    image?GETVIEWSET` - returns view settings as XML string

This request causes the view settings for the specified **image** to be returned (if any), formatted as an XML string. The XML data are retrieved from the image database or from a text file.

GET image?PUTVIEWSET+a

`GET    image?PUTVIEWSET+a` - stores view settings passed as XML string

This request causes the view settings for the specified **image** to be added or replaced. View settings are stored in the image database (if any), or in a text file named **<image>_adj.xml**, where <image> is the path to the digital slide file.

GET image?GETNEXTANNOTATIONID

`GET    image?GETNEXTANNOTATIONID` - gets next annotation ID for image

This request causes a request to be made which returns the next available annotation Id for the image. This will be the annotation Id to be used for the next layer created.

GET image?RUNALGORITHM+m+l+g+x+o+p

`GET    image?RUNALGORITHM+m+l+g+x+o+p` - requests algorithm processing for image

This request causes a request to be made to run an algorithm for the specified **image**. The [required] "m" parameter gives the name of the algorithm "macro" to be used (macros specify one or more algorithms to be run serially, along with default parameter values). The [optional] **l** parameter specifies the annotation layer to be used as the input region of analysis by the algorithms; this is a 1-based value, if zero or omitted the entire digital slide is used as the region of analysis. Also, if the **l** parameter is preceeded by an '@', then the string following it refers to an annotation identifier. The [optional] **g** parameter determines whether the server should generate a markup image. The default value is 0, and a value of 1 requests a markup image. If successful, the response is the algorithm queue job identifier. Please see the section "Server-Side Algorithm Processing" on page 12 for more information. The [optional] **x** parameter is an XML string that is the macro to execute. This allows for the execution of macros that have not yet been stored in the database. The [optional] **o** parameter specifies the output annotation layer to be used by the algorithms. This parameter is used when the algorithm is using incremental processing. The default value is 0, which means the algorithm will create a new output layer. The [optional] **p** parameter specifies whether algorithms capable of doing preprocessing should do so. The default value is 1, which means that preprocessing should be performed by the analysis algorithm.

This request is only supported if the **–ualgo** parameter is supplied at startup.

GET image?ALGORITHMSTATUS+j

`GET    image?ALGORITHMSTATUS+j` - returns algorithm processing status for image

This request retrieves status from algorithm processing of the specified **image**. The [required] **j** parameter gives the job queue identifier, as returned in response to a RUNALGORITHM request. Please see the section "Server-Side Algorithm Processing" on page 12 for more information.

The status is returned as a line of text, formatted as follows:

```
status|%complete|resLayer(|errorMessage)
```

Where:		
	status	Current status of the run: 0=queued, 1=active, 2=completed, -1=error.
	%complete	An integer in the range 0-100 indicating current progress.
	resLayer	The index to the annotation layer for the image which contains the algorithm results (1-based). The results themselves may be retrieved using the GETANNOTATIONS request.
	errorMessage	In case of error, a text string giving more information.

GET image?DODBCALL+<method>+<inputxml>

`GET    image?DODBCALL+<method>+<inputxml>` - call a DataServer method

This HTTP request can be used to execute any image related DataServer method. The required **method** parameter is the DataServer method name. The [optional] **inputxml** parameter is an XML string that contains the parameters for the method. The return from this method is dependent on the specific DataServer method that was called.

GET image?GETACCESSLEVEL

`GET    image?GETACCESSLEVEL` - get user's access level for an image

This HTTP request returns a string that indicates the user's access level for an image. The returned string will be one of the following:

FullAccess – indicates that the user has full access to the image.
ReadOnly – indicates that the user has read-only access to the image.
NoAccess – indicates that the user has no access to the image.
LoginFailed – indicates that the chkauth request for the image failed.

GET image?GETUSERRIGHTS

`GET    image?GETUSERRIGHTS` - get user's rights for an image

This HTTP request returns a string that indicates the user's rights. The returned string will be of the format:

SaveAnnotations=x SaveMacro=x RunAnalysis=x ViewLabel=x
where: x is 1 if user has the right, 0 if not

GET directory/template.apml

```
GET   directory/template.apml
```
- displays directory using APML template

This request initiates APML template processing. The specified directory is opened and its contents are loaded into the APML dictionary, and then the specified template is processed. The HTML generated from the request is returned.

