**Changelog:**

|  |  |  |
| --- | --- | --- |
| ***Date*** | ***Author*** | ***Description*** |
| 2010-01-25 | PH | 1. New specification of Web Services implementing the operation on BDR  2. Orderly nomenclature, more accurate description.  3. Comments on open issues. |
| 2010-02-25 | PH | Updating the documentation and separating the specification of the Database's services as a separate document |
| 2010-04-07 | PH | A major update after the completion of the implementation phase |
| 2010-06-30 | PH | Update for Release 1.1 |
| 2010-07-16 | PH | Extended Generic Attribute Metadata Functionality |
| 2010-07-21 | PH | The first version of UPS functionality |
| 2010-07-21 | PH | Extended Attribute Listing Functionality |
| 2010-07-26 | PH | Support for remembering subdirectories of a file |
| 2010-07-30 | PH | AdministrationWS, a new operation in BasicUpdates |
| 2010-08-23 | PH | Cart listing operation and minor corrections |
| 2010-08-29 | PH | Service interfaces after schema change |
| 2010-09-07 | PH | Another schema correction |
| 2010-09-09 | PH | Documentation update for Administration, Authorization, FileStoreman, and UPS |
| 2010-10-11 | PH | Searchable SessionGroup <> Session Operations and Documentation Revisions |
| 2010-10-11 | PH | Correction; added information about updatable static attributes |
| 2010-10-13 | PH | Correction; Newly implemented operations for struct navigation |
| 2010-10-18 | PH | Support for file type attributes |
| 2010-10-21 | PH | Attribute storage type hidden from the service client |
| 2011-09-28 | PH | Copy board and schema changes |
| 2011-10-18 | PH | Log recent changes for refreshing a shallow copy |
| 2011-10-24 | PH | Incremental copy of the board and timestamp for metadata |
| 2012-01-16 | PH | New authentication schemes |
| 2012-02-29 | PH | User Account Management Services |
| 2012-03-29 | PH | Activating and Updating Your Information |
| 2012-12-23 | PH | Password reset functionality. |
| 2014-03-03 | PH | Annotation Status Queries and Updates |
| 2014-03-24 | PH | An operation that lists the groups that the executing user belongs to |
| 2014-03-31 | PH | Updated listing of annotations and IDs in user data |

# Communication interface - functionality

All of the services described below require authentication with the user name and password contained at the message level (i.e. in the header part of the SOAP envelope). The login and password of the user authorized to use BDR, in accordance with the data of the account created for him/her, should be communicated to the customer group appointed to communicate with a given service. This account should not be confused with any PJWSTK account you may have. A requirement for communication with the service is that requests are made in the HTTPS protocol.

## Functionality for Database Querying – BasicQueriesWS

Current definition:

<https://v21.pjwstk.edu.pl/HMDB/res/BasicQueriesWSStandalone.wsdl>

*The following are operations prepared specifically and exclusively for the purpose of communication via Web Services. The signature of these operations is informally represented as an input element (blue) and an output element (green).*

### Navigation structure

The interfaces for editing and viewing data assume working on the hierarchy of the [[1]](#footnote-1) Session > Trial entity. In addition, the motion data structure is made up of the following other entities, but they are no longer hierarchical to each other: Measurement Configuration, Session Group (simple because it has only a name), Performer and Performer Configuration. These 4 consecutive entities no longer form a simple hierarchy. The links between them, together with the number of these connections, are illustrated in the diagram below from the technical documentation.



Circles are used to mark entities that will typically act as "vertices" – i.e. starting points for navigation "inside" the structure. Octagon (Session) – an entity from which we may also start navigation (e.g. by pre-selecting after an attribute). The rest of the entities are marked with a square. The names of the operations that enable navigation are not important from this point of view (they belong rather to the technical documentation).

In addition, there is a Lab entity, not shown in the figure above, which allows, through the associated service operations, to selectively retrieve lists of Sessions created in a given lab, as well as performers appearing in them. It can therefore act as a vertex for navigation, similar to the SessionGroup entity exposed above.

Each uploaded file must be assigned to the parent entity, and currently only 3 entities can play this role: Measurement Configuration, Session and Trial (marked with the letter F in the figure above). Listing files related to a given entity is done using the generic ListFilesWithAttributesXML operation, for which we provide the resource identifier and its type (i.e. one of the 3 supported entities). Therefore, if you want to include more precise information in the database – e.g. that a given file is a video image from the front camera as part of a video measurement, you should upload this file as subject to the appropriate Trial, and in a later step – assign it to the appropriate attribute of this Measurement.

For those entities that are marked with the letter A in the figure above, they can be defined and collected into thematic groups, generic attributes. Once a generic attribute is defined, you can enter its value for each instance of the corresponding entity. Currently, creating new generic attributes and groups of generic attributes is not supported in the user interface provided by the BDR Client. This functionality must be protected from wider access to prevent excessive heterogeneity of data input.

### Generic queries

#### GenericQueryXML(filter : FilterPredicateCollection, entitiesToInclude : string[]) : GenericQueryResult

**<filter> (<FilterPredicate> <PredicateID>int</PredicateID> <ParentPredicate>int</ParentPredicate> <ContextEntity>string</ContextEntity> <PreviousPredicate>int</PreviousPredicate> <NextOperator>string</NextOperator> <FeatureName>string</FeatureName> <Operator>string</Operator> <Value> string</Value> <AggregateFunction>string</AggregateFunction> <AggregateEntity>string</AggregateEntity> </FilterPredicate>)\*  
 </filter>   
<entitiesToInclude>(<string>string</string>)\*</entitiesToInclude>**

**<GenericQueryResult>**

**(<GenericResultRow > <PerformerID>int</PerformerID>?  
 <FirstName>string</FirstName>?  
 <LastName>string</LastName>?  
 <SessionID>int</SessionID>?  
 <UserID>int</UserID>?  
 <LabID>int</LabID>?  
 <MotionKind> string </MotionKind>?  
 <SessionDate>dateTime</SessionDate>?  
 <SessionDescription>string</SessionDescription>?  
 <TrialID>int</TrialID>?  
 <TrialDescription>string</TrialDescription>?  
 <Duration>int</Duration>?  
 <MeasurementID>int</ MeasurementID >?  
 <MeasurementConfID>int</ MeasurementConfID >?  
 <MeasurementConfName>string</ MeasurementConfName >?  
 <MeasurementConfDescription >string</ MeasurementConfDescription >?  
 <EndTime>int</EndTime>?  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*   
</Attributes>)?  
 </ GenericResultRow >)\***

**</ GenericQueryResult >**

The operation takes the structure of a filter expression in which:

* PredicateID is a unique (within the entire structure of the filter parameter) identifier of the predicate;
* ContextEntity specifies the property of which entity the predicate examines – possible values: "performer", "session", "trial", "segment", "file", or "GROUP" – for constructs grouping predicates
* ParentPredicate = parent predicate, grouping other conditions
* NextPredicate = identifier of the next predicate in concrete order
* FeatureName = the name of the property that is examined in the predicate
* Operator = symbol of the operator applied to the property
* Value = possible value used on the right side of the operator (e.g. in comparisons)
* AggregateFunction = the name of the aggregate function, if any, to replace FeatureName in the predicate
* AggregateEntity = the name of the entity to be included in the grouping associated with the aggregate function. If FeatureName is not empty – the aggregate function will receive its name, bound in the context of the AggregateEntity entity.

