

# Aperio DataServer

## Programmer's Reference

---



**©Copyright 2007 Aperio Technologies, Inc.**  
**Part Number/Revision: MAN-0066, Revision B**  
**Date: December 4, 2007**

This document applies to software versions Release 9.0 and later.

All rights reserved. This document may not be copied in whole or in part or reproduced in any other media without the express written permission of Aperio Technologies, Inc. Please note that under copyright law, copying includes translation into another language.

### User Resources

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

### Disclaimers

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.

### Trademarks and Patents

ScanScope is a registered trademark and ImageServer, TMA Lab, 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; and 7,116,440; 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:** Aperio Technologies, Inc.  
 1430 Vantage Court  
 Vista, CA 92081  
 United States

**European Office:** Aperio  
 3 The Sanctuary  
 Eden Office Park  
 Ham Green  
 Bristol BS20 0DD, UK

#### United States of America

Tel: 866-478-4111 (toll free)

Fax: 760-539-1116

**Customer Service** Tel: 866-478-4111 (toll free)

**Technical Support** Tel: 866-478-4111 (toll free)

Email: [support@aperio.com](mailto:support@aperio.com)

#### Europe

Tel: +44 (0) 1275 375123

Fax: +44(0) 1275 373501

**Customer Service** Tel: +44 (0) 1275 375123

**Technical Support** Tel: +44 (0) 1275 375123

Email: [europesupport@aperio.com](mailto:europesupport@aperio.com)

# Aperio DataServer Programmer's Reference

The Aperio DataServer provides an abstract interface to data stored in a backend database. Feature highlights are:

- A standard HTTP 1.1 server
- Implements SOAP for object access
- Abstracts applications from dependence on database schema
- Abstracts applications from dependence on backend database type
- Follows uniform application interface to specify input and report results
- Supports Image, Annotations, Macro and Job related methods
- Implements logic to support different slide scanning workflow scenarios
- Provides interface to Security database
- Implements logic that helps control access to Spectrum data

DataServer is a standard HTTP server that provides a list of methods to access data stored in standard databases. Currently DataServer supports only Microsoft SQL Server as the backend database. All data interchange is in xml format. To call any method, use the HTTP POST request with the SOAP header.

## Contents

<b>CONTENTS</b> .....	<b>3</b>
<b>INSTALLING DATASERVER</b> .....	<b>4</b>
<b>RUNNING DATASERVER</b> .....	<b>5</b>
<b>DATASERVER CLASSES</b> .....	<b>6</b>
<b>DATASERVER HTTP/SOAP HEADERS</b> .....	<b>6</b>
<b>APCONNECT</b> .....	<b>7</b>
<b>SAMPLE REQUEST RESPONSE</b> .....	<b>8</b>
<b>REQUEST/RESPONSE FOR METHODS EXPOSED BY IMAGE CLASS</b> .....	<b>8</b>
<b>Methods to Manage Image Meta-data</b> .....	<b>9</b>
<i>Method AddImage</i> .....	9
<i>Method ListImages</i> .....	10
<i>Method UpdateImageField</i> .....	10
<i>Method ResetImageField</i> .....	11
<i>Method GetImageData</i> .....	11
<i>Method GetImageData2</i> .....	12
<i>Method PutImageData</i> .....	13
<i>Method PutImageData2</i> .....	14
<i>Method GetImageId</i> .....	16
<i>Method CatalogueImage</i> .....	16
<i>Method CatalogueImage2</i> .....	17
<i>Method ListImagesByCase</i> .....	17

<i>Method GetAnnotations</i> .....	18
<i>Method PutAnnotations2</i> .....	18
<i>Method GetRecordImages</i> .....	19
<i>Method DeleteSlidesAndImages</i> .....	20
<b>Methods to Manage Spectrum Data Hierarchy</b> .....	<b>21</b>
<i>Method GetRecordDocuments</i> .....	21
<i>Method GetChildList</i> .....	21
<i>Method ListUnassignedRecords</i> .....	22
<i>Method GetFilteredRecordList</i> .....	22
<b>Methods to Manage Analysis Macros and Jobs</b> .....	<b>24</b>
<i>Method ListMacros</i> .....	24
<i>Method GetMacroInfo2</i> .....	24
<i>Method PutMacro</i> .....	25
<i>Method DeleteMacro</i> .....	25
<i>Method ListJobs</i> .....	26
<i>Method AddNewJob</i> .....	26
<i>Method CancelJob</i> .....	27
<b>Generic Data Management Methods</b> .....	<b>27</b>
<i>Method DeleteRecordsData</i> .....	27
<b>SPECTRUM SECURITY METHODS REQUEST/RESPONSE</b> .....	<b>28</b>
<b>Methods to Manage User Access</b> .....	<b>28</b>
<i>Method Logon</i> .....	28
<i>Method IsValidToken</i> .....	29
<i>Method Logoff</i> .....	29
<i>Method GenLogonReport</i> .....	30
<b>Methods to Manage Users</b> .....	<b>30</b>
<i>Method ListUsers</i> .....	30
<i>Method AddUsers</i> .....	31
<i>Method UpdateUser</i> .....	32
<i>Method ChangeUserPassword</i> .....	32
<i>Method DeleteUser</i> .....	32
<b>Methods to Manage User Access</b> .....	<b>33</b>
<i>Method ListDataGroups</i> .....	33
<i>Method AddDataGroup</i> .....	33
<i>Method UpdateDataGroup</i> .....	33
<i>Method DeleteDataGroup</i> .....	34
<i>Method ListAccessByUser</i> .....	34
<i>Method ListAccessByDataGroup</i> .....	35
<i>Method UpdateAccessByDataGroup</i> .....	35
<i>Method UpdateAccessByUser</i> .....	36

## Installing DataServer

Aperio DataServer comes with a self installing executable. Running the installer installs DataServer as a Windows service. DataServer also requires .NET SDK version 1.1 or higher be installed. This is also done automatically by the DataServer installer.

The backend database should be installed separately as should the database schema. DataServer currently works with only Microsoft SQL server, Microsoft SQL Server Express (installed by

default), or MSDE. The database schema corresponding to a particular version of DataServer can be installed by running its own installer provided by Aperio.

## Running DataServer

DataServer is installed as a Windows service that starts automatically when the computer starts. To stop DataServer, click on **Start >Run**. In the **Run** window, type **services.msc**. You should see the Services window. In the Services window, locate ApDataService. Right-click on ApDataService and click **Stop** in the context menu.

Restarting the computer will always restart DataServer. To manually restart DataServer, follow the procedure described in the previous paragraph and click on **Start** instead of **Stop** in the context menu.