Please see "APML Template Processing" on page 12 and "APML Dictionary Information" on page 40 for more information.

GET image/template.apml

```
GET   image/template.apml
```
- displays image file using APML template

This request initiates APML template processing. The specified digital slide image is opened and its information is loaded into the APML dictionary, and then the specified template is processed. The HTML generated from the request is returned.

Please see "APML Template Processing" on page 12 and "APML Dictionary Information" on page 40 for more information.

GET GETMACROLIST

```
GET   GETMACROLIST
```
- return names of all available algorithm macros

This request returns the names of all available algorithm macros. A text line is returned for each macro, the format of each line is:

macroId,macroName

(The last line does not have a newline at end.) The macroId is a number identifying the macro, as assigned by the database; the macroName is a text string supplied when the macro is created.

GET GETMACROINFO?macroId

```
GET   GETMACROINFO?macroId
```
- return information for specified macro

This request returns information for the specified macro (referred by **macroId**). A block of XML is returned with the associated algorithm name, version, parameter values, etc.

GET GETNEXTANNOTATIONID

```
GET   GETNEXTANNOTATIONID
```
- return next available annotation Id

This request returns the next available annotation Id from the database, as applied to the next layer created. *This format of the request is deprecated*; the form with `image?GETNEXTANNOTATIONID` is preferred, as it enables the annotation Id to be associated with an image file (and subsequently to be cleaned up in the database).

GET RESET(?image)

```
GET   RESET(?image)
```
- resets caches and program statistics

This request clears file information and other caches.

If no image is specified, all files are closed, all directories and files are cleared from the file cache, the authorization cache is cleared, and all program statistics are reset. This requires administrator access if authorization checking is enabled.

If an image is given, the specified file is closed and cleared from the file cache, and all corresponding entries are cleared from the authorization cache. This requires update access to the specified file if authorization checking is enabled.

GET STATS?i

```
GET   STATS?i
```
- returns program statistics

This HTTP request causes a statistics "snapshot" to be written into the program logs and to be returned as text. An example of the response is shown below:

```
Fri Sep 23 19:14:04 === Aperio ImageServer v7.05W Sep 23 2005 19:08:55 (3916)
Fri Sep 23 19:14:04 === connections (cur/hwm/tot): 1/16/2958
Fri Sep 23 19:14:04 ===                  timeouts: 42
Fri Sep 23 19:14:04 ===           file reqs (tot): 201
Fri Sep 23 19:14:04 ===    dir reqs (tot-dir/I/T): 1729-118/1493/0
Fri Sep 23 19:14:04 ===  image reqs (tot-jpg/blk): 17712-5739/9835
Fri Sep 23 19:14:04 ===             (I/F/L/P/C)  : 1879/40/73/133/0
Fri Sep 23 19:14:04 ===             (S/T/R/D)    : 0/0/0/0
Fri Sep 23 19:14:04 ===     db reqs (GD/PD)      : 0/0
Fri Sep 23 19:14:04 ===             (GA/PA/RA/AS): 0/0/0/0
Fri Sep 23 19:14:04 ===  image data (total)      : 15544/262.50Mc/2.52Gu
Fri Sep 23 19:14:04 ===  image data (last hour)  : 211/4.29Mc/40.17Mu
Fri Sep 23 19:14:04 ===  image data (last minute): 23/130.11Kc/2.62Mu
Fri Sep 23 19:14:04 ===   image rps (total)      : 0.16/2.75Kc/27.03Ku
Fri Sep 23 19:14:04 ===   image rps (last hour)  : 0.06/1.22Kc/11.43Ku
Fri Sep 23 19:14:04 ===   image rps (last minute): 0.38/2.17Kc/44.80Ku
Fri Sep 23 19:14:04 ===       files (tot-hhwm)   : 446/2144
Fri Sep 23 19:14:04 ===      memory (avail/tot)  : 903/1527
Fri Sep 23 19:14:04 ===    cache MB (max/hwm/cur): 825/508/507
Fri Sep 23 19:14:04 === cache files (max/hwm/cnt): 1024/0/0
Fri Sep 23 19:14:04 === cache buffs (max/hwm/cnt): 41940/17147/17147
Fri Sep 23 19:14:04 ===  cache hits (hit/tot)    : 103482/209323
Fri Sep 23 19:14:04 ===  elapsed time (int/total): 1*03:07:16 /  1*03:07:16
```

If the **i** parameter is given, it specifies a refresh interval in seconds.