The entitiesToInclude parameter specifies the list of entities to be included in the join that defines the structure of the operation result, namely: "performer", "session", "trial", "measurement, \_conf".

#### GenericQueryUniformXML(filter : FilterPredicateCollection, entitiesToInclude : string[]) : GenericUniformAttributesQueryResult

**<filter> (<FilterPredicate> <PredicateID>int</PredicateID> <ParentPredicate>int</ParentPredicate> <ContextEntity>string</ContextEntity> <PreviousPredicate>int</PreviousPredicate> <NextOperator>string</NextOperator> <FeatureName>string</FeatureName> <Operator>string</Operator> <Value> string</Value> <AggregateFunction>string</AggregateFunction> <AggregateEntity>string</AggregateEntity> </FilterPredicate>)\*  
 </filter>   
<entitiesToInclude>(<string>string</string>)\*</entitiesToInclude>**

**< GenericUniformAttributesQueryResult>**

**(<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)\***

**</ GenericUniformAttributesQueryResult>**

The operation is a variant of the GenericQueryXML operation described above, but differs from it in that all properties of the described instances – both those derived from static and generic attributes – are specified in a uniform way.

### Search by ID

#### GetPerformerByIdXML(id : int) : PerformerDetailsWithAttributes

**<id>int </id>**

**< PerformerDetailsWithAttributes> <PerformerID>int</PerformerID> <FirstName>string</FirstName> <LastName>string</LastName>  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 </ PerformerDetailsWithAttributes >**

Returns the data of a single performer.

#### GetSessionByIdXML(id : int) : SessionDetailsWithAttributes

**<id>int </id>**

**< SessionDetailsWithAttributes> <SessionID>int</SessionID> <UserID>int</UserID> <LabID>int</LabID> <MotionKind> string </MotionKind> <SessionDate>dateTime</SessionDate>?  
 <SessionDescription>string</SessionDescription>?  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 <SessionLabel>string</SessionLabel>?  
 </ SessionDetailsWithAttributes >**

Returns data for a single session.

#### GetSessionLabel(id : int) : string

**<id>int </id>**

**string**

Returns the label of a single session.

#### GetTrialByIdXML(id : int) :TrialDetailsWithAttributes

**<id>int </id>**

**< TrialDetailsWithAttributes> <TrialID>int</TrialID> <SessionID>int</SessionID> <TrialDescription>string</TrialDescription>  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 </ TrialDetailsWithAttributes >**

Returns data for a single case.

#### GetMeasurementConfigurationByIdXML(id : int) : MeasurementConfDetailsWithAttributes

**<id>int </id>**

**< MeasurementConfDetailsWithAttributes> <MeasurementConfID>int</MeasurementConfID>   
 <MeasurementConfName>string</ MeasurementConfName >   
 <MeasurementConfKind>string</ MeasurementConfKind >   
 <MeasurementConfDescription>string</ MeasurementConfDescription >   
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 </ MeasurementConfDetailsWithAttributes >**

Returns data for a single measurement configuration.

#### GetPerformerConfigurationByIdXML (id : int) : PerformerConfDetailsWithAttributes

**<id>int </id>**

**< PerformerConfDetailsWithAttributes> < PerformerConfID>int</ PerformerConfID >   
 < SessionID >int</ SessionID >   
 < PerformerID >int</ PerformerID >   
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 </ PerformerConfDetailsWithAttributes>**

Returns data for a single performer configuration.

### Performer Enquiries

#### ListPerformersXML() : PerformerList

**-**

**<PerformerList> (< PerformerDetails PerformerID=int FirstName=string LastName=string /> )\*  
</PerformerList>**

Returns a list of all performers.

#### ListPerformersWithAttributesXML() : PerformerWithAttributesList

**-**

**<PerformerWithAttributesList> (< PerformerDetailsWithAttributes> <PerformerID>int</PerformerID> <FirstName>string</FirstName> <LastName>string</LastName>  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 </ PerformerDetailsWithAttributes > )\*  
</PerformerWithAttributesList>**

Returns a list of all performers along with generic attribute values.

#### ListLabPerformersWithAttributesXML(labID : int) : LabPerformerWithAttributesList

**<labID>int </labID>**

**<LabPerformerWithAttributesList> (< PerformerDetailsWithAttributes> <PerformerID>int</PerformerID> <FirstName>string</FirstName> <LastName>string</LastName>  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 </ PerformerDetailsWithAttributes > )\*  
</LabPerformerWithAttributesList>**

Returns a list of performers recording in the specified lab, along with generic attribute values.

#### ListSessionPerformersWithAttributesXML(sessionID : int) : SessionPerformerWithAttributesList

**-**

**< SessionPerformerWithAttributesList> (< PerformerDetailsWithAttributes> <PerformerID>int</PerformerID> <FirstName>string</FirstName> <LastName>string</LastName>  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 </ PerformerDetailsWithAttributes > )\*  
</ SessionPerformerWithAttributesList>**

Returns a list of performers in the specified session, along with the values of their generic attributes.

### Session Queries

#### ListPerformerSessionsXML(performerID : int) : PerformerSessionList

**<performerID>int </performerID>**

**<PerformerSessionList> (< SessionDetails SessionID=int UserID=int LabID=int MotionKind= string SessionDate=dateTime SessionDescription=string SessionLabel = string? /> )\*  
</PerformerSessionList>**

Returns a list of all sessions for a given performer. The score is calculated indirectly based on whether the searched session contains at least one measurement with the participation of a given performer.

#### ListPerformerSessionsWithAttributesXML(performerID : int) : PerformerSessionWithAttributesList

**<performerID>int </performerID>**

**<PerformerSessionWithAttributesList> ( <SessionDetailsWithAttributes> <SessionID>int</SessionID> <UserID>int</UserID> <LabID>int</LabID> <MotionKind> string </MotionKind> <SessionDate>dateTime</SessionDate> <SessionDescription>string</SessionDescription>  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 <SessionLabel>string</SessionLabel>?  
 </ SessionDetailsWithAttributes > )\*  
</PerformerSessionWithAttributesList>**

Returns a list of all sessions of a given performer, along with generic attribute values. The score is calculated indirectly based on whether the searched session contains at least one measurement with the participation of a given performer.