DataServer has various configuration parameters. These are set in a file named:

*<DataServer Install Folder>\DataServer.exe.config*

where “DataServer Install Folder” is the path DataServer was installed to. Open this file in a text editor like Notepad and edit parameters. Note after saving your changes to these parameters, you must restart DataServer for these parameters to take effect.

The following are parameters that can be configured

<code>ip</code>	<code>&lt;ip_address&gt;</code>	- hostname or IP address (default = localhost)
<code>prt</code>	<code>&lt;port&gt;</code>	- Port number to listen on
<code>connString</code>	<code>&lt;string&gt;</code>	- Connection string to connect to the database
<code>spawnApp</code>	<code>&lt;app_path&gt;</code>	- Path to the application to be spawned

The **ip** parameter specifies the IP address that DataServer should listen on. Default value for this parameter is “0.0.0.0”. Leave it at this for DataServer to listen on all interfaces available on the machine. You can specify a dotted format IP address or a DNS name of the local machine. If the DNS name is specified, the first IP address that appears in the list of IP addresses associated with the machine is used.

The **prt** parameter specifies the port DataServer should listen on. By default this is 86.

The **connstring** parameter has 4 parts:

- *server*: This is the computer that hosts the database server. By default this is set to “localhost”. If the database server is on the same machine as DataServer, this need not be changed. Else change this to the DNS name of the machine running the database server.
- *database*: By default this is aperio. You should not have to change this.
- *user id*: User id to access database.
- *password*: Password for the database user.

The **spawnApp** parameter specifies an application to spawn after a scan. By default this is a zero length string. DataServer provides the ability to spawn any executable after a successful scan. Use this parameter to specify the executable it should spawn. To disable the feature, leave it as zero length string.

## DataServer Classes

Methods supported by Aperio DataServer are divided into 2 classes. Image related methods are implemented in `Aperio.Images` and Security related methods are implemented in `Aperio.Security`. The class must be specified as part of the URI in the HTTP header<sup>1</sup>. These classes provide application level interface abstracting schema and specific database system details.

`Aperio.Images` exposes methods that provide support for image metadata, annotation and analysis jobs related data. `Aperio.Security` exposes methods used to manage user accounts and access levels. It also exposes methods for verifying user credentials and checking access levels.

## DataServer HTTP/SOAP Headers

DataServer supports HTTP 1.1 format with some header fields that are required. Any field that is not required but is present will be ignored. DataServer supports only POST requests. An example of an HTTP header is shown below.

```
POST /Aperio.Images/Image.asmx HTTP/1.1  
Content-Length: length  
SOAPAction: http://www.aperio.com/webservices/MethodName
```

These are the required fields. Any additional fields in the header are ignored. The first line specifies DataServer class. DataServer supports 2 classes so this line has either `/Aperio.Images/Image.asmx` or `/Aperio.Security/Security2.asmx`. `Content-Length` specifies the content length after the end of header (after the 2 cr-lf). `SOAPAction` field is used to specify the method to be executed. Replace "`MethodName`" with the name of the method to be executed.

The content itself should contain the input XML for the web method called<sup>1</sup>. DataServer does not support namespaces and any namespace specified in the xml is ignored.

The response header is as follows:

```
HTTP/1.1 200 OK  
Content-Type: text/xml; charset=utf-8  
Content-Length: length
```

---

<sup>1</sup> Refer to later sections for a detailed description of input and output xml for each method.

*length* is the length of the body that has the output xml from the method call. This length is after the 2 cr-lf indicating the end of header. 200 OK is the only HTTP response that DataServer returns. If an error occurs, the error info is in the response body.

## ApConnect

ApConnect.dll implements a COM interface that makes it easy for COM aware applications to call webmethods exposed by DataServer. ApConnect abstracts HTTP and SOAP logic. It also handles the socket connection logic to the DataServer for the application. The most important method implemented in ApConnect IApdb interface is:

*HRESULT SendXMLMessage([in] BSTR MessageType, [in] BSTR XMLRequest, [in] BSTR Server, [in] short Port, [out, retval] long \*pStat)*

*MessageType* describes the method to be called and the class it is implemented in its format is */Aperio.Images/Image?MethodName* where Image is the class name.

*XMLRequest* is the input xml to the method.

*Server* is the DNS name or IP address of the machine hosting DataServer.

*Port* is the port DataServer is listening on.

*pStat* is the status that indicates if ApConnect could successfully call DataServer and obtain a response. If 0, the call was successful, else it indicates an error<sup>2</sup>.

Once this method succeeds (returns 0), call XMLResponse to get DataServer response.

*HRESULT XMLResponse([out, retval] BSTR \*pVal);*

*pVal* points to the string containing the response XML from DataServer. See later section for a description of this XML format.

A static library version of ApConnect is also available for applications that are not COM aware.

---

<sup>2</sup> Note success indicated here does not mean the method called succeeded. It only indicates successful communication with DataServer.

## Sample Request Response

ApConnect COM component and ApConnectLib both have similar interfaces for database access. Using either component, a sample access to *GetImageData* is illustrated below.

```
string method = "/Aperio.Images/Image?GetImageData";
string inpXML = "<ImageId>6125</ImageId>";
string outXML;

if(0 != SendXMLMessage( method, inpXML, "dbServer", 84 ))
    return error;
get_XMLResponse(outXML);
printf(outXML.c_str());
```

Output looks like

```
<GetImageDataResponse><GetImageDataResult><ASResult>0</ASResult><ASMessage></ASMessage></GetImageDataResult><ImageData><Image><ImageId>6125</ImageId><Rack>7</Rack><Slot>9</Slot><Location>C:\Stripes\6125.aci</Location><ScanDate>12/21/2005 4:28:52 AM</ScanDate><ScanScopeId>SS1001</ScanScopeId><CreatedBy>d1688306</CreatedBy><Description>none</Description><CompressDate>12/21/2005 4:33:44 AM</CompressDate><CompressedFileLocation>\\aperio-vsr\images\2005-12-21\6125.svs</CompressedFileLocation><FileName>6125</FileName><BarcodeId></BarcodeId><Assay></Assay><User1></User1><User2></User2><User3></User3><User4></User4><User5></User5><User6></User6><User7></User7><User8></User8><User9></User9><User10></User10><CaseId></CaseId><ScanStatus>Success</ScanStatus><ScanStatusDetails></ScanStatusDetails><QualityFactor>99.00000</QualityFactor></Image></ImageData></GetImageDataResponse>
```

## Request/Response for Methods Exposed by Image Class

Each method implemented has a set of input parameters and a set of output values. Each input parameter is an XML node as is each output value. Every method has a generic response xml format that looks like:

