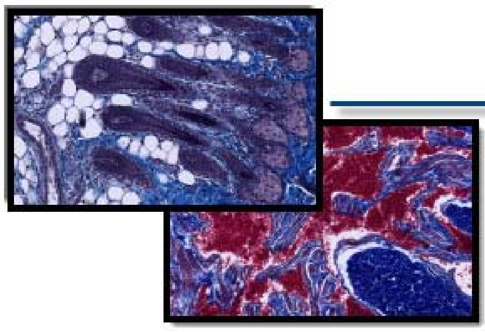
Aperio DataServer

Programmer's Reference



aperio®

User Resources

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

Disclaimers

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

This manual is not a substitute for the detailed operator training provided by Aperio or for other advanced instruction. Aperio 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 Service Engineer.

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

Aperio products are FDA cleared for specific clinical applications, and are intended for research and educational use for other applications. They are not approved by the FDA for primary diagnosis. For clearance updates, visit www.aperio.com.

Aperio grants to those users of Aperio 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 the express written consent of Aperio and its origin is not misrepresented.

Trademarks and Patents

Aperio and ScanScope are registered trademarks and Genie, ImageScope, Aperio ePathology Solutions, eSlideShare, and eSlide Manager are trademarks of Aperio. 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 1360 Park Center Drive Vista, CA 92081 United States	Aperio UK Ltd Prama House 167 Banbury Road Summertown Oxford OX2 7HT United Kingdom	Aperio 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) Email: info@aperio.com	**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 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 eSlide Manager 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

Prerequisites

Name	Details
DataServer	The DataServer service should be properly installed and configured. DataServer is installed as a Windows service that starts automatically when the computer starts. To ensure the DataServer service is running, click on **Start >Run**. In the **Run** window, type **services.msc**. The Services window should display. In the Services window, locate **ApDataService** and ensure it **Status** is **Started**.
Microsoft SQL Server	Aperio DataServer currently works with only Microsoft SQL server and Microsoft SQL Server Express (installed by default). Make sure SQL Server is installed and the Aperio database schema has been applied.

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[1]. 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[1]. 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
```

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.

Request/Response for Methods Exposed by Image Class

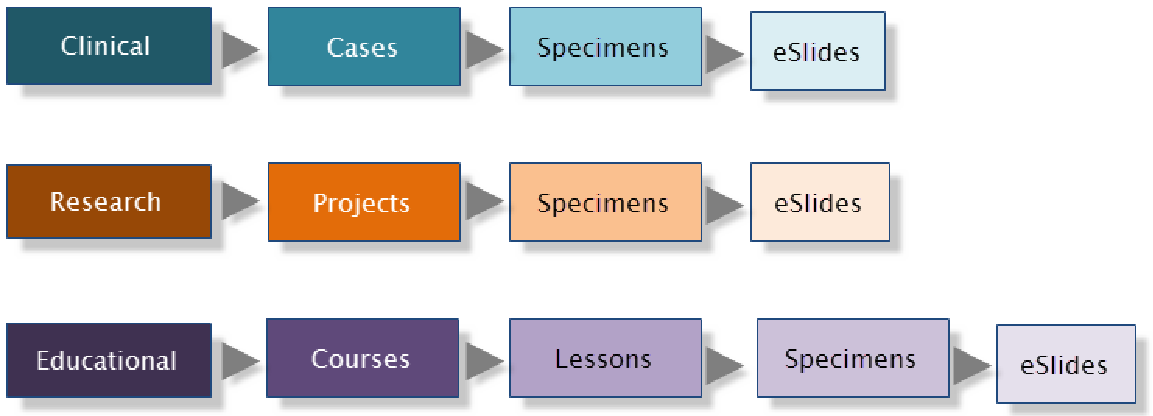
Each method implemented has a set of input parameters and a set of output values. Each input parameter is an XML node as is each output value. Every method has a generic response xml format that looks like:

```
<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 eSlide Manager product. Along with this, DataServer started supporting methods to create, update and search through the hierarchical data-structure. eSlide Manager can be run in 3 modes—Education, Research or Clinical. Currently, only one mode is supported in any install of eSlide Manager. 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.