#### ListLabSessionsWithAttributesXML(labID : int) : LabSessionWithAttributesList

**<labID>int </labID>**

**<LabSessionWithAttributesList> ( <SessionDetailsWithAttributes> <SessionID>int</SessionID> <UserID>int</UserID> <LabID>int</LabID> <MotionKind> string </MotionKind> <SessionDate>dateTime</SessionDate> <SessionDescription>string</SessionDescription>  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 <SessionLabel>string</SessionLabel>?  
 </ SessionDetailsWithAttributes > )\*  
</LabSessionWithAttributesList>**

Returns a list of sessions originating from the specified lab, along with generic attribute values.

#### ListGroupSessionsWithAttributesXML(sessionGroupID : int) : GroupSessionWithAttributesList

**<sessionGroupID>int </sessionGroupID>**

**< GroupSessionWithAttributesList> ( <SessionDetailsWithAttributes> <SessionID>int</SessionID> <UserID>int</UserID> <LabID>int</LabID> <MotionKind> string </MotionKind> <SessionDate>dateTime</SessionDate> <SessionDescription>string</SessionDescription>  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 <SessionLabel>string</SessionLabel>?   
</ SessionDetailsWithAttributes > )\*  
</ GroupSessionWithAttributesList>**

Returns a list of all sessions belonging to the Session Group specified by the identifier, along with the values of the generic attributes.

#### ListMeasurementConfSessionsWithAttributesXML (measurementConfID: int) : MeasurementConfSessionWithAttributesList

**<measurementConfID >int </ measurementConfID >**

**< MeasurementConfSessionWithAttributesList > ( <SessionDetailsWithAttributes> <SessionID>int</SessionID> <UserID>int</UserID> <LabID>int</LabID> <MotionKind> string </MotionKind> <SessionDate>dateTime</SessionDate> <SessionDescription>string</SessionDescription>  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 <SessionLabel>string</SessionLabel>?   
</ SessionDetailsWithAttributes > )\*  
</ MeasurementConfSessionWithAttributesList >**

Returns a list of all sessions belonging to the Session Group specified by the identifier, along with the values of the generic attributes.

### Session Group Queries

#### ListSessionSessionGroups (sessionID : int) : SessionGroupDefinitionList

Returns a list of Session Groups to which the session indicated by the ID is assigned.

**< sessionID >int</ sessionID >**

**<SessionGroupDefinitionList xmlns="http://ruch.bytom.pjwstk.edu.pl/MotionDB/BasicQueriesService">   
( <SessionGroupDefinition> <SessionGroupID>int</ SessionGroupID > <SessionGroupName>string</ SessionGroupName ></ SessionGroupDefinition > )\*  
</SessionGroupDefinitionList >**

### Trial enquiries

#### ListSessionTrialsXML(sessionID : int) : SessionTrialList

**<sessionID>int </sessionID>**

**<SessionTrialList> (< TrialDetails TrialID=int SessionID=int TrialDescription=string / > )\*  
</SessionTrialList>**

Returns a list of all observations for a given session.

#### ListSessionTrialsWithAttributesXML(sessionID : int) : SessionTrialWithAttributesList

**<sessionID>int </sessionID>**

**<SessionTrialWithAttributesList> ( < TrialDetailsWithAttributes >  
 <TrialID>int</TrialID> <SessionID>int</SessionID> <TrialDescription>string</TrialDescription>  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 <SessionLabel>string</SessionLabel>?  
 </ TrialDetailsWithAttributes > )\*  
</SessionTrialWithAttributesList>**

Returns a list of all observations for a given session, along with generic attribute values.

### Queries for measurement configurations

#### ListMeasurementConfigurationsWithAttributesXML () : MeasurementConfWithAttributesList

**-**

**<MeasurementConfWithAttributesList>  
 (< MeasurementConfDetailsWithAttributes> <MeasurementConfID>int</MeasurementConfID>   
 <MeasurementConfName>string</ MeasurementConfName >   
 <MeasurementConfKind>string</ MeasurementConfKind >   
 <MeasurementConfDescription>string</ MeasurementConfDescription >   
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 </ MeasurementConfDetailsWithAttributes > )\*  
</ MeasurementConfWithAttributesList>**

Returns a list of all defined survey configurations.

### Performer configuration queries

#### ListSessionPerformerConfsWithAttributesXML(sessionID : int) : PerformerConfWithAttributesList

**<sessionID>int </sessionID >**

**<SessionPerformerConfWithAttributesList>  
 ( < PerformerConfDetailsWithAttributes> <PerformerConfID>int</PerformerConfID>   
 <SessionID>int</ SessionID >   
 < PerformerID >int</ PerformerID >   
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 </ PerformerConfDetailsWithAttributes> )\*  
</ SessionPerformerConfWithAttributesList >**

Returns a list of performer configurations present in a given session.

### File Enquiries

#### ListFilesXML(subjectID : int, subjectType : string) : FileList

**<subjectID>int</subjectID> <subjectType>string</subjectType>**

**<FileList> (< FileDetails FileID=int FileName=string FileDescription=string SubdirPath=string? AttributeName=string / > )\*  
</FileList>**

Returns a list of files assigned to the specified instance, indicated by the subjectID parameter of the entity instance specified by the subjectType ("session", "trial"," measurement \_conf") parameter. SubdirPath will occur if the file was stored in a multicast write operation, specifying the subdirectory that originally contained it.

#### ListFilesWithAttributesXML(subjectID : int, subjectType : string) : FileWithAttributesList

**<subjectID>int</subjectID> <subjectType>string</subjectType>**

**<FileWithAttributesList> ( <FileDetailsWithAttributes   
FileID=int FileName=string FileDescription=string SubdirPath=string? AttributeName=string >  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 </ FileDetailsWithAttributes > )\*  
</FileWithAttributesList>**

Returns a list of files assigned to the specified instance, indicated by the subjectID parameter of the entity instance specified by the subjectType parameter ( "performer", "session", "trial", "measurement", " measurement \_conf") along with the values of generic attributes. SubdirPath will occur if the file was stored in a multicast write operation, specifying the subdirectory that originally contained it.

#### ListFileAttributeDataXML(subjectID : int, subjectType : string) : FileList

**<subjectID>int</subjectID> <subjectType>string</subjectType>**

**<FileList> (< FileDetails FileID=int FileName=string FileDescription=string SubdirPath=string? AttributeName=string / > )\*  
</FileList>**

Returns a list of files assigned as file type values to generic attributes specified by the subjectID parameter of the entity instance specified by subjectType ("performer", "session", "trial", "measurement", " measurement \_conf", " performer \_conf"). SubdirPath will occur if the file was stored in a multicast write operation, specifying the subdirectory that originally contained it.

#### ListFileAttributeDataWithAttributesXML(subjectID : int, subjectType : string) : FileWithAttributesList

**<subjectID>int</subjectID> <subjectType>string</subjectType>**

**<FileWithAttributesList> ( <FileDetailsWithAttributes   
FileID=int FileName=string FileDescription=string SubdirPath=string? AttributeName=string >  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 </ FileDetailsWithAttributes > )\*  
</FileWithAttributesList>**

Returns a list of files assigned as file type values to the generic attributes specified by the subjectID parameter of the subjectType entity instance specified by the subjectType parameter (" "performer", "session", "trial", "measurement", " measurement \_conf", " performer \_conf") along with the values of the generic attributes. SubdirPath will occur if the file was stored in a multicast write operation, specifying the subdirectory that originally contained it.