GET LOG?n+i

```
GET   LOG?n+i
```
- returns log entries

This HTTP request returns the n most recent log entries from the program log, as text. The default if n is omitted is 100. If the i parameter is given, it specifies a refresh interval in seconds.

GET TRACE?n

```
GET   TRACE?n
```
- sets log trace level

This HTTP request sets the log trace level as follows:

0 – normal logging; startup/shutdown and errors only
1 – tracing; each request is logged
2 – debug; additional information for each request
3 – message dumps; headers on requests/responses are dumped into log

The initial trace level is set from the **–v** flag, or defaults to 0 if not set. Higher trace levels place more information in logs but also fill logs faster and can slow down performance during periods of high activity.

APML Dictionary Information

This section describes the information which is placed in the APML dictionary for directory and image files. These data are placed in the dictionary prior to template substitution and processing for GET directory/template.apml (page 36) and GET image/template.apml requests (page 37). See also the overview section "APML Template Processing" on page 12.

During initialization, ImageServer creates a "global" APML dictionary and loads environment information into it. These values are constant. ImageServer also creates a sub-dictionary named "global.parms" which is populated with command-line parameter information. These values are also constant. When a template processing request is received, ImageServer creates a dictionary named "request" which is populated with information for the request. This dictionary is destroyed after the request is processed and re-created each time.

"global" dictionary values

Variables	
vers	ImageServer version
hostname	machine host name
port	access port number
ipaddr	IP address of machine (a.b.c.d)

"**global.parms" dictionary values**

Variables	
ipmask	IP address mask (-ipmask)
basedir	base directory path (-dir)
apmldir	APML template directory (-apml)
htmldir	HTML file directory (-html)
authost	authorization host name/address (-authip)
authport	authorization port number (-authprt)
dbhost	database host name/address (-dbip)
dbport	database port number (-dbprt)
label	label processing flag (-label)
aci	ACI processing flag (-aci)
ovly	Overlay image processing flag (-ovly)
icm	Integrated Color Management processing flag (-icm)
uttl	Title update flag (-uttl)
udb	Database update flag (-udb)

uannot	Annotation update flag (-uannot)
ualgo	Algorithm update flag (-ualgo)
reqtmot	Request processing timeout in seconds (-tmot)
cachelimit	Cache limit in MB (-cache)

The following values are always stored in the **request** dictionary:

"request" dictionary values – all requests

Variables	
uagent	HTTP user-agent string
ieflag	user-agent is Internet Explorer flag
winflag	platform is Windows flag
path	URI path for directory/file (doesn't include template)
upath	URL-encoded URI path (for use in HTML)
apmlfile	filename of APML template
isdir	whether request is for directory or file
rootdir	flag to indicate whether request is for base directory
apachehost	name of the Apache host that reverse-proxied the request
error	error message string; *only present if error occurred*

Tables	columns	
arg		one row for each positional parameter
	value	value of positional parameter

The following values are stored for directory requests (if opened successfully):

"request" dictionary values – directory requests

Variables	
dir	directory path to directory (doesn't include directory itself)
name	directory name
title	directory title, if any

Tables	columns	
dir		one row for each entry in directory
	name	name of directory entry
	type	"file" or "dir"
	size	size of entry in bytes
	title	entry title, if any
		following only appear for digital slide images:
	width	width of image in pixels
	height	height of image in pixels
	depth	depth of image in pixels (1 => 2D image)
	tilewidth	tile width of image in pixels
	tileheight	tile height of image in pixels (0 => stripped)
	appmag	apparent magnification (blank if unknown)
	mpp	microns per pixel (blank if unknown)
	imagesize	size of image in bytes
	comptype	compression/file type

compqual	compression quality
bigtiff	0/1 – whether image is BigTIFF
hasthumb	0/1 – whether image has thumbnail
hasmacro	0/1 – whether image has macro image
haslabel	0/1 – whether image has label image
scanscope	ScanScope ID (SS*xxxx*)
scandate	Date scanned (*mm*/*dd*/*yy*)
scantime	Time scanned (*hh*:*mm*:*ss*)
desc	image file description