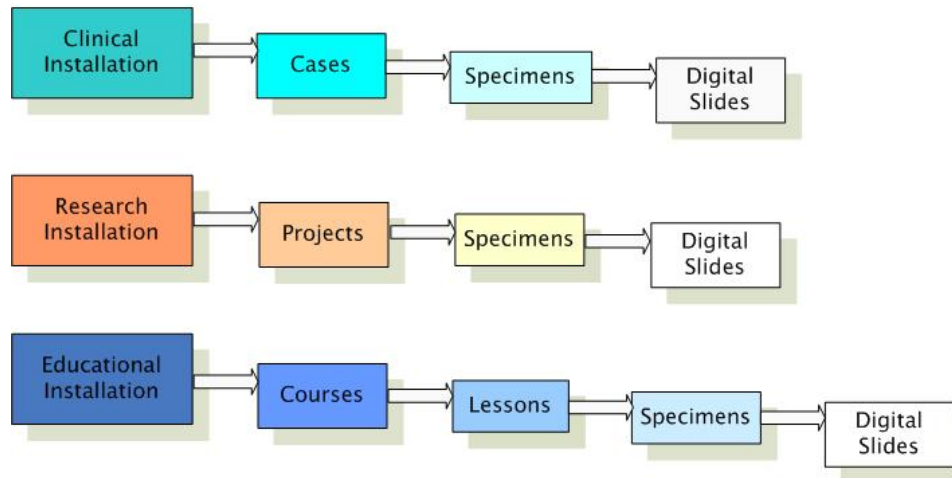
```
<MethodNameResponse>
  <MethodNameResult>

    <ASResult>nnnn</ASResult><ASMessage>...</ASMessage><ASDetails></ASDetails>
  </MethodNameResult>
  output xml
</MethodNameResponse>
```

Where ASResult has a numeric code and ASMessage has a text description of the status. 0 in ASResult node indicates success any negative value indicates failure. If the method called has any output, it is specified by the *output xml*.

In release 8, Aperio moved from flat image meta-data storage to a hierarchical model with the release of the Spectrum product. Along with this, DataServer started supporting methods to create, update and search through the hierarchical data-structure. Spectrum can be run in 3 modes—Education, Research or Clinical. Currently, only one mode is supported in any install of Spectrum. Each mode has a different hierarchy consisting of different data entities. The hierarchy is depicted in the image below. Each level is linked to its parent by a combination of ParentId, ParentTable.

### Spectrum Plus Configurations



Each level could also be associated with documents. In addition to this the images can be associated with Specimen. The hierarchy is used by DataServer when data is deleted and the permissions on any data is changed. The search functionality also assumes this hierarchy so searching for all cases in a project would result in an error. Refer to this information as we go through each method's behavior as appropriate.

## Methods to Manage Image Meta-data

Image meta-data includes data such as location, image type (scanned image, Z-Stacks, markup images etc.), scan information and annotations. Images associated with specimen and any non SVS scan images can be listed here as well. In general CompressedFileLocation field contains the location of the image and description contains some information about what type of an image it is. The following date formats are accepted by DataServer “MM/dd/yyyy H:mm:ss”, “yyyy-dd-MM\\Thh:mm:ss.fffzzz”. Time information is optional.

### *Method AddImage*

*Description:* Adds an image to the Image table and returns the image id.

Input xml:

```
<AddImageInfo>
  <ScanDate>mm/dd/yyyy hh:mm:ss</ScanDate>
</AddImageInfo>
```

Output xml:

```
<ImageId>nnnn</ImageId>
```

**Method ListImages**

Description: Lists image data filtered by ImageId >= StartingId and rowCount &lt;= MaxCount.

Input xml:

```
<StartingId>int</StartingId>
<MaxCount>int</MaxCount>
```

Output xml:

```
<ImageDataArray>
  <ImageData>
    <AcceptedDate></AcceptedDate>
    <CompressDate></CompressDate>
    <CompressedFileLocation></CompressedFileLocation>
    <CreatedBy></CreatedBy>
    <Description></Description>
    <FileName></FileName>
    <ImageId></ImageId>
    <ImageMarkedDeleted></ImageMarkedDeleted>
    <ImageStripesDeleted></ImageStripesDeleted>
    <Location></Location>
    <ScanDate></ScanDate>
    <ScanScopeId></ScanScopeId>
  </ImageData>
</ImageDataArray>
```

**Method UpdateImageField**

Description: Updates a single column in a row in the Image table filtered by ImageId.

Input xml:

```
<ImageId></ImageId>
<FieldName></FieldName>
<FieldValue></FieldValue>
```

Output xml: No output other than standard result

### Method *ResetImageField*

*Description:* Resets the value of a field in a row filtered by ImageId.

*Input xml:*

```
<ImageId></ImageId>  
<FieldName></FieldName>
```

*Output xml:* No output other than standard result

### Method *GetImageData*

*Description:* Returns image meta-data filtered using the following logic:

```
If ImageId is sepcified use ImageId  
else If CompressedFileLocation is specified use CompressedFileLocation  
else If BarcodeId is specified use BarcodeId  
else If Rack and Slot are specified Use Rack and Slot  
    If ScanScopeId is specified filter by ScanScopeId  
    AND ScanDate is NOT NULL
```

```
If Count not specified  
    return 1 row  
Else If Count lessthanorequal 0  
    return all images that match criteria  
else return requested count
```

*Input xml:*

```
<ImageId></ImageId>  
<CompressedFileLocation></CompressedFileLocation>  
<BarcodeId></BarcodeId>  
<Rack></Rack>  
<Slot></Slot>  
<ScanScopeId></ScanScopeId>  
<Location></Location>  
<Count></Count>
```

Output xml:

```

<ImageData><Image>
  <ImageId></ImageId>
  <Rack></Rack>
  <Slot></Slot>
  <Location></Location>
  <ScanDate></ScanDate>
  <ScanScopeId></ScanScopeId>
  <CreatedBy></CreatedBy>
  <Description></Description>
  <CompressDate></CompressDate>
  <CompressedFileLocation></CompressedFileLocation>
  <FileName></FileName>
  <BarcodeId></BarcodeId>
  <Assay></Assay>
  <User1></User1>
  ...
  <User10></User10>
  <CaseId></CaseId>
  <ScanStatus></ScanStatus>
  <ScanStatusDetails></ScanStatusDetails>
  <QualityFactor></QualityFactor>
</Image></ImageData>

```

**Method GetImageData2**

Description: Release 8 onwards, DataServer supports the second version of GetImageData. Other than some internal optimization, the difference is that the current version accepts a token and verifies user access to the requested data. Applications should transition to using this instead of GetImageData. Support for GetImageData will be removed in future releases.

