

Java WS Core WS-Enumeration Design

July 14, 2006

Jarek Gawor {gawor@mcs.anl.gov}

Table of Contents

WS-Enumeration Specification.....	2
Client.....	2
API.....	2
ClientEnumeration.....	2
ClientEnumIterator.....	4
Examples.....	6
ClientEnumeration.....	6
ClientEnumIterator.....	7
Command line clients.....	8
ws-enumerate-start.....	8
ws-enumerate.....	8
ws-enumerate-end.....	8
Service.....	9
Service WSDL.....	9
Service Implementation.....	9
Enumeration Implementation Details.....	9
Example.....	10
API.....	11
EnumIterator.....	11
SimpleEnumIterator.....	12
IndexedObjectFileEnumIterator.....	12
IndexedObjectFileWriter.....	12
IndexedObjectFileReader.....	12
IndexedObjectFileUtils.....	12
Other Implementation Details.....	13
WS-Enumeration WSDL and schema changes.....	13

WS-Enumeration Specification

Specification:

<http://www.w3.org/Submission/WS-Enumeration/>

Schema:

<http://schemas.xmlsoap.org/ws/2004/09/enumeration/enumeration.xsd>

WSDL:

<http://schemas.xmlsoap.org/ws/2004/09/enumeration/enumeration.wsdl>

Client

API

There are two main client-side API for enumerations. The *ClientEnumeration* API provide basic functions for managing enumeration lifetime and retrieving its data. The *ClientEnumIterator* API provide *java.util.Iterator* abstraction for retrieving enumeration data and support automatic data deserialization.

ClientEnumeration

The *ClientEnumeration* API provide basic functions for managing enumeration lifetime and retrieving its data.

The *ClientEnumeration* must be initialized with a *javax.xml.rpc.Stub* instance that is a Stub for the service that implements the WS-Enumeration operations and with an *EnumerationContextType* object returned by the *enumerate* operation of the service or any other operation that initiates an enumeration.

The *javax.xml.rpc.Stub* instance must define all of the WS-Enumeration operations except the *enumerate* operation. Also, the Stub instance must be property configured with the security properties if calling a secure service.

IterationResult pull(IterationConstraints constraints)

Retrieves the next set of elements of the enumeration. The input parameter defines the constraints for the operation such as the maximum of elements to retrieve, the maximum number of characters that the consumer can accept and the maximum amount of time in which the data needs to be returned. The return parameter contains the results of the iteration and an end of sequence flag to indicate if there is more data to be returned. The results of the iteration are of the *javax.xml.soap.SOAPElement* type.

This method calls the WS-Enumeration *pull* operation on the data service.

IterationResult pull()

Same as *pull(IterationConstraints)* function but uses default constraints (maximum number of elements set to 1, no maximum characters limit and no time limit).

void release()

Explicitly releases the enumeration. In general, the enumeration context is automatically released when a client finishes retrieving all the enumeration data or the enumeration expires if it was configured with an expiration time or duration. In cases where no expiration time was set for the enumeration or when not enumerating over the entire data the enumeration should be released explicitly.

This method calls the WS-Enumeration *release* operation on the data service.

EnumExpiration renew(EnumExpiration expiration)

Sets a new expiration time/duration of the enumeration. The input parameter can be null to configure the enumeration without an expiration time/duration (the enumeration will not expire). The expiration time/duration cannot be in the past (as according to the service clock). The service can choose to accept a different expiration time then specified. The return parameter can also be null to indicate that the enumeration does not have an expiration time/duration.

This method calls the WS-Enumeration *renew* operation on the data service

EnumExpiration getStatus()

Gets the current expiration time/duration of the enumeration. The return parameter can be null to indicate that the enumeration does not have an expiration time/duration.

This method calls the WS-Enumeration *getStatus* operation on the data service

ClientEnumIterator

The *ClientEnumIterator* API provide simple-to-use API for enumerating over data using the WS-Enumeration operations. The *ClientEnumIterator* class implements the *java.util.Iterator* interface but the implementation of these functions does not follow the *Iterator* contract exactly because of the WS-Enumeration specification limitations. The *ClientEnumIterator* class uses the *ClientEnumeration* API underneath and in contrast to the *ClientEnumeration* API offers automatic data deserialization.