The following values are stored for *image file* requests (if opened successfully):

"**request" dictionary values – image file requests**

Variables	
dir	directory path to image (doesn't include image itself)
name	image file name
size	image file size in bytes (compressed)
imagesize	image data size in bytes
width	width of image in pixels
height	height of image in pixels
depth	depth of image in pixels (1 => 2D image)
tilewidth	tile width of image in pixels
tileheight	tile height of image in pixels
appmag	apparent magnification (blank if unknown)
mpp	microns per pixel (blank if unknown)
comptype	compression type: 0=none, 1=LZW, 2=JPEG, 3=JPEG2000
compqual	compression quality (0-100)
compcodec	compression codec description
bigtiff	0/1 – whether image is BigTIFF
hasthumb	0/1 – whether image has thumbnail
thumbwidth	thumbnail width in pixels (if hasthumb=1)
thumbheight	thumbnail height in pixels (if hasthumb=1)
hasmacro	0/1 – whether image has macro image
macrowidth	macro image width in pixels (if hasmacro=1)
macroheight	macro image height in pixels (if hasmacro=1)
haslabel	0/1 – whether image has label image
labelwidth	label image width in pixels (if haslabel=1)
labelheight	label image height in pixels (if haslabel=1)
title	image file title, if any
desc	image file description

Tables	columns	
layers		one row for each layer in image file
	zoffset	z-offset value for layer (foremost=0)
	depth	layer depth in pixels

	focus	focus value for layer (-1 -> +1)

Tables	columns	
layers		one row for each layer in image file
	zoffset	z-offset value for layer (foremost=0)
	depth	layer depth in pixels
	focus	focus value for layer (float, -1 -> +1)
levels-*layer*		one table for each layer, e.g. layer 0 has **levels-0** each table has one row for each level in the layer
	width	width of layer in pixels
	height	height of layer in pixels
	zoom	zoom value for layer (float)

The following values are stored in the **request.annotations** dictionary for image file requests *if* the image has annotations:

"request.annotations" dictionary values – image file requests

Variables	
layers	number of annotation layers

Tables	columns	
layers		one row for each annotation layer
	Layer	layer number (1-based)
	Attributes	number of attributes for this layer
	Regions	number of regions for this layer
	Id	annotation Id (only used in database)
	ReadOnly	0 or 1 indicating whether layer is read-only
	LayerType	3 = algorithm, 4 = manual
	MarkupImage	path to markup image for layer, if any
	minX	minimum X value of regions in layer (relative to base)
	maxX	maximum X value of regions in layer
	minY	minimum Y value of regions in layer
	maxY	maximum Y value of regions in layer
attributes-*layer*		one row for each attribute for layer *layer*
	Index	index for attribute (1-based)
	Name	attribute name
	Value	attribute value
regions-*layer*		one row for each annotation region in layer
	Region	region number (1-based)
	Attributes	number of attributes for this region
	RegionType	0=polyline, 1=polygon, 2=ellipse, 3=arrow, 4=ruler
	Zoom	zoom level at which region was authored

	ImageLocation	path to markup image for region, if any
	minX	minimum X value of region (relative to base)
	maxX	maximum X value of region
	minY	minimum Y value of region
	maxY	maximum Y value of region
attributes-*layer*-*region*		one row for each attribute for region *region* of layer *layer*
	Index	index for attribute (1-based)
	Name	attribute name
	Value	attribute value

ACI File Format

ACI files (Aperio Composite Image) are text files which describe a set of image files to be "composited" together. The **–aci** parameter enables ImageServer to make ACI files visible as a single virtual file, in which case TIFF files are hidden (the component image files). If **–aci** is not specified ACI files are ignored and TIFF files are served normally.

ACI files may contain lines with one of four formats:

```
# <comment...>
<layer> <Z-offset>
<image_path>  <Y-offset>  <X-offset>  (<quality parameters...>)
<attribute> = <value>
```

Comment lines may be present in an ACI file, prefixed with a pound sign (#). Comment lines are ignored by ImageServer.