Returns image meta data filtered using the following logic:

```

if ImageId is sepcified use ImageId
else if CompressedFileLocation is specified use CompressedFileLocation
else if BarcodeId is specified use BarcodeId
else if Rack and Slot are specified Use Rack and Slot
    if ScanScopeId is specified filter by ScanScopeId
    AND ScanDate is NOT NULL

```

If Count not specified

```
return 1 row
```

Else If Count lessthanorequal 0

```
return all images that match criteria
```

```
else return requested count.
```

Input xml:

```
<Token></Token>
<ImageId></ImageId><CompressedFileLocation></CompressedFileLocation><BarcodeId></BarcodeId>
<Rack></Rack><Slot></Slot><ScanScopeId></ScanScopeId><Location></Location><Count></Count>
```

Output xml:

```
<ImageData>
  <Image>
    <ImageId></ImageId><Rack></Rack><Slot></Slot><Location></Location>
    <ScanDate></ScanDate><ScanScopeId></ScanScopeId>
    <CreatedBy></CreatedBy><Description></Description>
    <CompressDate></CompressDate>
    <CompressedFileLocation></CompressedFileLocation>
    <FileName></FileName>
    <BarcodeId></BarcodeId><Assay></Assay>
    <User1></User1>...<User10></User10>
    <CaseId></CaseId><ScanStatus></ScanStatus>
    <ScanStatusDetails></ScanStatusDetails><QualityFactor></QualityFactor>
  </Image>
</ImageData>
```

**Method PutImageData**Description:

Accepts one row set and either updates or inserts Image table. PutImageData uses various fields to determine if data should be updated or inserted. In each case, a field is considered for filtering only if the corresponding node is present and the value is non zero length string.

The logic used to update is:

```
If ImageId is specified, always update by ImageId
else if CompressedFileLocation is specified and is found in the database, update
by CompressedFileLocation
else if BarcodeId is specified and is found in the database, update by BarcodeId
else if Rack AND Slot are specified
    if ScanScopeId is specified search for Rack, Slot, ScanScopeId and
    ScanDate != NULL,
    if found,
        use Rack, Slot, ScanScopeId and ScanDate != NULL to update
    else search for Rack, Slot and ScanDate != NULL, if found
        use Rack, Slot and ScanDate != NULL to update
else if Location is specified, search for Location, if found, update by Location
else insert
```

If specified BarcodeId already exists and corresponding ImageId does not match specified ImageId,  
error is returned.

*Input xml:*

```
<ImageId></ImageId>
<Rack></Rack>
<Slot></Slot>
<Location></Location>
<ScanDate></ScanDate>
<ScanScopeId></ScanScopeId>
<CreatedBy></CreatedBy>
<Description></Description>
<CompressDate></CompressDate>
<CompressedFileLocation></CompressedFileLocation>
<FileName></FileName>
<BarcodeId></BarcodeId>
<Assay></Assay>
<User1></User1>
...
<User10></User10>
<CaseId></CaseId>
<ScanStatus></ScanStatus>
<ScanStatusDetails></ScanStatusDetails>
<QualityFactor></QualityFactor>
```

*Output xml:*

```
<ImageData>
  <Image>
    <ImageId>nnnn</ImageId>
  </Image>
</ImageData>
```

## Method PutImageData2

*Description:* In release 8, DataServer supports the second version of PutImageData. Other than some internal optimization, the only difference is that the current version accepts a token and parent information and verifies user access to the requested data. Applications should transition to using this instead of PutImageData. Support for PutImageData will be removed in future releases.

Accepts a token, a row set and optionally a Parent table and Id.

PutImageData2 either updates or inserts Image table. PutImageData uses various fields to determine if data should be updated or inserted. In each case, a field is considered for filtering only if the corresponding node is present and the value is non zero length string.

The logic used to update is:

If ImageId is specified always update by ImageId  
 else if CompressedFileLocation is specified and is found in the database,  
 update by CompressedFileLocation  
 else if BarcodeId is specified and is found in the database,  
 update by BarcodeId  
 else if Rack AND Slot are specified  
 if ScanScopeId is specified search for Rack, Slot, ScanScopeId and ScanDate != NULL,  
 if found,  
     use Rack, Slot, ScanScopeId and ScanDate != NULL to update  
     else search for Rack, Slot and ScanDate != NULL, if found  
     use Rack, Slot and ScanDate != NULL to update  
 else if Location is specified,  
 search for Location, if found, update by Location  
 else insert

If specified BarcodeId already exists and corresponding ImageId does not match specified ImageId, error is returned.

Input xml:

```
<Token></Token>
<ParentTable></ParentTable>
<ParentId></ParentId>
<ImageData>
  <Image>
    <ImageId></ImageId><Rack></Rack><Slot></Slot>
    <Location></Location><ScanDate></ScanDate>
    <ScanScopeId></ScanScopeId><CreatedBy></CreatedBy>
    <Description></Description><CompressDate></CompressDate>
    <CompressedFileLocation></CompressedFileLocation>
    <FileName></FileName><BarcodeId></BarcodeId><Assay></Assay>
    <User1></User1> ...<User10></User10>
    <CaseId></CaseId><ScanStatus></ScanStatus>
    <ScanStatusDetails></ScanStatusDetails>
    <QualityFactor></QualityFactor>
  </Image>
</ImageData>
```

Output xml:

```
<ImageData>
  <Image>
    <ImageId>nxxx</ImageId>
  </Image>
</ImageData>
```

### Method GetImageId

*Description:* Searches for an ImageId using some specific input fields. If not found, adds a row, initializes the row with any specified data and returns ImageId. The following logic is used to find ImageId:

if BarcodeId is specified, find ImageId with specified BarcodeId  
 else if Rack AND Slot are specified, find ImageId with specified Rack, Slot AND ScanDate == NULL

If ImageId found using Rack and Slot,  
     Update ScanDate, StartStatus for the row, return ImageId  
 else if ImageId found using BarcodeId  
     Insert a new row, clear BarcodeId from old row, update new row with  
     ScanDate, StartStatus and BarcodeId  
     return new ImageId  
 else Insert a new row set all specified fields and return new ImageId.

*Input xml:*

```
<BarcodeId></BarcodeId>
<Rack></Rack>
<Slot></Slot>
<ScanDate></ScanDate>
<StartStatus></StartStatus>
<ScanScopeId></ScanScopeId>
```

*Output xml:*

```
<ImageData>
  <ImageId>nnnn</ImageId>
  <Assay></Assay>
  <User1></User1>
</ImageData>
```

### Method CatalogueImage

*Description:* Catalogs image data by ImageId. If ImageId is not specified, it returns an error.

*Input xml:*

```
<ImageId></ImageId>
<CreatedBy></CreatedBy>
<Location></Location>
<Description></Description>
<FileName></FileName>
```

*Output xml:* No output other than standard result

## Method CatalogueImage2

*Description:* Catalogs image data by ImageId. If ImageId is not specified, it returns an error.