The *ClientEnumIterator* must be initialized with a *javax.xml.rpc.Stub* instance that is a Stub for the service that implements the WS-Enumeration operations and with an *EnumerationContextType* object returned by the *enumerate* operation of the service or any other operation that initiates an enumeration.

The *javax.xml.rpc.Stub* instance must define all of the WS-Enumeration operations except the *enumerate* operation. Also, the Stub instance must be property configured with the security properties if calling a secure service.

During iteration the *ClientEnumIterator* makes remote calls to the data service to retrieve the next set of items (calls the WS-Enumeration *pull* operation). The frequency of these remote calls is controlled by the *maxElements* setting of the *IterationConstraints* of the *ClientEnumIterator*. If that number is small the *ClientEnumIterator* will make a lot of remote calls but will use a small amount of memory (since it always keeps a few of the items in the memory). But if that number is big the *ClientEnumIterator* will make fewer remote calls but will use more memory (since it now keeps more items in the memory).

void setItemType(Class itemType)

Sets the type of the object returned by the *next()* operation. By default the objects returned by the *next()* operation will be of the *javax.xml.soap.SOAPElement* type. If the item type is set, the enumeration elements will be automatically deserialized into this type. This assumes the enumeration elements are all of the same type.

void setIterationConstraints (IterationConstraints constraints)

Sets iteration constraints for the iterator. By default the constraints are not set and defaults are assumed (maximum number of elements set to 1, no maximum characters limit and no time limit).

Object next()

Returns the next object in the enumeration. The returned object can be null. If item type is set (using the *setItemType* method) the current object will be automatically converted into that type and returned. Otherwise, object of the *javax.xml.soap.SOAPElement* type is returned. If the enumeration has ended (*hasNext()* returns *false*) or has been released on the client the *NoSuchElementException* is raised. Also, in certain cases the *NoSuchElementException* can also be raised even though *hasNext()* returned *true*. If deserialization is performed and it fails a *ConversionException* is raised and the index of the iteration is not advanced (so that the user can specify another item type or disable deserialization).

This method calls the WS-Enumeration *pull* operation on the data service.

boolean hasNext()

Tests if there are more elements in the enumeration to be returned. If it returns *false*, there are no more elements to be returned. If *true*, there **might** be more elements to be returned. This method can return *true* even though the *next()* operation consistently returns null. Also, this method can return *true* even though the *next()* operation will throw *NoSuchElementException*.

Object convert(SOAPElement element)

Performs object conversion on the enumeration element. This function is called by the *next()* function every time the *next()* function is called. It can be used to deserialize the enumeration element into appropriate Java object. This function is meant to be overwritten by the subclasses of the *ClientEnumIterator* to provide custom deserialization behavior. If deserialization fails a *ConversionException* is thrown.

void release()

Explicitly releases the enumeration context.

This method calls the WS-Enumeration *release* operation on the data service. *hasNext()* will return *false* and *next()* will throw *NoSuchElementException* after this method is called.

Examples

ClientEnumeration

This example shows how to iterate over the data using *ClientEnumeration* API.

```
import org.globus.ws.enumeration.ClientEnumeration;
import org.globus.ws.enumeration.IterationResult;
import org.globus.ws.enumeration.IterationConstraints;
...

EnumerationServiceAddressingLocator locator =
    new EnumerationServiceAddressingLocator();

URL serviceURL =
    new URL("http://127.0.0.1:8080/wsrf/services/EnumerationService");

EnumerationPortType port =
    locator.getEnumerationPortTypePort(serviceURL);

// obtain the enumeration context from the service somehow
EnumerationContextType context = ...

// create iteration constraints (return maximum of 10 elements)
IterationConstraints constraints =
    new IterationConstraints(10, -1, null);

// create client enumeration
ClientEnumeration enumeration =
    new ClientEnumeration((Stub)port, context);

// iterate over the data
IterationResult iterResult;
do {
    // retrieve the enumeration data with given constraints
    iterResult = enumeration.pull(constraints);
    Object [] items = iterResult.getItems();
    if (items != null) {
        // display the enumeration data
        for (int i=0; i < items.length; i++) {
            System.out.println(items[i]);
        }
    }
} while (!iterResult.isEndOfSequence());
```