### Annotation queries

#### ListMyAnnotationsXML () : UserAnnotations

**-**

**<UserAnnotations> (< Annotation TrialID=int UserID=int Status=int Comment=string Note=string / > )\*  
</UserAnnotations>**

Returns a list of annotations made by the user performing the operation call.

#### ListAwaitingAnnotationsXML () : UserAnnotations

**-**

**<AwaitingAnnotations> (< Annotation TrialID=int UserID=int Status=int Comment=string Note=string / > )\*  
</AwaitingAnnotations>**

Returns a list of annotations that have been completed by the author and are awaiting review by the reviewer.

#### ListMyReviewedAnnotationsXML () : UserAnnotations

**-**

**<ReviewedAnnotations> (< Annotation TrialID=int UserID=int Status=int Comment=string Note=string / > )\*  
</ReviewedAnnotations>**

Returns a list of annotations retrieved for evaluation, whose evaluator is the user performing the operation call.

#### ListAnnotationsXML () : UserAnnotations

**-**

**<Annotations> (< Annotation TrialID=int UserID=int Status=int Comment=string Note=string / > )\*  
</Annotations>**

Returns a list of all annotations present in the system, regardless of their status.

#### ListCompletedAnnotationsXML () : UserAnnotations

**-**

**<CompletedAnnotations> (< Annotation TrialID=int UserID=int Status=int Comment=string Note=string / > )\*  
</CompletedAnnotations>**

Returns a list of all accepted annotations (that is, those with status = 4).

### Queries related to the date of the last update

#### GetDBTimestamp () : DateTime

Returns the date of the last modification of the contents of the database scoped by the shallow copy or the files whose descriptions are listed in the shallow copy. If you determine that the database has changed from the version stored on the client side, you can use the operation to download an incremental shallow copy: **GetShallowCopyIncrement(since : DateTime) : string**.

#### GetMetadataTimestamp () : DateTime

Returns the date when the metadata scoped to the shallow copy was last modified. If modifications occur to the set of metadata stored in the client, you must retrieve a fresh set of metadata: **GetMetadata() : string** (Incremental fetch is not available for metadata.)

### Metadata operations

#### ListAttributesDefined(attributeGroupName : string, entityKind : string) : AttributeDefinitionList

Returns definitions of attributes defined for a given entity type – all or only those that belong to the specified Attribute Group.

**< attributeGroupName >string</ attributeGroupName >   
< entityKind >string</ entityKind >**

**<AttributeDefinitionList xmlns="http://ruch.bytom.pjwstk.edu.pl/MotionDB/BasicQueriesService">   
( <AttributeDefinition> <AttributeName>string</ attributeName > <AttributeType>string</ attributeType> <AttributeEnum>int</ attributeEnum>?  
 <AttributeGroupName>string</ attributeGroupName >  
 <Unit>string</ Unit >?  
 (<EnumValues>  
 <Value>string</Value>\*  
 </ EnumValues >)?  
</AttributeDefinition> )\*  
</AttributeDefinitionList >**

Acceptable values for the entityKind argument are:

* Performer
* session,
* Trial
* measurement\_conf,
* file.

You can specify the special name "\_ALL" in place of the attribute group name to return the definitions of all attributes. Similarly, you can also specify the special name "\_ALL" as the entity type, and all attribute definitions will be returned.

The current set of reconciled values for the AttributeType field are:

Int

Decimal

nonNegativeInteger

nonNegativeDecimal

dateTime

date

TIMECODE

Float

#### ListAttributeGroupsDefined(entityKind : string) : AttributeGroupDefinitionList

Returns definitions of attributes defined for a given entity type – all or only those that belong to the specified Attribute Group.

**< entityKind >string</ entityKind >**

**<AttributeGroupDefinitionList xmlns="http://ruch.bytom.pjwstk.edu.pl/MotionDB/BasicQueriesService">   
( <AttributeGroupDefinition> <AttributeGroupName>string</ attributeGroupName > <DescribedEntity>string</ DescribedEntity ></AttributeGroupDefinition> )\*  
</AttributeGroupDefinitionList >**

Acceptable values for the entityKind argument are:

* Performer
* session,
* Trial
* measurement\_conf,
* file.

You can specify the special name "\_ALL\_ENTITIES" or simply "\_ALL" in place of the entity type name to list the attribute groups for all entities.

#### ListSessionGroupsDefined() : SessionGroupDefinitionList

Returns a list of all system-defined Session Groups.

**-**

**<SessionGroupDefinitionList xmlns="http://ruch.bytom.pjwstk.edu.pl/MotionDB/BasicQueriesService">   
( <SessionGroupDefinition> <SessionGroupID>int</ SessionGroupID > <SessionGroupName>string</ SessionGroupName ></ SessionGroupDefinition > )\*  
</SessionGroupDefinitionList >**

#### ListMotionKindsDefined() : MotionKindDefinitionList

Returns definitions of attributes defined for a given entity type – all or only those that belong to the specified Attribute Group.

**-**

**<MotionKindDefinitionList xmlns="http://ruch.bytom.pjwstk.edu.pl/MotionDB/BasicQueriesService">   
( <MotionKindDefinition> <MotionKindID>int</ MotionKindID > <MotionKindName>string</ MotionKindName ></ MotionKindDefinition > )\*  
</MotionKindDefinitionList >**

#### ListEnumValues(attributeName : string, entityKind : string) : EnumValueList

Returns a list of enumeration values defined for a given attribute.

**< attributeName >string</ attributeName >   
< entityKind >string</ entityKind >**

**<EnumValueList xmlns="http://ruch.bytom.pjwstk.edu.pl/MotionDB/BasicQueriesService">   
( <EnumValue> string </ EnumValue > )\*  
</ EnumValueList >**

## Functionality for updating the Database component – BasicUpdatesWS

Current definition: <https://v21.pjwstk.edu.pl/HMDB/BasicUpdatesWS.svc?wsdl>

All operations from this service can communicate an UpdateException, which has IssueKind and Details properties.

### Creating an Instance

#### CreatePerformer (performerData : PerformerData) : int

Used to manually create a performer entity instance based on the data entered by the user into the client software. The result is the ID of the newly created performer. The current structure of the parameter is given below.

**<performerData><Name>string</Name><Surname>string</Surname></performerData>**

**Int**

#### CreateSession (labID : int, motionKindName : string, sessionDate : DateTime, sessionDescription : string, sessionGroutIDs : int[]) : int

Used when manually creating a session instance by the user. The result is the ID of the newly created session. The user ID of the user calling the session is recovered from the context of the operation call.