*Input xml:*

```
<ImageId></ImageId>
<CreatedBy></CreatedBy>
<Location></Location>
<Description></Description>
<FileName></FileName>
<Rack></Rack>
<Slot></Slot>
<TWidth></TWidth>
<THeight></THeight>
<ScanStatus></ScanStatus>
<ScanStatusDetails></ScanStatusDetails>
<RunTime></RunTime>
<QualityFactor></QualityFactor>
```

*Output xml:* No output other than standard result

## Method ListImagesByCase

*Description:* Returns all at most MaxCount Images that have the specified CaseId. Set MaxCount to -1 to get all images.

*Input xml:*

```
<CaseId></CaseId><MaxCount></MaxCount>
```

Output xml:

```

<imgDataArr>
  <ImageData>
    <AcceptedDate></AcceptedDate>
    <CompressDate></CompressDate>
    <CompressedFileLocation></CompressedFileLocation>
    <CreatedBy></CreatedBy>
    <Description></Description>
    <FileName></FileName>
    <ImageId></ImageId>
    <ImageMarkedDeleted></ImageMarkedDeleted>
    <ImageStripesDeleted></ImageStripesDeleted>
    <Location></Location>
    <ScanDate></ScanDate>
    <ScanScopeId></ScanScopeId>
  </ImageData>
  <ImageData>...</ImageData>
</imgDataArr>
<ReturnedCount>nnnn</ReturnedCount>

```

**Method GetAnnotations**

*Description:* Returns Annotation layers for an Image. You can either get all layers for an Image (identified by ImageId) or get a specific layer (identified by AnnotationId and type). Either ImageId or AnnotationId has to be specified. Type is optional.

Input xml:

```

<ImageId></ImageId><AnnotationId></AnnotationId><Type></Type>

```

Output xml:

```

<AnnotationsNode>
  <Annotations>
    <Annotation Id="nnnn" Type="nnnn" attr1="abc" attr2=""> <ImageId></ImageId>
    <InputAnnotationId></InputAnnotationId>
    <Attributes>
      <Attribute Name="abc" Value="xyz"></Attribute>
      <Attribute Name="abc2" Value="xyz2"></Attribute> </Attributes>
    <Regions>regions xml</Regions>
  </Annotation>
</Annotations>
</AnnotationsNode>

```

**Method PutAnnotations2**

*Description:* Stores annotations in the database. PutAnnotations2 accepts multiple layers and stores them based on ImageId or AnnotationId. PutAnnotations2 returns an error if both ImageId and AnnotationId are not specified. All input layers that do not have AnnotationId specified are inserted. All input layers that have AnnotationId and are NOT

readonly (Type!=3 OR ReadOnly=0) are updated. All layers that exist in the database but are not specified in the input list are deleted.

**Input xml:**

```

<AnnotationsNode>
  <Annotations>
    <Annotation>
      <Annotation Id="nnnn" Type="nnnn" ReadOnly="0" attr1="abc" attr2="">
        <ImageId></ImageId>
        <InputAnnotationId></InputAnnotationId>
        <Attributes>
          <Attribute Name="abc" Value="xyz"></Attribute> <Attribute
            Name="abc2" Value="xyz2"></Attribute>
        </Attributes>
        <Author></Author>
        <CreateDate></CreateDate>
        <ModifiedDate></ModifiedDate>
        <Status></Status>
        <Regions>regions xml</Regions>
      </Annotation>
    </Annotations>
  </AnnotationsNode>

```

**Output xml:** No output other than standard result

### ***Method GetRecordImages***

**Description:** Accepts a token, table name and record id to look for. If token has read or higher access to specified record's DataGroup, then the method returns all images that are children of this record.

**Input xml:**

```

<Token></Token>
<TableName></TableName>
<Id></Id>

```

Output xml:

```

<ImageDataArray>
  <ImageData>
    <Image>
      <ImageId></ImageId>
      <Rack></Rack>
      <Slot></Slot>
      <Location></Location>
      <ScanDate></ScanDate>
      <ScanScopeId></ScanScopeId>
      <CreatedBy></CreatedBy>
      <Description></Description>
      <CompressDate></CompressDate>
      <CompressedFileLocation></CompressedFileLocation>
      <FileName></FileName>
      <BarcodeId></BarcodeId>
      <Assay></Assay>
      <User1></User1>
      ...
      <User10></User10>
      <CaseId></CaseId>
      <ScanStatus></ScanStatus>
      <ScanStatusDetails></ScanStatusDetails>
      <QualityFactor></QualityFactor>
    </Image>
  </ImageData>
</ImageDataArray>

```

**Method DeleteSlidesAndImages**

**Description:** Accepts a token, a list of slide and image ids to delete. If token has full access to specified record's DataGroup, then the method deletes slide and image records.

Input xml:

```

<SlideArray>
<Slide>
<SlideId>1</SlideId>
  <ImageArray>
    <ImageId/>
    <ImageId/>
  </ImageArray>
</Slide>
</SlideArray>

```

**Output xml:** No output other than standard result

## Methods to Manage Spectrum Data Hierarchy

Spectrum data tables have pre-set hierarchical relationships as shown in the diagram above. The following functions allow you to manage and search these data.

### *Method GetRecordDocuments*

*Description:* Accepts a token, table name and record id to look for. If token has read or higher access to specified record's DataGroup, then the method returns all documents that are children of this record.

*Input xml:*

```
<Token></Token>
<TableName></ TableName >
<Id></Id>
```

*Output xml:*

```
<GenericDataSet>
  <DataRow>
    <ClmName1>ClmVal1</ClmName1><ClmName2>ClmVal2</ClmName2>
  </DataRow>
  <DataRow>
    <ClmName1>ClmVal1</ClmName1><ClmName2>ClmVal2</ClmName2>
  </DataRow>
</GenericDataSet>
```

### *Method GetChildList*

*Description:* Accepts a token, Parent table, parent record id and a child id. If the user has access to the specified record data, all the first level children data is returned.

*Input xml:*

```
<Token></Token> <ParentTableName></ParentTableName><ParentId></ParentId>
<ChildTableName></ChildTableName>
```

Output xml:

```

<GenericDataSet>
  <DataRow>
    <ClmName1>ClmVal1</ClmName1><ClmName2>ClmVal2</ClmName2>
  </DataRow>
  <DataRow>
    <ClmName1>ClmVal1</ClmName1><ClmName2>ClmVal2</ClmName2>
  </DataRow>
</GenericDataSet>

```

**Method ListUnassignedRecords**

*Description:* Accepts a token and table name. This method returns all Spectrum records accessible by the user and is not associated to any other Spectrum record. Records can be paged.