ClientEnumIterator

This example shows how to iterate over the data using *ClientEnumIterator* API.

```
import org.globus.ws.enumeration.ClientEnumIterator;
import org.globus.ws.enumeration.IterationConstraints;

...

EnumerationServiceAddressingLocator locator =
    new EnumerationServiceAddressingLocator();

URL serviceURL =
    new URL("http://127.0.0.1:8080/wsrf/services/EnumerationService");

EnumerationPortType port =
    locator.getEnumerationPortTypePort(serviceURL);

// obtain the enumeration context from the service somehow
EnumerationContextType context = ...

// create iteration constraints (return maximum of 10 elements)
IterationConstraints constraints =
    new IterationConstraints(10, -1, null);

// create the client iterator
ClientEnumIterator iterator =
    new ClientEnumIterator((Stub)port, context);

iterator.setIterationConstraints(constraints);

// iterate over the data
try {
    while(iterator.hasNext()) {
        Object obj = iterator.next();
    }
} catch (NoSuchElementException e) {
    // next() can throw this exception even though
    // hasNext() returned true
}
```

Command line clients

ws-enumerate-start

Starts an enumeration. It calls the *enumerate* operation on the data service and prints out the enumeration context to the console. The enumeration context then can be passed to *ws-enumerate* or *ws-enumerate-end* clients.

ws-enumerate

Enumerates over the data. It calls the *pull* operation on the data service and prints out the retrieved data to the console. The client requires an argument that is a filename that contains the enumeration context (created either by *ws-enumerate-start* or other means). The *-n*, *--maxElements* option can be used to configure the maximum number of elements to retrieve from the data service at a time. The *-r*, *--maxCharacters* option can be used to configure the maximum number of characters the client can accept at a time. The *-t*, *--maxTime* option can be used to specify the maximum amount of time in which the enumeration data has to be assembled. Any combination of these options can be specified at the same time.

ws-enumerate-end

Releases an enumeration. It calls the *release* operation on the data service. The client requires an argument that is a filename that contains the enumeration context (created either by *ws-enumerate-start* or other means).

Service

Service WSDL

The service that wishes to support enumerations must define the WS-Enumeration operations in its WSDL. All operations except the *enumerate* operation must be defined in service WSDL. The *enumerate* operation of WS-Enumeration specification is an optional operation and therefore it is up to the service designer to decide if the service should define and implement this operation or if the service will provide some other operation that will initiate an enumeration. Any operation of the service can initiate an enumeration by returning an element of the *wsen:EnumerationContextType* type to the client.

Service Implementation

The service must implement the *enumerate* operation of the WS-Enumeration specification or provide some other operation that will initiate an enumeration and return an element of the *wsen:EnumerationContextType* type to the client.

For all the other WS-Enumeration operations the service must be configured with the built-in enumeration operation provider (*EnumProvider*). Of course, the service can choose to provide its own implementation for the WS-Enumeration operations but will need to replicate a lot of the built-in functionality.

The *EnumProvider* is configured in the same way as any other operation provider in the service deployment description (WSDD) file. All the WS-Enumeration operations should have the same security settings.

Enumeration Implementation Details

Internally, enumerations are managed and implemented just like any other WS-Resources. That is, there are enumeration resources (*EnumResource*) which are managed by the enumeration resource home (*EnumResourceHome*). The enumeration resources contain lifetime information and have a reference to the iterator (*EnumIterator*) that provides the actual data iteration functionality.

Types

There are two types of enumerations: transient and persistent. The transient enumerations live only while the container is running and are not restored after a container restart. The persistent enumerations are restored after a container restart. The type of enumeration has no impact on how the data of the enumeration is stored or retrieved. For example, a transient enumeration can query a database, retrieve data from a file or have all the data

in memory. It is entirely up to the service developers to decide how the data is retrieved, if the data is static or dynamic, etc.

In general it is not recommended to keep the entire enumeration data in memory. If the data is static, it is recommended to store the data in a database or a file, etc. and retrieve it in an efficient way.