[1] Refer to later sections for a detailed description of input and output xml for each method.



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.

Sample HTTP/SOAP Request Tool (DsClient)

DsClient is a tool that provides an easy way to test and validate the XML input to DataServer methods and evaluate the response returned. To use the tool you simply need to supply the following inputs:

Under ***Connection*** (DataServer location and authentication):

- ***Server Name*** - This is the computer that hosts DataServer. By default this is set to "localhost". If DsClient is being run on the same machine as DataServer, this need not be changed. Else change this to the DNS name of the machine running DataServer.
- ***Port*** - Port number on which the DataServer HTTP service is configured to listen.
- ***User Name*** - User ID of account authorized to access DataServer.
- ***Password*** - Password of account authorized to access DataServer.

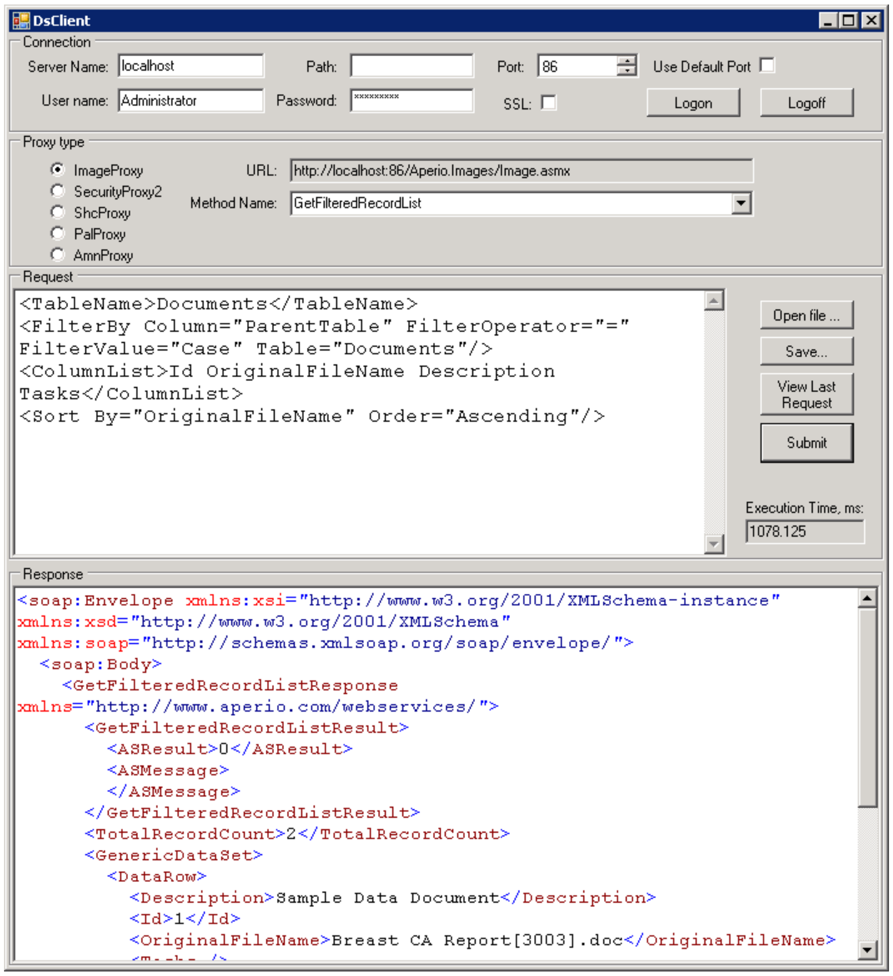
Under ***Proxy Type*** (DataServer class and method to execute):

- ***Proxy Type*** - Select the radio button next to the class that contains the method to be executed.
- ***Method Name*** - Name of the method to be executed.