#### CreateTrial (sessionID : int, trialDescription : string) : int

Used for manual creation of an observation instance (trial) by the user. The result is the ID of the newly created observation.

#### CreateMeasurementConfiguration (mcName : string, mcKind : string, mcDescription : string) : int

Used when manually creating an instance of the measurement configuration (measurement\_configuration) by the user. The result is the ID of the newly created configuration.

### Grouping and assigning

#### AssignSessionToGroup(sessionID : int, performerID : int) : int

Allows you to include an existing session in the database to one of the previously defined groups. The result is a Boolean value that tells you the success of the operation.

#### AssignPerformerToSession(sessionID : int, groupID : int) : int

Allows you to attach the specified performer to the selected session. This attribution is rough, as we also record the fact of the performer's participation in individual measurements. At this level, the PerformerConfiguration entity is created and its ID is returned as a result.

#### AddPerformerToMeasurement(sessionID : int, measurementID : int) : bool

Allows you to attach the specified performer to the specified measurement. The result is a Boolean value that tells you the success of the operation.

### Editing Generic Attributes

#### SetPerformerAttribute(performerID : int, attributeName : string, attributeValue : string, update : bool)

It allows you to place additional descriptive data related to the specified performer in the database. A pre-defined attribute (in terms of name and type) is connected to the described performer by the value that characterizes the performer. The update parameter specifies whether the attribute value for the given performer is to be overwritten if the presence of an attribute value is detected.

The operation also allows you to update static attributes (but omit identifiers and most foreign keys). Updates to the following static attributes are supported for the Performer entity:

'FirstName', 'string', 'shortString'

'LastName', 'string', 'shortString'

Since we allow three types of such generic attributes: integer[[2]](#footnote-2), float and string, this operation, in the case of the first two mentioned types, will verify the possibility of converting the attributeValue parameter to the appropriate number.

Similarly, for attributes with values of an enumeration type, the operation should include checking whether the entered value belongs to a predefined set of values for that attribute.

#### SetSessionAttribute(sessionID : int, attributeName : string, attributeValue : string, update : bool)

It allows you to place additional descriptive data related to the specified session in the database. A previously defined (in terms of name and type) attribute is linked to the session described by the value that characterizes the session. The update parameter specifies whether the attribute value is to be overwritten if it detects the presence of an attribute value for a given session.

The operation also allows you to update static attributes (but omit identifiers and most foreign keys). Updates to the following static attributes are supported for the Session entity:

'LabID', 'int', 'ID'

'MotionKind', 'string', 'shortString'

'SessionDate', 'string', 'dateTime'

'SessionDescription', 'string', 'longString'

Since we allow three types of such generic attributes: integer, float and string, this operation, in the case of the first two mentioned types, will verify the possibility of converting the attributeValue parameter to the appropriate number.

Similarly, for attributes with values of an enumeration type, the operation should include checking whether the entered value belongs to a predefined set of values for that attribute.

#### SetTrialAttribute(trialID : int, attributeName : string, attributeValue : string, update : bool)

It allows you to place additional descriptive data related to the indicated observation in the database. A previously defined (in terms of name and type) attribute is linked to the described observation by the value that characterizes the observation. The update parameter specifies whether the attribute value is to be overwritten if it detects the presence of an attribute value for a given observation.

The operation also allows you to update static attributes (but omit identifiers and most foreign keys). Updates to the following static attributes are supported for the Trial entity:

'TrialDescription', 'string', 'longString'

Since we allow three types of such generic attributes: integer, float and string, this operation, in the case of the first two mentioned types, will verify the possibility of converting the attributeValue parameter to the appropriate number.

Similarly, for attributes with values of an enumeration type, the operation should include checking whether the entered value belongs to a predefined set of values for that attribute.

#### SetMeasurementConfAttribute (measurementConfID: int, attributeName : string, attributeValue : string, update : bool)

It allows you to place additional descriptive data related to the indicated measurement configuration in the database. A previously defined (in terms of name and type) attribute is linked to the described observation by the value that characterizes this configuration. The update parameter specifies whether the attribute value is to be overwritten if the presence of an attribute value is detected for the given configuration.

The operation also allows you to update static attributes (but omit identifiers and most foreign keys). Updates to the following static attributes are supported for the MeasurementConf entity:

-

Since we allow three types of such generic attributes: integer, float and string, this operation, in the case of the first two mentioned types, will verify the possibility of converting the attributeValue parameter to the appropriate number.

Similarly, for attributes with values of an enumeration type, the operation should include checking whether the entered value belongs to a predefined set of values for that attribute.

#### SetPerformerConfAttribute (performerConfID: int, attributeName : string, attributeValue : string, update : bool)

It allows you to place additional descriptive data related to the indicated performer configuration in the database. A previously defined (in terms of name and type) attribute is linked to the described observation by the value that characterizes this configuration. The update parameter specifies whether the attribute value is to be overwritten if the presence of an attribute value is detected for the given configuration.

The operation also allows you to update static attributes (but omit identifiers and most foreign keys). Updates to the following static attributes are supported for the MeasurementConf entity:

'MeasurementConfName', 'string', 'shortString'

'MeasurementConfKind', 'string', 'shortString'

'MeasurementConfDescription', 'string', 'longString'

Since we allow three types of such generic attributes: integer, float and string, this operation, in the case of the first two mentioned types, will verify the possibility of converting the attributeValue parameter to the appropriate number.

Similarly, for attributes with values of an enumeration type, the operation should include checking whether the entered value belongs to a predefined set of values for that attribute.

#### SetFileAttribute(fileID : int, attributeName : string, attributeValue : string, update : bool)

Allows you to place additional descriptive data related to the specified file in the database. A previously defined (in terms of name and type) attribute is linked to the described file with the value that characterizes the file. The update parameter specifies whether the attribute value is to be overwritten if it detects the presence of an attribute value for the specified file.

The operation also allows you to update static attributes (but omit identifiers and most foreign keys). Updates to the following static attributes are supported for the File entity:

'FileName', 'string', 'shortString'

'FileDescription', 'string', 'longString'

'SubdirPath', 'string', 'shortString'

Since we allow three types of such generic attributes: integer, float and string, this operation, in the case of the first two mentioned types, will verify the possibility of converting the attributeValue parameter to the appropriate number.

Similarly, for attributes with values of an enumeration type, the operation should include checking whether the entered value belongs to a predefined set of values for that attribute.

#### SetFileTypedAttributeValue(resourceID : int, entity : string, attributeName : string, fileID : int, update : bool)

Allows you to place additional descriptive data in the database related to the specified entity and specified by the generic attribute with the specified name. The previously defined (in terms of name and type) attribute is connected to the described entity instance with a value, which in the case of this operation is a file. The update parameter specifies whether the attribute value is to be overwritten if it detects the presence of an attribute value for the specified file.