Input xml:

```

<Token></Token> <TableName>int</TableName><MaxCount>int</MaxCount>
<PageIndex></PageIndex><RecordsPerPage></RecordsPerPage>

```

Output xml:

```

<GenericDataSet>
  <DataRow>
    <ClmName1>ClmVal1</ClmName1><ClmName2>ClmVal2</ClmName2>
  </DataRow>
  <DataRow>
    <ClmName1>ClmVal1</ClmName1><ClmName2>ClmVal2</ClmName2>
  </DataRow>
</GenericDataSet>

```

**Method GetFilteredRecordList**

*Description:* Accepts a token and various filter parameters. These parameters can be set to columns belonging to any level in the hierarchy as long as they all belong to the same hierarchy. The data returned is at the level specified by the *TableName* parameter. This method also supports grouping, sorting and paging of data.

If token is valid, returns a list of data records in the requested page that pass the filter and the user has access to along with access flags. The total number of records that pass the filter is also returned.

Parameter description:

**TableName:** This must be set to one of the levels in Spectrum hierarchy. The output records are from this table. This is a required parameter.

**ColumnList:** This is an optional parameter. It contains a list of columns to be included in the output. The format for this is [ColumnName1][ ColumnName2][ ColumnName3]. The columns listed here must belong to the level from which data is requested (indicated by **TableName** parameter). The exceptions to this rule are **DataGroupName** and **LastJobStatus**, the later is relevant only when requested level is **Slide**.

The **distinct** attribute forces the **DISTINCT** SQL operator to be applied. If **distinct** is requested when the column list has a memo (ntext SQL type) field, error occurs.

**PageIndex:** **PageIndex** along with **RecordsPerPage** is used to provide paging of record list. **PageIndex** is 1 based. If either **PageIndex** or **RecordsPerPage** is set to 0 or not specified, all records are returned.

**RecordsPerPage:** **RecordsPerPage** is used to specify how many records the application wants. This along with **PageIndex** specified the subset of record list that should be returned. If either **PageIndex** or **RecordsPerPage** is set to 0 or not specified, all records are returned.

**FilterBy:** Set of these parameters defines the search filter. Each node defines a search parameter.

The *Column* attribute is used to specify the database column to search on (e.g. *Id* column at the slide level). **DataGroupName** and **LastJobStatus** are special column names that may not be directly a part of the level you are searching at. **DataGroupName** can be used at any level whereas the **LastJobStatus** is relevant only at the *Slide* level.

The *FilterOperator* attribute is used to define the operator applied on that column. These are SQL operators. At this time only 4 operators are supported (=, <>, > and <).

*FilterValue* attribute is used to specify the value against which the operator is applied.

*Table* parameter is optional. If it is not specified, the column is assumed to belong to the data-table (table from which data is returned). This parameter would be used to do across hierarchy searching.

Input xml:

```
<Token></Token>
<TableName></TableName>
<ColmnList Distinct = "True/False">[ColumnName1][ ColumnName2]</ColumnList>
<PageIndex></PageIndex><RecordsPerPage></ RecordsPerPage >
<FilterBy Column="" FilterOperator="" FilterValue="" Table="" />
<FilterBy Column="" FilterOperator="" FilterValue="" />
<GroupBy>clmname</GroupBy>
<Sort By="ClmName" Order="Ascending/Descending">
```

Output xml:

```

<TotalRowCount></TotalRowCount>
<GenericDataSet>
  <DataRow>
    <ColumnName1>ColumnValue1</ColumnName1>
  </DataRow>
  <DataRow>
    <ColumnName1>ColumnValue1</ColumnName1>
  </DataRow>
</GenericDataSet>

```

## Methods to Manage Analysis Macros and Jobs

Spectrum supports running analysis jobs on dedicated job processing servers. A job takes a macro (indicating which algorithm to run), an image to be processed and optionally an annotation layer as input. The output from job processing is another annotation layer associated with the same image.

The following functions allow you to manage job queue as well as macros.

### *Method ListMacros*

*Description:* Accepts token and optional StartingId and MaxCount. If token is valid, this method returns a list of macro names and ids. If StartingId, MaxCount or both are specified in the input, these are used to filter the returned macro list.

*Input xml:*

```
<Token></Token> <MacroId></MacroId><MacroName></ MacroName >
```

*Output xml:*

```

<MacroDataArray>
  <MacroData>
    <MacroId></ MacroId><MacroName></MacroName>
  </MacroData>
  <MacroData>
    <MacroId></ MacroId><MacroName></MacroName>
  </MacroData>
</MacroDataArray>

```

### *Method GetMacroInfo2*

*Description:* Returns macro info for the specified MacroId.

*Input xml:*

```
<Token></Token> <MacroId></MacroId>
```

*Output xml:*

```

<MacroInfo Id="" Name="" MacroMarkedDeleted="">
  <Assay Id="" Name="">
    <Algorithms>
      <Algorithm Name="" Description="" SequenceOrder="">
        <Parameter Title="" Value="" Description="" Type="">
          <ParameterAttribute>
            <Min>0</Min><Max>2500</Max><Step>100</Step>
          </ParameterAttribute>
        </Parameter>
        <Parameter Title="" Value="" Description="" Type="">
          <ParameterAttribute>
            <Min>0</Min><Max>2500</Max><Step>100</Step>
          </ParameterAttribute>
        </Parameter>
      </Algorithm>
    </Algorithms>
  </Assay>
</MacroInfo>

```

### Method PutMacro

*Description:* Accepts token and MacroInfo node. If macro id specified, the macro is updated. Error is returned if the specified macro name already exists in the DB.

#### Input xml:

```

<Token></Token>
<MacroInfo Id="" Name="">
  <Algorithms>
    <Algorithm Id="" Name="" Description="">
      <Parameter Title="abc" Value="123" Description="" Type="n">
        <ParameterAttribute>
          <Min>0</Min><Max>25</Max><Step>1</Step>
        </ParameterAttribute>
      </Parameter>
      <Parameter Title="xyz" Value="456" Description="" Type="n">
        <ParameterAttribute>
          <Min>0</Min><Max>25</Max><Step>1</Step>
        </ParameterAttribute>
      </Parameter>
    </Algorithm>
  </Algorithms>
</MacroInfo>

```

#### Output xml:

```

<MacroId>n timer</MacroId>

```

### Method DeleteMacro

*Description:* Deletes macro identified by MacroId.

Input xml:

```
<Token></Token><MacroId></MacroId>
```

Output xml: No output other than standard result

**Method ListJobs**

Description: Accepts token and JobQueueId or ImageId or both. Job list filtered by ImageId, JobQueueId or both is returned. One of the identifiers must be specified.