Under ***Request*** (DataServer method input parameters):

- ***Request*** - Place the XML input parameters with data into the Request box for the method to be executed. The XML input should not include the HTTP/SOAP headers or footers.

Once the required data has been entered into the DsClient tool, click on the **Logon** button near the top right of the tool. This will validate the *Connection* inputs and create a session token for the user. Once a successful logon has been achieved, click the **Submit** button on the lower right of the *Request* input box. This will submit the HTTP/SOAP request to DataServer and then populate the *Response* box with the returned results.



```
<TableName>Documents</TableName>
<FilterBy Column="ParentTable" FilterOperator="="
FilterValue="Case" Table="Documents"/>
<ColumnList>Id OriginalFileName Description
Tasks</ColumnList>
<Sort By="OriginalFileName" Order="Ascending"/>
```

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetFilteredRecordListResponse
xmlns="http://www.aperio.com/webservices/">
      <GetFilteredRecordListResult>
        <ASResult>0</ASResult>
        <ASMessage>
        </ASMessage>
      </GetFilteredRecordListResult>
      <TotalRecordCount>2</TotalRecordCount>
      <GenericDataSet>
        <DataRow>
          <Description>Sample Data Document</Description>
          <Id>1</Id>
          <OriginalFileName>Breast CA Report[3003].doc</OriginalFileName>
```

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 GetImageData2

Description: The same functionality as GetImageData, but it can return multiple found records, not only the first one.

Input xml:

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

Output xml:

```
<GetImageDataResponse>
        <GetImageDataResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </GetImageDataResult>
        <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>
                        <ScanStatusDetails></ScanStatusDetails>
                        <QualityFactor></QualityFactor>
                </Image>
        </ImageData>
  </GetImageDataResponse>
```

Method PutImageData2

Description: 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 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
CompresseFileLocation
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, an error is returned.

Input 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>
                <Barcode></Barcode>
                <Assay></Assay>
                <CaseId></CaseId>
                <ScanStatus></ScanStatus>
                <ScanStatusDetails></ScanStatusDetails>
                <QualityFactor></QualityFactor>
        </Image>
</ImageData>
```

Output xml:

```
<PutImageData2Response>
        <PutImageData2Result>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
```

```
		</PutImageData2Result>
		<ImageData>
			<Image>
				<ImageId>nnnn</ImageId>
			</Image>
		</ImageData>
	</PutImageData2Response>
```

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:

```
<PutImageDataResponse>
	<PutImageDataResult>
		<ASResult>nnnn</ASResult>
		<ASMessage></ASMessage>
	</PutImageDataResult>
	<ImageData>
		<ImageId>nnnn</ImageId>
		<Assay></Assay>
		<User1></User1>
	</ImageData>
</PutImageDataResponse>
```

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>
<CompressedFileLocation></CompressedFileLocation>
```

Output xml: No output other than standard result

```
<CatalogueImage2Response>
        <CatalogueImage2Result>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </CatalogueImage2Result>
</CatalogueImage2Response>
```

Method GetAnnotations2

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>
or
<GenieTrainingSetId></GenieTrainingSetId>
or
<AnnotationTemplateId></AnnotationTemplateId>
and/or
<AnnotationId></AnnotationId>
<Type></Type>
```

The following parameter causes the method to return an encoded string (used by Spectrum) as opposed to the xml.

```
<ResultAsString>true/false</ResultAsString>
```

Output xml:

```
<GetAnnotationsResponse>
	<GetAnnotationsResult>
		<ASResult>nnnn</ASResult>
		<ASMessage></ASMessage>
	</GetAnnotationsResult>
	<AnnotationsNode>
		<Annotations>
			<Annotation Id="nnnn" Type="nnnn" attr1="abc" attr2="">
				<ImageId></ImageId>
				<GenieTrainingSetId></GenieTrainingSetId>
				<AnnotationTemplateId></AnnotationTemplateId>
				<ResultAsString>true/false</ResultAsString>
				<InputAnnotationId></InputAnnotationId>
				<Attributes>
					<Attribute Name="abc" Value="xyz"></Attribute>
					<Attribute Name="abc2" Value="xyz2"></Attribute>
				</Attributes>
				<Regions>regions xml</Regions>
			</Annotation>
		</Annotations>
	</AnnotationsNode>
</GetAnnotationsResponse>
```