The entity parameter values supported by the operation are:

* Performer
* session
* Trial
* measurement
* measurement\_conf
* performer\_conf

#### ClearAttributeValue(resourceID : int, attributeName : string, entity : string)

An operation that allows you to delete previously set generic attribute values for specified resources. The identification of the value to be deleted is based on the indication of:

* Id
* Name of the entity to which the order applies – acceptable values: performer | session | trial | measurement | measurement\_conf | file | performer\_conf,
* attribute name.

Due to the gradual introduction of authorization mechanisms, an order may be rejected if the user does not have permission to access a given entity or a given group of attributes.

### Update the status of an annotation

Annotations can change their state according to the state diagram shown by invoking one of the following two operations.



For startup, 0, and 1 states, calls to SetMyAnnotationStatus by the annotation author are allowed. For states 2 and 3, calls to the SetAnnotationReview operation are allowed (for any eligible reviewer and for the reviewer who downloaded the annotation for the review, respectively).

#### SetMyAnnotationStatus (trialID : int, status : int, comment : string)

An operation available to the author of the annotation. Declares the creation of a new annotation object, or updates the state of an existing one. Allowed state changes – as shown in the diagram above – only from initial states of none, 1 or 0. Specifying an empty string as the comment parameter will preserve the existing value of the comment.

#### SetAnnotationReview (trialID : int, userID : int, status : int, note : string)

An operation available to the annotation reviewer. Declares that a reviewer has started a review, or records the result of the review. Allowed state changes – as shown in the diagram above – only from states 2 or 3. Specifying an empty string as the note parameter will preserve the existing value of the comment.

## Server File Transport Component - FileStoremanWS

Current definition: <https://v21.pjwstk.edu.pl/HMDB/FileStoremanWS.svc?wsdl>

All operations from this service can communicate a FileAccessServiceException, which has IssueKind and Details properties.

All file save operations include, after successfully writing resources to the database, deleting the disk directory containing them, which is described by a relative path in the path parameter.

#### StoreMeasurementConfFile (mcID : int, path : string, description : string, filename : string) : int

It implements the introduction of a single measurement configuration file under the control of the database. The path parameter contains a path relative to the root of the File Transport Services area on the server's disk, without the file name itself. The file name is specified in the filename parameter. Once the file is successfully uploaded, the service removes it from the upload area. The result is the file ID assigned to it as part of the File table in the database. The value of the filename parameter cannot exceed 255 characters.

Prerequisite:

1. A measurement configuration with a given identifier has already been created in the database.

2. The file has been successfully placed on the server's disk via the File Transport Service.

#### StoreSessionFile (sessionID : int, path : string, description : string, filename : string) : int

Brings a single session file under the control of the database. The path parameter contains a path relative to the root of the File Transport Services area on the server's disk, without the file name itself. The file name is specified in the filename parameter. Once the file is successfully uploaded, the service removes it from the upload area. The result is the file ID assigned to it as part of the File table in the database. The value of the filename parameter cannot exceed 255 characters.

Prerequisite:

1. A session with the specified ID has already been created in the database.

2. The file has been successfully placed on the server's disk via the File Transport Service.

#### StoreTrialFile (trialID : int, path : string, description : string, filename : string) : int

It realizes the introduction of a single observation file under the control of the database. The path parameter contains a path relative to the root of the File Transport Services area on the server's disk, without the file name itself. The file name is specified in the filename parameter. Once the file is successfully uploaded, the service removes it from the upload area. The result is the file ID assigned to it as part of the File table in the database. The value of the filename parameter cannot exceed 255 characters.

Prerequisite:

1. An observation with a given ID has already been created in the database.

2. The file has been successfully placed on the server's disk via the File Transport Service.

#### StoreMeasurementConfFiles (performerID : int, path : string, description : string)

Brings files under the control of the database. Applies to the files that specify the measurement configuration. The path parameter specifies a relative path, in the upload area, to the directory containing the files to be uploaded. After successfully uploading the files, the service removes them from the upload area. Any relative path (including the file name) deep from the location specified in the path parameter cannot be longer than 255 characters.

#### StoreSessionFiles (sessionID : int, path : string, description : string)

Brings files under the control of the database. Applies to files that describe the session as a whole. The path parameter specifies a relative path, in the upload area, to the directory containing the files to be uploaded. After successfully uploading the files, the service removes them from the upload area. After successfully uploading the files, the service removes them from the upload area. Any relative path (including the file name) deep from the location specified in the path parameter cannot be longer than 255 characters.

#### StoreTrialFiles (trialID : int, path : string, description : string)

Brings files under the control of the database. Applies to files that describe a single case. The path parameter specifies a relative path, in the upload area, to the directory containing the files to be uploaded. After successfully uploading the files, the service removes them from the upload area. Any relative path (including the file name) deep from the location specified in the path parameter cannot be longer than 255 characters.

#### RetrieveFile (fileID : int) : FileData

The service is supposed to extract the given file from the database and make it available to the File Transport Service in the location described by the URI.

**< fileID >int</ fileID >**

**<FileData xmlns="http://ruch.bytom.pjwstk.edu.pl/MotionDB/FileStoremanService"> <FileLocation> string </ FileLocation >   
<SubdirPath> string </ SubdirPath ></ FileData >**

The structure of the result contains in the FileLocation field a path relative to the root of the File Transport Service repository, along with the file name, which is restored to the same form as the name of the originally uploaded file. The SubdirPath field will appear with non-empty content if the file was saved in a multiwrite operation, specifying the original subdirectory containing it.

#### DownloadComplete (fileID : int, path : string)

Confirms that the download is complete. It also serves as a signal for the service to clean up the file. The ID of the downloaded file and the path parameter are passed, which contains the same information that was previously received by the client as a result of calling the RetrieveFile operation in the FileLocation field.

If for some reason the client program does not send such a signal, the file will be deleted after 24 hours (ToDo).

#### GetShallowCopy() : string

The service returns a relative path (relative to the root of the FTP area) to the generated file with a shallow copy of the database, including the subdirectory and the name of the file itself. The XML file can then be downloaded via FTPS. A diagram of the shallow copy file is available at:

[https://v21.pjwstk.edu.pl/HMDB/res/ShallowCopy.xsd](http://v21.pjwstk.edu.pl/Motion/res/ShallowCopy.xsd)

#### GetShallowCopyIncrement(since : DateTime) : string

The service returns a relative path (relative to the root of the FTP area) to the generated file with a shallow copy of the database (limited, however, to only those data instances that have changed after the date specified in the parameter), including the subdirectory and the name of the file itself. The XML file can then be downloaded via FTPS. A diagram of the shallow copy file is available at:

[https://v21.pjwstk.edu.pl/HMDB/res/ShallowCopy.xsd](http://v21.pjwstk.edu.pl/Motion/res/ShallowCopy.xsd)

#### GetMetadata() : string

The service returns a relative path (relative to the root of the FTP area) to the generated metadata file (including the names of session groups, motion types, labs, as well as the names of attribute groups and attribute definitions belonging to these groups – where applicable, listing the allowed enumeration values for it), including the subdirectory and the name of the file itself. The XML file can then be downloaded via FTPS. A diagram of the shallow copy file is available at:

[https://v21.pjwstk.edu.pl/HMDB/res/Metadata.xsd](http://v21.pjwstk.edu.pl/Motion/res/Metadata.xsd)

## Functionality of the private user area – UserPersonalSpaceWS

All of the operations described below identify the area of the appropriate user based on the authentication data reachable from the context of the operation call.