Layer lines specify the Z-offset for an image layer. All the following image lines define image files for that layer, until the next layer line is encountered. Layers must be described in ascending order of Z-offset. Any image lines before the first layer line are treated as belonging to the first layer with a Z-offset of zero. All layer lines are optional.

Image lines specify an <image_path>, a relative path to an image file to be included in the composite image. The <Y-offset> specifies the pixel offset to this image within the virtual image in the Y direction (up and down) relative to the upper left corner of the virtual image, and the <X-offset> specifies the pixel offset in the X direction (across). In some cases one or more quality parameters may also be present; they are ignored by ImageServer.

Attribute lines give various image attributes of the image. The image description is formed by concatenating all image attributes together, in the form <attribute>=<value>, delimited with vertical bar characters |.

When images are composited together, if they are overlapped, the image to the right or to the bottom is assumed to be “on top.” That is, the virtual image is formed by layering images from the upper left to the lower right.

The simplest form of an ACI file simply has one or more image lines, for example:

```
image1.tif  0  0
image2.tif  0  200
image3.tif  300 0
image4.tif  400 500
```

In this example **image1.tif** is located at (0,0), or aligned with the upper left corner of the virtual image. **Image2.tif** is located at (200,0), aligned with the top of the virtual image and offset 200 pixels to the right from the left edge. **Image3.tif** is located at (0,300), aligned with the left edge of the virtual image and offset 300 pixels down from the top, and **image4.tif** is located at (500,400), oriented 400 pixels down from the top of the virtual image and 500 pixels to the right from the left edge.

The boundary of the virtual image is the largest rectangle which passes completely through composited image data (that is, "jaggies" around the outside of the image are "trimmed"). Any regions within the boundaries of the virtual image which are not covered by any of the component images are filled with white pixels.

CWS File Format

CWS (Composite WebSlide) images are *directories*[10]. A CWS directory contains a large number of small JFIF files (.JPG), and one or two text .INI files which describe how the JFIF files are combined to form a single image. ImageServer makes CWS directories visible as a single virtual file; the components of the CWS directory are hidden. The **–cws** parameter enables ImageServer to make CWS directories visible as a single virtual file. If **–cws** is not specified, CWS directories are processed like ordinary directories.

The main reason to use CWS format digital slides is that CWS directories may be accessed remotely via any webserver *without any special server software*. For example, Microsoft IIS (Internet Information Server, a standard component of Microsoft Windows) may used for this purpose. Any CWS image directory can be copied to a webserver and then accessed remotely using tools like ImageScope or WebViewer, and analyzed using Aperio Algorithm Framework. However, ImageServer does provide full support for CWS files. In some situations it may be desirable to have both ImageServer and a standard webserver running side-by-side providing access to the same CWS files.

A CWS image is organized as one or more levels at different resolutions. Each level consists of a collection of JFIF files which “tile” the image at this resolution. The level is described by a .INI

[10]A Composite WebSlide, also known as a CWS slide, is a proprietary format created by Bacus Laboratories, Inc. (“Bacus”). WebSlide is a registered trademark of Bacus Laboratories, Inc.

file which gives the relative coordinates of the files within the level. The .INI files have the following names:

ScanSlide.INI -	Lowest resolution level, typically magnification of 1.5X. If the slide has only one level, only this .INI file may be present.
FinalScan.INI -	Higher resolution level(s), typically magnification of 20x or 40x. There may be intermediate levels between the highest resolution and the lowest resolution, in which case they are described in this file also.

The entire format of the .INI files is beyond the scope of this document, but coordinates of images within the files are given in stage units. A stage unit is about 1/8th of a micron. The stage unit coordinate system has (0,0) in the center of a "stage", and the images are positioned relative to this; stage units ascend to the *left* and *up*. Coordinates of individual images are given based on the *center* of each image. A typical image block size is 752 x 480 pixels, although other dimensions are possible and supported. CWS images may have multiple Z-layers for the highest resolution level.

Aperio ImageServer Programmer's Reference

MAN-0070, Revision D