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:

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

Output xml:

```
<GetAnnotationsResponse>
   <GetAnnotationsResult>
           <ASResult>nnnn</ASResult>
           <ASMessage></ASMessage>
   </GetAnnotationsResult>
   <AnnotationIdMapping>
           <AnnotationId New="-3" Old="3353"/>
   </AnnotationIdMapping>
</GetAnnotationsResponse>
```

Method PutAnnotations3

Description: Stores annotations in the database. PutAnnotations3 accepts multiple layers and stores them based on ImageId or AnnotationId. PutAnnotations3 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. Unlike PutAnnotations2, all layers that exist in the database but are not specified in the input list are NOT deleted.

Input xml:

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

Output xml:

```
<GetAnnotationsResponse>
    <GetAnnotationsResult>
            <ASResult>nnnn</ASResult>
            <ASMessage></ASMessage>
    </GetAnnotationsResult>
    <AnnotationIdMapping>
            <AnnotationId New="-3" Old="3353"/>
    </AnnotationIdMapping>
</GetAnnotationsResponse>
```

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:

```
<GetRecordImagesResponse>
        <GetRecordImagesResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </GetRecordImagesResult>
        <ImageDataArray>
                <ImageData>
                        <AcceptedDate></AcceptedDate>
                        <CompressDate></CompressDate>
                        <CompressedFileLocation></CompressedFileLocation>
                        <CreatedBy></CreatedBy>
                        <Description></Description>
                        <FileName></FileName>
                        <ImageId></ImageId>
                        <ImageStripesDeleted></ImageStripesDeleted>
                        <Location></Location>
                        <ScanDate></ScanDate>
                        <ScanScopeId></ScanScopeId>
                </ImageData>
        </ImageDataArray>
</GetRecordImagesResponse>
```

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.

****WARNING**** The image file will also be permanently deleted from ImageServer and cannot be recovered.

Input xml:

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

Output xml: No output other than standard result

```
<DeleteSlidesAndImagesResponse>
    <DeleteSlidesAndImagesResult>
            <ASResult>nnnn</ASResult>
            <ASMessage></ASMessage>
    </DeleteSlidesAndImagesResult>
```

```
</DeleteSlidesAndImagesResponse>
```

Methods to Manage eSlide Manager Data Hierarchy

eSlide Manager 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:

```
<GetRecordDocumentsResponse>
        <GetRecordDocumentsResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </GetRecordDocumentsResult>
        <GenericDataSet>
                <DataRow>
                        <ColumnName1>ColumnValue1</ColumnName1>
                </DataRow>
                <DataRow>
                        <ColumnName1>ColumnValue1</ColumnName1>
                </DataRow>
        </GenericDataSet>
</GetRecordDocumentsResponse>
```

Method GetChildList

Description: Accepts a token, parent table name, parent id and child table name. If token is valid and user can view parent data, it Returns a list of child data records that belong to the specified parent record. Each row has the AccessFlags for that record.

Input xml:

```
<Token></Token>
<InstallationMode></InstallationMode>
<ParentTableName></ParentTableName>
```

```
<ParentId></ParentId>
<ChildTableName></ChildTableName>
```

Output xml:

```
<GetChildListResponse>
        <GetChildListResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </GetChildListResult>
        <GenericDataSet>
                <DataRow>
                        <ColumnName1>ColumnValue1</ColumnName1>
                </DataRow>
                <DataRow>
                        <ColumnName1>ColumnValue1</ColumnName1>
                </DataRow>
        </GenericDataSet>
</GetChildListResponse>
```

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.

*Note - GetFilteredRecordList can be used to query any table in the database.