Current definition:

<https://v21.pjwstk.edu.pl/HMDB/res/UserPersonalSpaceWSStandalone.wsdl>

### Saving and retrieving defined filters

#### UpdateStoredFilters (filter : FilterPredicateCollection )

**<filter> (<FilterPredicate> <PredicateID>int</PredicateID> <ParentPredicate>int</ParentPredicate> <ContextEntity>string</ContextEntity> <PreviousPredicate>int</PreviousPredicate> <NextOperator>string</NextOperator> <FeatureName>string</FeatureName> <Operator>string</Operator> <Value> string</Value> <AggregateFunction>string</AggregateFunction> <AggregateEntity>string</AggregateEntity> </FilterPredicate>)\*  
 </filter>**

**-**

Allows you to save a set of defined filters in your Custom Area.

#### ListStoredFilters (filter : FilterPredicateCollection )

-

**<FilterList> (<FilterPredicate> <PredicateID>int</PredicateID> <ParentPredicate>int</ParentPredicate> <ContextEntity>string</ContextEntity> <PreviousPredicate>int</PreviousPredicate> <NextOperator>string</NextOperator> <FeatureName>string</FeatureName> <Operator>string</Operator> <Value>string</Value> <AggregateFunction>string</AggregateFunction>  
 <AggregateEntity>string</AggregateEntity> </FilterPredicate>)\* </ FilterList >**

Allows you to save a set of defined filters in your Custom Area.

### Creating and updating a Shopping Cart

#### CreateBasket (basketName : string)

Creates a new Shopping Cart with the specified name for the current user.

#### RemoveBasket (basketName : string)

Deletes the Cart for the current user.

#### ListUserBaskets()

-

**<BasketDefinitionList> (<BasketDefinition> <BasketName>string</ BasketName > </ BasketDefinition >)\* </ BasketDefinitionList >**

Returns the names of the current user's existing shopping carts.

#### AddEntityToBasket (basketName : string, resourceID : int, entity : string )

Saves the specified resource in a basket with the specified name (the cart must exist beforehand). Depending on the value of the entity parameter, the resource ID is identified as a foreign key from the corresponding table. Currently, it is supported to store the following types of entities in the cart:

* Performer
* session,
* Trial.

#### RemoveEntityFromBasket (basketName : string, resourceID : int, entity : string )

Removes the specified resource from the basket with the specified name (the cart must exist beforehand). Depending on the value of the entity parameter, the resource ID is identified as a foreign key from the corresponding table. Currently, it is supported to store the following types of entities in the cart:

* Performer
* session,
* Trial.

### Checking out the contents of the Shopping Cart

#### ListBasketPerformersWithAttributesXML(basketName : string) : BasketPerformerWithAttributesList

**<basketName>string </ basketName >**

**<BasketPerformerWithAttributesList> (< PerformerDetailsWithAttributes> <PerformerID>int</PerformerID> <FirstName>string</FirstName> <LastName>string</LastName>  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 </ PerformerDetailsWithAttributes > )\*  
</BasketPerformerWithAttributesList>**

Returns a list of performers previously placed in the basket name.

#### ListBasketSessionsWithAttributesXML(basketName : string) : BasketSessionWithAttributesList

**<basketName>string </ basketName >**

**<BasketSessionWithAttributesList> ( <SessionDetailsWithAttributes> <SessionID>int</SessionID> <UserID>int</UserID> <LabID>int</LabID> <MotionKind> string </MotionKind> <SessionDate>dateTime</SessionDate> <SessionDescription>string</SessionDescription>  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 <SessionLabel>string</SessionLabel>?  
 </ SessionDetailsWithAttributes > )\*  
</BasketSessionWithAttributesList>**

Returns a list of sessions previously placed in the specified cart name.

#### ListBasketTrialsWithAttributesXML(basketName : string) : SessionTrialWithAttributesList

**<basketName>string </ basketName >**

**<BasketTrialWithAttributesList> ( < TrialDetailsWithAttributes > <TrialID>int</TrialID> <SessionID>int</SessionID> <TrialDescription>string</TrialDescription> <Duration>int</Duration>  
 (<Attributes>  
 (<Attribute  
 Name = string  
 Value = string  
 entity=string?  
 AttributeGroup = string  
 Type = string  
 / > )\*  
 </Attributes>)?  
 <SessionLabel>string</SessionLabel>?  
 </ TrialDetailsWithAttributes> )\*  
</BasketTrialWithAttributesList>**

Returns a list of all cases previously placed in the specified basket name.

## Functionality for configuration management – including a set of metadata – AdministrationWS

Current definition: <https://v21.pjwstk.edu.pl/HMDB/AdministrationWS.svc?wsdl>

All operations from this service can communicate an AdministrationOperationException, which has IssueKind and Details properties.

The entity argument must be set as: performer | session | trial | measurement\_conf | performer\_conf | file

The authorization mechanisms require that the principal of the following operations belongs to the MotionOperators group.

#### DefineAttributeGroup(groupName : string, entity : string)

This operation allows you to define a new, empty group of attributes, dedicated to grouping attributes describing the specified entity.

An operation can signal errors when:

* The attribute group name you entered is already in use.
* Incorrect entity name was provided.

#### RemoveAttributeGroup(groupName : string, entity : string)

This action allows you to delete a previously defined group of attributes. A group is identified by a combination of its name and an entity name that is described by the group's attributes. The operation throws an error if a group with the given name and the described entity is not available in the database. If the group had attributes assigned to it – they will be cascaded.

#### DefineAttribute(attributeName : string, groupName : string, entity : string, isEnum : boolean, pluginDescriptor : string, type : string, unit : string)

This operation allows you to define a new generic attribute, assigned to exactly one predefined group. The attributeName parameter specifies the desired name of the attribute to be created, while the groupName and entity parameters identify the attribute group to which it should belong.

The type argument specifies the type of data to be stored as an attribute value. Acceptable values: "int", "decimal", "nonNegativeInteger", "nonNegativeDecimal", "dateTime", "date", "TIMECODE", "float".

The isEnum and unit arguments, respectively, determine whether an attribute has enumerated values and, if applicable, the units in which its value is expressed.

The pluginDescriptor argument will specify the metadata of the development component that contains the analytic attribute logic.

An operation can signal errors when:

* An attribute with the given name, entity and group already exists.
* the group indicated in groupName does not exist,
* A data type outside of the above set is specified.

#### RemoveAttribute(attributeName : string, groupName : string, entity : string)

Deletes a previously defined attribute. If it had values defined for any instances, they will be cascaded. The operation throws an error if the attribute with the given name, group, and described entity does not exist in the database.