Input xml:

```
<Token></Token> <JobQueueId></JobQueueId><ImageId></ImageId>
```

Output xml:

```
<JobDataArray>
  <JobData>
    <JobQueueId></JobQueueId><SubmittedDate></SubmittedDate>
    <ImageId></ImageId><ImageFileName></ImageFileName>
    <ImageDescription></ImageDescription>
    <CompressedFileLocation></CompressedFileLocation>
    <MacroId></MacroId><MacroName></MacroName>
    <InputParametersXML></InputParametersXML>
    <Priority></Priority><InputAnnotationId></InputAnnotationId>
    <ProcessDate></ProcessDate><Status></Status>
    <CompletedDate></CompletedDate><CreateMarkup></CreateMarkup>
  </JobData>
</JobDataArray>
```

**Method AddNewJob**

Description: Accepts a token and AddJobInfo node. Input to a job consists of image (identified by ImageId), optional annotation layer (identified by AnnotationId or relative layer index), a macro specifying input parameters to algorithm (identified by either a MacroId or macro xml string) and a flag to specify whether a markup is required.

ImageId has to be specified and an error is returned if specified ImageId is not found in the database.

Annotation layer can be specified by using either the absolute id in the database or a relative layer index for that image. The layer index is assumed to be 1 based (i.e., count starts at 1). The following logic is used to determine which layer to use when processing the image.

```
if InputAnnotationId not specified
  look for AnnotationLayerIndex
```

```

if InputAnnotationId == -1
    use last man-made annotation layer for the image
else if absolute InputAnnotationId specified
    use that layer
else if AnnotationLayerIndex specified
    use it as a 1 based index into the list of annotations for that image
else
    run on entire image

```

Macro can be specified either by using the identifier in the database (use ListMacros to get a list of macro ids) or by specifying the XML node with the parameter set (refer to GetMacroInfo2 documentation for XML node format).

Input xml:

```

<Token></Token>
<AddJobInfo>
    <ImageId></ImageId>
    <InputAnnotationId></InputAnnotationId>
    <AnnotationLayerIndex></AnnotationLayerIndex>
    <MacroId></MacroId>
    <CreateMarkup></CreateMarkup>
    <InputParametersXML>parameters xml from GetMacroInfo</InputParametersXML>
</AddJobInfo>

```

Output xml:

```

<JobQueueId>nnnn</JobQueueId>

```

### Method CancelJob

Description: Cancel a set of jobs in progress or not yet started.

Input xml:

```

<Token></Token>
<JobId>nnnn</JobId>
<JobId>nnnn</JobId>...

```

Output xml: No output other than standard result

## Generic Data Management Methods

Spectrum provides some generic methods to manage data in the database.

### Method DeleteRecordsData

Description: Deletes record(s) from the database.

*Input xml:*

```
<TableName>Name</TableName>
<IdArray>
  <Id>nnnn</Id>
</IdArray>
```

*Output xml:* No output other than standard result

## Spectrum Security Methods Request/Response

Spectrum security is implemented using Users, Privileges, DataGroups and AccessLevels. Every instance of data in Spectrum belongs to a DataGroup and to only one DataGroup. There are Users in the system whose access to these DataGroups can be set up using AccessLevels. There are 3 levels of access that can be set for a user/DataGroup pair: None, ReadOnly and Full. Users also have Privileges. User Privileges determine what a kind of actions the user can take on various data. Right now Spectrum supports only Administrator/Non-Administrator Privileges. Users with Administrator privileges can change users' access to DataGroups as well as perform other user maintenance actions. Users with admin privileges also have full access to all DataGroups.

All methods in DataServer accept a token that identifies the user making the request. A token can be obtained by calling the Logon method of Security2 class. The AccessLevel/Privileges associated with the user/DataGroup pair is used by DataServer to determine whether the requested method can be executed by the owner of the token. Tokens expire and the expiration time can be set in DataServer config file using the AuthCacheTimeout parameter.

Security related methods have a similar interface to Image related methods. Every method accepts a token, input data and returns a status error, message and output data. Starting with release 8, DataServer exposes the second version of Security class. The URI to get to this class is */Aperio.Security/Security2*. The 1.0 version of security class supported in previous releases is now obsolete.

Following is a list of methods exposed by the security class.

### Methods to Manage User Access

Any Spectrum user must logon to Spectrum before accessing any data. The following methods allow you to get a token and audit user activity in the system.

#### *Method Logon*

*Description:* This method accepts two parameters UserName and PassWord.

*Input xml:*

```
<UserName> </UserName><PassWord></PassWord>
```

*Output xml:*

```
<Token>{0}</Token>
<UserData>
  <UserId>{0}</UserId>
  <FullName>{0}</FullName>
  <LoginName>{0}</LoginName>
  <UserMustChangePassword>true/false</UserMustChangePassword>
  <Privileges>
    <AdminUser>true/false</AdminUser>
  </Privileges>
  <LastLoginTime>{0}</LastLoginTime>
</UserData>
```

*Remarks:* Use the token returned to call all other DataServer methods. If the UserMustChangePassword flag is set, the user cannot access any DataServer method except Logon and PutUserData. User has to change password before being granted access to other methods. Refer to the Spectrum user guide for information regarding enabling/disabling this feature.

**Method IsValidToken**

*Description:* Accepts a token and verifies if it is still valid.

*Input xml:*

```
<Token></Token>
```

*Output xml:*

```
<Valid>True/False</Valid>
```

**Method Logoff**

*Description:* Logs off user from the system and invalidates the token.

*Input xml:*

```
<Token>abcd</Token>
```

*Output xml:* No output other than standard result

### Method *GenLogonReport*

*Description:* GenLogonReport generates a report of user login activity based on the filters specified. Admin privilege is required to execute this method. Any number of filters can be specified. Each filter is linked to the next by an AND logical operator. Admin privileges are required to execute this method.

*Input xml:*

```
<Token></Token>
<FilterBy Column="ClnName" FilterOperator="" FilterValue="">
<FilterBy Column="ClnName" FilterOperator="" FilterValue="">
```

*Output xml:*

```
<AuthEventReport>
  <AuthEvent>
    <UserId></UserId>
    <LoginName></LoginName>
    <EventName></EventName>
    <EventTime></EventTime>
    <EventResult></EventResult>
    <EventDetails></EventDetails>
  </AuthEvent>
</AuthEventReport>
```

Remarks: DataServer maintains record of 3 login events: Logon, Logoff and Logon Timeout. This list may be expanded to include other events in the future. Each one of these events can have Successful or Failed status. Each of these events is recorded against the UserId of the user that executed these events.

So, the *Column* parameter can be one of the following: EventName, EventResult, or UserId.

## Methods to Manage Users

The following methods allow you to manage Spectrum users.

### Method *ListUsers*

*Description:* Accepts token and if valid, returns the list of users. User must have admin privileges to execute this request.