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

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>
<RequestType>DataAndCount, DataOnly, or CountOnly</RequestType>
<FilterBy Column="" FilterOperator="" FilterValue="" />
<FilterBy Column="" FilterOperator="" FilterValue="" />
<GroupBy>clmname</GroupBy>
<Sort By="ClmName" Order="Ascending/Descending">
<PageIndex></PageIndex>
<RecordsPerPage></RecordsPerPage>
```

Output xml:

```
<GetFilteredRecordListResponse>
        <GetChildListResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </GetFilteredRecordListResult>
        <TotalRowCount></TotalRowCount>
        <GenericDataSet>
                <DataRow>
                        <ColumnName1>ColumnValue1</ColumnName1>
                </DataRow>
                <DataRow>
                        <ColumnName1>ColumnValue1</ColumnName1>
                </DataRow>
        </GenericDataSet>
</GetFilteredRecordListResponse>
```

Methods to Manage Analysis Macros and Jobs

eSlide Manager 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 GetMacroInfo2

Description: Returns macro info for the specified MacroId.

Input xml:

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

Output xml:

```
<GetMacroInfoResponse>
        <GetMacroInfoResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </GetMacroInfoResult>
        <MacroInfo>
                <MacroId></MacroId>
                <MacroName></MacroName>
                <MacroMarkedDeleted></MacroMarkedDeleted>
                <Algorithms>
                        <MacroAlgorithm>
                                <MacroAlgorithmId></MacroAlgorithmId>
                                <AlgorithmName></AlgorithmName>
                                <SequenceOrder></SequenceOrder>
                                <LocationPath></LocationPath>
                                <Parameters>
                                        <MacroParameterData></MacroParameterData>
                                        <MacroParameterId></MacroParameterId>
                                        <ParameterName></ParameterName>
                                        <ParameterValue></ParameterValue>
                                </Parameters>
                        </MacroAlgorithm>
                </Algorithms>
        </MacroInfo>
</GetMacroInfoResponse>
```

Method PutMacro

Description: Accepts a macro definition and either inserts or updates it. Returns a MacroId.

Input xml:

```
<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>
                                        <List>
                                                <Item Value="" Description="" />
                                        </List>
                                </ParameterAttribute>
                        </Parameter>
                </Algorithm>
        </Algorithms>
</MacroInfo>
```

Output xml:

```
<AddNewMacroResponse>
        <AddNewMacroResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </AddNewMacroResult>
        <MacroId>nnnn</MacroId>
</AddNewMacroResponse>
```

Method DeleteMacro

Description: Deletes macro identified by MacroId.

Input xml:

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

Output xml: No output other than standard result

```
<DeleteMacroResponse>
        <DeleteMacroResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </DeleteMacroResult>
</DeleteMacroResponse>
```

Method ListJobs

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

Input xml:

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

Output xml:

```
<ListJobsResponse>
        <ListJobsResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </ListJobsResult>
        <ReturnedCount></ReturnedCount>
        <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>
</ListJobsResponse>
```

Method AddNewJob

Description: Accepts 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).

<u>Input xml</u>:

```
<AddJobInfo>
<ImageId></ImageId>
<InputAnnotationId></InputAnnotationId>
<MacroId></MacroId>
<Priority></Priority>
<CreateMarkup></CreateMarkup>
<InputParametersXML>parameters xml from GetMacroInfo</InputParametersXML>
</AddJobInfo>
<AddJobInfo>
        ...
</AddJobInfo>
```

<u>Output xml</u>:

```
<AddNewJobResponse>
        <AddNewJobResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </AddNewJobResult>
        <JobQueueId></JobQueueId>
        <JobQueueId></JobQueueId>
</AddNewJobResponse>
```

Method CancelJob

Description: Cancels the requested jobs if it's not started or completed. Else the jobs are marked for cancel.

Input xml:

```
<JobId>nnnn</JobId>
<JobId>nnnn</JobId>
<JobId>nnnn</JobId>
…
```

Output xml: No output other than standard result