#### AddAttributeEnumValue(attributeName : string, groupName : string, entity : string, value : string, clearExisting : boolean)

This operation allows you to add (one at a time) new enumeration values to a previously defined attribute, identified by a combination of attributeName, groupName, entity parameters.

If clearExisting = true, previously entered enumeration values are removed. If we specify an empty string as value, the value will not be added.

Deletes a previously defined attribute. If it had values defined for any instances, they will be cascaded. The operation throws an error if an attribute with the specified name, group, and entity does not exist in the database, or if an attempt was made to enter a value for an attribute that was not defined as having enumerated values.

#### DownloadAreaCleanup(olderThanMinutes : int)

The operation allows you to remove wastelands in the form of files put up for download, for which the completion of the download has not been confirmed. The olderThanMinutes parameter indicates the time in minutes above which the file should be considered waste.

## Funkcjonalność związana z autoryzacją – AuthorizationWS

Current definition: [https://v21.pjwstk.edu.pl/HMDB/res/AuthorizationWSStandalone.wsdl](https://v21.pjwstk.edu.pl/HMDB/AuthorizationWS.svc?wsdl)

All operations from this service can communicate an AuthorizationException that has IssueKind and Details properties.

### User management

#### ListUsers() : UserList

**-**

**<UserList> ( < UserDetails >  
 <ID>int</ID >  
 <Login>string</ Login > <FirstName>string</FirstName> <LastName>string</LastName> </ UserDetails > )\*  
</BasketTrialWithAttributesList>**

The operation returns a list of all defined users in the database. Requires caller authorization as a member of the motion\_administrators group.

#### GetMyUserData() : UserData

**-**

**<ID>int</ID >  
 <Login>string</ Login > <FirstName>string</FirstName> <LastName>string</LastName>  
 <Email>string</Email>**

The operation returns the data of the user authenticating the request to call this operation. This will allow the client application to populate the forms used to update user data through the Web services interface with current data.

#### ListMyUserGroupsAssigned () : MyUserGroupsAssigned

**-**

**<MyUserGroupsAssigned> ( < UserGroup> <UserGroupID>int</ UserGroupID> <UserGroupName>string</UserGroupName> </ UserGroup> )\*  
</MyUserGroupsAssigned>**

The operation returns the names and IDs of the user groups to which the authenticating request is assigned. This will allow the client application to determine the roles that the user currently plays and offer commands to invoke the appropriate commands.

### Updating a user account

#### UpdateUserAccount(login : string, email : string, pass : string, newPass : string, firstName : string, lastName : string)

The operation updates an existing active user record in the database. If incomplete data is provided, or if the password or e-mail address does not meet the criteria for correctness, an exception is signaled. The login and pass parameters must always be specified and **correspond to the current authentication data of a given user**.

If you specify the string "-nochange-" as the value of the newPass parameter, the password will not be changed as a result of this operation.

If you specify the firstName parameter as the value of the text string "-nochange-", the first name, last name, and email will not be changed as a result of this operation.

### Permissions management – user and session groups

#### EvokeGroupMembership(grantedUserLogin : string, groupName : string)

Assigns the user specified by the login name to the specified user group. Requires HMDB admin privileges.

#### RevokeGroupMembership(grantedUserLogin : string, groupName : string)

Excludes the user specified by the login name from the specified user group. Requires HMDB admin privileges.

### Permissions management – single sessions

#### ListSessionPrivileges(sessionID : int) : SessionPrivilegeList

**<basketName>string </ basketName >**

**< SessionPrivilegeList> ( < SessionPrivilege > <Login>string</ Login > <CanWrite>boolean</ CanWrite > </ SessionPrivilege > )\*  
</ SessionPrivilegeList >**

Returns a list of users who have named permissions to access a given session, along with information (CanWrite element) whether the user has the right to modify the session content.

#### GrantSessionPrivileges(grantedUserLogin : string, sessionID : string, write : boolean)

Creates a named permission for the specified user to access the specified session. The write parameter specifies whether the permission includes the ability to update the session.

#### RemoveSessionPrivileges(grantedUserLogin : string, sessionID : string)

The operation removes the named permissions to access the specified session by the specified user.

#### AlterSessionVisibility(sessionID : string, isPublic : boolean, isWritable : boolean)

The action modifies the general visibility settings for a session, specifying whether to make (or remain) a public session and, if so, whether to extend public access to modify the session.

## Creating new accounts – AccountFactoryWS

Current definition: <https://v21.pjwstk.edu.pl/HMDB/AccountFactoryWS.svc?wsdl>

The service does not require authentication. It involves the creation and activation of an account, after which the user's login becomes usable, so other operations, including authentication, may use the normal authentication mechanism.

### Creating a user account

#### CreateUserAccount(login : string, email : string, pass : string, firstName : string, lastName : string)

The operation creates a new user record in the database. In the case of providing incomplete data, or when the provided login or e-mail address is already used by another account present in the database, an exception is signaled. The operation sends a message containing a hyperlink and an account activation code to the e-mail address provided in the call.

#### ActivateUserAccount(login : string, activationCode : string) : bool

The operation marks the user indicated with the login as active. If a login and activation code pair is provided that does not match the code actually generated for a given account, an exception is signaled.

#### ResetPassword(email : string) : bool

The operation prepares the account associated with the specified email address for the reset of its password. An activation code is generated and stored in the database. At the same time, the user's status is changed to a value that will allow the generated code to be treated as a password on the website used to modify the user profile. A message containing instructions on how to reset the password is sent to the specified email address.

## Functionality related to the creation and updating of user accounts

### Creating and activating user accounts

Website Address: <https://v21.pjwstk.edu.pl/HMDB/UserAccountCreation.aspx>

Interacting with the page consists of three steps:

1) Fill in the data in the upper section (login, password, email, first name, last name) and confirm the entered data with the **Create User Account button**. The form validates (minimum login length – 4 characters, password requires a minimum of 6 and a maximum of 20 characters including an uppercase letter, lowercase letter and number). In addition, it may be signaled that a given login is unavailable or that a given email has been previously registered in the database.

2) Receive an email with an activation code.

3) Visit the website again, enter your login, password, email and activation code and confirm with the **Activate User Account button**.



### Editing data for active user accounts

Website address: <https://v21.pjwstk.edu.pl/HMDB/UserAccountUpdate.aspx>

Interaction with the website requires the creation and activation of a user account.

It is necessary to enter a login and password. Depending on the type of update you need, select one or both of the checkboxes on the page and fill in the controls accordingly.

Remark! Updating any one attribute from the name, surname and email requires filling in all three fields – with updated or changed data, respectively.



1. We use English-language names here – such as those used in service interfaces. Hence the slight linguistic awkwardness in the description below. [↑](#footnote-ref-1)
2. Currently, it can also be a decimal number with 0 to 2 decimal places. [↑](#footnote-ref-2)