*Input xml:*

```
<Token></Token>
```

Output xml:

```

<UserDataArray>
  <UserData>
    <UserId>nnnn</UserId>
    <LoginName>abc</LoginName>
    <FullName>def ghi</FullName>
    <IsGuest>T/F</IsGuest>
    <IsLocked>T/F</IsLocked>
    <IsExpired>T/F</IsExpired>
    <ExpireDate>T/F</ExpireDate>
    <LastLogin>yyyy-MM-dd HH:mm</LastLogin>
    <Privileges>
      <AdminUser>T/F</AdminUser>
    </Privileges>
    <AccountStatus>
      <LockOnInvalidPass>T/F</LockOnInvalidPass>
      <LockIfUnused>T/F</LockIfUnused>
      <UserMustChangePassword>
        T/F
      </UserMustChangePassword>
      <PasswordCanExpire>T/F</PasswordCanExpire>
    </AccountStatus>
    <IsSystemUser>T/F</IsSystemUser>
  </UserData>
</UserDataArray>

```

**Method AddUsers**

Description: Accepts a token and user data. If token is valid and has admin access, the specified user is added to Spectrum. If successful, UserId of the new user is returned.

Input xml:

```

<Token>
<UserData>
  <LoginName></LoginName>
  <Password></Password>
  <FullName></FullName>
  <Privileges></Privileges>
  <AccountStatus></AccountStatus>
</UserData>

```

Output xml:

```

<UserId></ UserId >

```

### Method UpdateUser

*Description:* Accepts a token and user data. If token is valid, the specified Spectrum user is added or updated. Users can update their own data. To update other users' data, admin privileges are required.

*Input xml:*

```
<Token>
<UserData>
  <LoginName></LoginName>
  <Password></Password>
  <FullName></FullName>
  <Privileges></Privileges>
  <AccountStatus></AccountStatus>
</UserData>
```

*Output xml:*

```
<UserId></ UserId >
```

### Method ChangeUserPassword

*Description:* If user's "must change password" flag is set, this method can be used to change only the password.

*Input xml:*

```
<Token>
<UserData>
  <UserId></UserId>
  <Password></Password>
</UserData>
```

*Output xml:* No output other than standard result

### Method DeleteUser

*Description:* Deletes the specified user from Spectrum. If the user is logged on, the token is invalidated.

*Input xml:*

```
<Token></Token>
<UserId></UserId>
```

*Output xml:* No output other than standard result

## Methods to Manage User Access

The following methods allow you to manage DataGroups and access controls.

### *Method ListDataGroups*

*Description:* Accepts a token and lists all DataGroups. Admin privileges are required to execute this method.

*Input xml:*

```
<Token></Token>
<StartingId>nnnn</StartindId>
<Count></Count>
```

*Output xml:*

```
<DataGroupArray>
  <DataGroup>
    <Id></Id><Name></Name>
    <Description></Description>
    <Default>True/False</Default>
  </DataGroup>
</DataGroupArray>
```

### *Method AddDataGroup*

*Description:* Accepts a token and data about a DataGroup and adds it to Spectrum. Admin privileges are required to execute this method. If successful, the id is returned. The name of the DataGroup cannot be "Default". Also, duplicate names are not allowed.

*Input xml:*

```
<Token></Token>
<DataGroup>
  <Name></Name>
  <Description></Description>
</DataGroup>
```

*Output xml:*

```
<DataGroupId>nnnn</ DataGroupId >
```

### *Method UpdateDataGroup*

*Description:* This method accepts a token and updates DataGroup information. The DataGroup to be updated is identified by the Id. Admin privileges are required to execute this method.

*Input xml:*

```
<Token>
<DataGroup>
  <Id></Id>
  <Name></Name>
  <Description></Description>
</DataGroup>
```

*Output xml:*

```
<DataGroupId></DataGroupId>
```

**Method DeleteDataGroup**

*Description:* Accepts a token and Id of the DataGroup to be deleted. Admin privileges are required to execute the method. “Default” DataGroup cannot be deleted. All data belonging to the deleted DataGroup are moved to default DataGroup.

*Input xml:*

```
<Token></Token>
<DataGroupId></DataGroupId>
```

*Output xml:* No output other than standard result

**Method ListAccessByUser**

*Description:* This method lists access levels to all DataGroups for a particular user. If a UserId is not specified, AccessLevels are returned for the user that owns the token. To list AccessLevels of other users, admin privileges are required.

*Input xml:*

```
<Token></Token>
<UserId></UserId>
```

*Output xml:*

```
<AccessDataByUserArray>
  <UserId>ppp</UserId>
  <FullName>abc def</FullName>
  <AccessDataByUser>
    <DataGroupId>qqq</DataGroupId>
    <DataGroupName>ghi jkl</DataGroupName>
    <AccessLevel>rrr</AccessLevel>
    <Default>True/False</Default>
  </AccessDataByUser>
</AccessDataByUserArray>
```

**Method *ListAccessByDataGroup***

*Description:* This method lists access to a DataGroup for all users. Admin privileges are required to run this method.

*Input xml:*

```
<Token></Token>
<DataGroupId></DataGroupId>
```

*Output xml:*

```
<AccessDataByDataGroupArray>
  <DataGroupId>ppp</DataGroupId>
  <Name>abc def</Name>
  <AccessDataByDataGroup>
    <UserId>qqq</UserId>
    <FullName>ghi jkl</FullName>
    <AccessLevel>rrr</AccessLevel>
  </AccessDataByDataGroup>
</AccessDataByDataGroupArray>
```

**Method *UpdateAccessByDataGroup***

*Description:* Accepts a token and an array of access data for a DataGroup for a set of users and updates them. Admin privileges are required to execute this method.

*Input xml:*

```

<Token> </Token>
<AccessDataByDataGroupArray>
  <DataGroupId>ppp</DataGroupId>
  <Name>abc def</Name>
  <AccessDataByDataGroup>
    <UserId>qqq</UserId>
    <FullName>ghi jkl</FullName>
    <AccessFlags a1="T/F" a2="T/F"/>
  </AccessDataByDataGroup>
</AccessDataByDataGroupArray>

```

*Output xml:* No output other than standard result

### Method *UpdateAccesByUser*

*Description:* Accepts a token and access array of a user for a set of DataGroups and updates the user's access. Admin privileges are required to run this method.

*Input xml:*

```

<Token></Token>
<AccessDataByUserArray>
  <UserId>ppp</UserId>
  <FullName>abc def</FullName>
  <AccessDataByUser>
    <DataGroupId>qqq</UserId>
    <Name>ghi jkl</Name>
    <AccessFlags a1="T/F" a2="T/F"/>
  </AccessDataByUser>
</AccessDataByUserArray>

```

*Output xml:* No output other than standard result



**Aperio DataServer Programmer's Reference**  
**MAN-0066, Revision B**