```
<CancelJobResponse>
        <CancelJobResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </CancelJobResult>
</CancelJobResponse>
```

Generic Data Management Methods

eSlide Manager provides some generic methods to manage data in the database.

Method DeleteRecordData

Description: Accepts token, data table name and id. If user has full access ondata-group, deletes the row. If ValidateOnly is set to 1, then method does not save the record. This is used by Spectrum to check if it is possible to delete the tree before actually deleting it.

****WARNING**** All images will be permanently deleted from ImageServer with a case.

Input xml:

```
<TableName>Name</TableName>
<Id>nnnn</Id>
<ValidateOnly>0 or 1, default 0</ValidateOnly>
```

Output xml: No output other than standard result

```
<DeleteRecordDataResponse>
        <DeleteRecordDataResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </DeleteRecordDataResult>
</DeleteRecordDataResponse>
```

Method DeleteRecordsData

Description: Accepts token, data table name and id. If user has full access on the data-group, it deletes the row.

Input xml:

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

Output xml: No output other than standard result

```
<DeleteRecordsDataResponse>
```

```
        <DeleteRecordsDataResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </DeleteRecordsDataResult>
</DeleteRecordsDataResponse>
```

Method PutRecordData

Description: Accepts a token, data table name, data id and DataRow nodes. If logged on user has full access to the data group that the record belongs to, updates the row. Access is checked for each row. If a row has no data-group id, default data-group is assumed. Operates only on eSlide Manager data tables.

Input xml:

```
<TableName>Name</TableName>
<Id>nnnn</Id>
<DataRow>
        <ClmName1>Clm1Value</ClmName1>
        ...
        <ClmNameN>ClmValueN</ClmNameN>
</DataRow>
```

Output xml: No output other than standard result

```
<PutRecordDataResponse>
        <PutRecordDataResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </PutRecordDataResult>
</PutRecordDataResponse>
```

eSlide Manager Security Methods Request/Response

eSlide Manager security is implemented using Users, Privileges, DataGroups and AccessLevels. Every instance of data in eSlide Manager 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 eSlide Manager 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 eSlide Manager user must logon to eSlide Manager before accessing any data. The following methods allow you to get a token and audit user activity in the system.

Method Logon

Description: Logs user on to the system and returns a token. This token is stored in a cache along with the last access time and user access permissions. Cache class maintains timeout state for these tokens.

Input xml:

PasswordEnc contains encrypted password with the following format !password%time. The time stamp should be within a few seconds of the current time.

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

Output xml:

```
<LogonResponse>
	<LogonResult>
		<ASResult>nnnn</ASResult>
		<ASMessage></ASMessage>
	</LogonResult>
	<token></token>
	<UserData>
		<UserId></UserId>
		<FullName></FullName>
		<LoginName></LoginName>
		<Phone></Phone>
		<E_Mail></E_Mail>
		<UserMustChangePassword>True/False</UserMustChangePassword>
		<Privileges>
			<AdminUser>True/False</AdminUser>
		</Privileges>
		<LastLoginTime>2008-04-15 16:06:45</LastLoginTime>
		<PasswordDaysLeft>-1</PasswordDaysLeft>
		<AutoView>true/false</AutoView>
		<StartPage>SubmitSearch.php?TableName=Case</StartPage>
		<ScanDataGroupId></ScanDataGroupId>
	</UserData>
</LogonResponse>
```

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 eSlide Manager user guide for information regarding enabling/disabling this feature.

Method IsValidToken

Description: Accepts a token and verifies if it is still valid. If Renew = true, then renews the valid token.

Input xml:

```
<Token></Token>
<Renew>true/false</Renew>
```

Output xml:

```
<IsValidTokenResponse>
	<IsValidTokenResult>
		<ASResult>nnnn</ASResult>
		<ASMessage></ASMessage>
	</IsValidTokenResult>
	<Valid>True/False</Valid>
	<TokenExpirationTime></TokenExpirationTime>
</IsValidTokenResponse>
```

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

```
<LogoffResponse>
	<LogoffResult>
		<ASResult>nnnn</ASResult>
		<ASMessage></ASMessage>
	</LogoffResult>
</LogoffResponse>
```

Methods to Manage Users

The following methods allow you to manage eSlide Manager users.

Method AddUsers

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

Input xml:

PasswordEnc contains encrypted password with the following format !password%time. The time stamp should be within a few seconds of the current time.

```
<Token>nnnn</Token>
<UserData>
	<LoginName>abc</LoginName>
	<Password>def</Password>
```

```
        <PasswordEnc></PasswordEnc>
        <FullName>1stName Last Name</FullName>
        <Privileges p1="T/F" p2="T/F">
        <ExternalId></ExternalId>
        <AutoView></AutoView>
</UserData>
```

Output xml:

```
<AddUserResponse>
        <AddUserResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </AddUserResult>
        <UserId>nnnn</UserId>
</AddUserResponse>
```

Method UpdateUser

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

Input xml:

```
<Token>nnnn</Token>
<UserData>
        <UserId>nnnn</UserId>
        <LoginName>abc</LoginName>
        <Password>def</Password>
        <FullName>1stName Last Name</FullName>
        <Privileges p1="T/F" p2="T/F">
        <AutoView></AutoView>
</UserData>
```

Output xml:

```
<UpdateUserResponse>
        <UpdateUserResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </UpdateUserResult>
        <UserId></UserId>
</UpdateUserResponse>
```

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

```
<ChangeUserPasswordResponse>
        <ChangeUserPasswordResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </ChangeUserPasswordResult>
</ChangeUserPasswordResponse>
```

Method DeleteUser

Description: Deletes the specified user from eSlide Manager. If the user is logged on, the token is invalidated.

Input xml:

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

Output xml: No output other than standard result

```
<DeleteUserResponse>
        <DeleteUserResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </DeleteUserResult>
</DeleteUserResponse>
```

Methods to Manage User Access

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

Method AddDataGroup

Description: Accepts a token and data about a DataGroup and adds it to eSlide Manager. 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>nnnn</Token>
<IsPrivate></IsPrivate>
<SecondSlideClientSystemId></SecondSlideClientSystemId>
<DataGroup>
        <Name>xxxx</Name>
        <Description>yyyy</Description>
</DataGroup>
```

Output xml:

```
<AddDataGroupResponse>
        <AddDataGroupResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </AddDataGroupResult>
        <DataGroupId>nnnn</DataGroupId>
</AddDataGroupResponse>
```

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

Output xml: No output other than standard result

```
<UpdateDataGroupResponse>
        <UpdateDataGroupResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </UpdateDataGroupResult>
</UpdateDataGroupResponse>
```

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

```
<DeleteDataGroupResponse>
        <DeleteDataGroupResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </DeleteDataGroupResult>
</DeleteDataGroupResponse>
```

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:

```
<ListAccessByUserResponse>
        <ListAccessByUserResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </ListAccessByUserResult>
</ListAccessByUserResponse>
<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:

```
<ListAccessByDataGroupResponse>
            <ListAccessByDataGroupResult>
                        <ASResult>nnnn</ASResult>
                        <ASMessage></ASMessage>
            </ListAccessByUserResult>
</ListAccessByDataGroupResponse>
<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

```
<UpdateAccessLevelByDataGroupResponse>
    <UpdateAccessLevelByDataGroupResult>
            <ASResult>nnnn</ASResult>
            <ASMessage></ASMessage>
    </UpdateAccessLevelByDataGroupResult>
</UpdateAccessLevelByDataGroupResponse>
```

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

```
<UpdateAccessLevelByUserResponse>
        <updateAccessLevelByUserResult>
                <ASResult>nnnn</ASResult>
                <ASMessage></ASMessage>
        </UpdateAccessLevelByUserResult>
</UpdateAccessLevelByUserResponse>
```

Aperio DataServer Programmer’s Reference
MAN-0066, Revision C