Visibility

The enumeration resources also contain visibility properties (*VisibilityProperties*) to restrict what service and/or resource can access the particular enumeration resource. In general, an enumeration created by service S is only accessible through service S. Similarly, an enumeration created by resource R is only accessible through resource R.

Security

The service or resource through which the enumeration data is accessed can be configured with a security descriptor to further control access to the data.

Example

This example shows how to create a transient enumeration on the server-side with the help of the built-in WS-Enumeration operation provider.

```
import org.globus.ws.enumeration.EnumResourceHome;
import org.globus.ws.enumeration.EnumIterator;
import org.globus.ws.enumeration.EnumResource;
import org.globus.ws.enumeration.EnumProvider;
...

// obtain enumeration resource home
EnumResourceHome enumHome = EnumResourceHome.getEnumResourceHome();

// create iterator for the data
EnumIterator iter = ...;

// create transient enumeration resource for the iterator
// with visibility properties obtained from the context
EnumResource resource = enumHome.createEnumeration(iter, false);

// get resource key for the enumeration resource
ResourceKey key = enumHome.getKey(resource);

// create EnumerationContextType to be returned to the client
EnumerationContextType enumContext =
    EnumProvider.createEnumerationContextType(key);
```

API

EnumIterator

This API is used by the service developers to write their own *EnumIterator* implementations in order to retrieve the enumeration data in a fast and efficient way. A new *EnumIterator* instance must be created for each new enumeration. The implementations should assume a single thread access. Only one client is allowed to access a particular enumeration at a time. The implementation must keep track of the progress of the enumeration (for example, store the index of the last item retrieved). For persistent enumerations, the *EnumIterator* implementation must be fully serializable using the Java serialization framework. That will enable the enumeration to be restored in case of a container restart or in other conditions. An application should not keep references to the *EnumIterator* objects it creates. Such references will prevent efficient memory management by the *EnumResourceHome*.

IterationResult next(IterationConstraints constraints)

Retrieves the next set of items of the enumeration. The *IterationConstraints* define constraints for this operation such as the maximum number of the items that can be returned, the maximum number of characters of the items, and timeout in which the items must be returned. The constraints can change between the calls. If the timeout value constraint is specified and the data is not collected in that time, a *TimeoutException* should be raised. If there are no more elements in the enumeration a *NoSuchElementException* is raised.

The *IterationResult* contains the result of the iteration that fulfills the specified constraints. It must always be non-null. The *IterationResult* itself contains a list of enumeration items of *javax.xml.soap.SOAPElement* type and a flag that indicates if an end of sequence has been reached.

void release()

Release any resources associated with this enumeration. For example, close database connections, delete files, etc. This method is called when the enumeration is explicitly released, expires, or the user finished enumerating through all the data.

SimpleEnumIterator

The *SimpleEnumIterator* is a concrete implementation of the *EnumIterator* interface. It is a very simple implementation that can enumerate over in-memory data passed either as an array of objects or a list (*java.util.List*). The enumeration contents can be of *javax.xml.soap.SOAPElement* type, simple types such as *java.lang.Integer*, etc. or Axis generated Java beans.

The *SimpleEnumIterator* can only be used with transient type of enumerations.

IndexedObjectFileEnumIterator

The *IndexedObjectFileEnumIterator* is another concrete implementation of the *EnumIterator* interface. It is a memory efficient implementation that can enumerate over data stored in an indexed file created by *IndexedObjectFileWriter*. The indexed file format is optimized for retrieving objects in a sequential and random manner. The *IndexedObjectFileEnumIterator* uses the *IndexedObjectFileReader* to read the indexed file and quickly locate and retrieve the next set of objects of the enumeration.

The *IndexedObjectFileEnumIterator* can be used with transient and persistent types of enumerations.

IndexedObjectFileWriter

The *IndexedObjectFileWriter* is used to create an indexed file. The objects stored in the file will be serialized using the Java serialization framework, therefore, only the objects that implement the *java.io.Serializable* interface can be used.

IndexedObjectFileReader

The *IndexedObjectFileReader* is used to read an indexed file created by the *IndexedObjectFileWriter*. The objects stored in the file will be deserialized using the Java serialization framework.

IndexedObjectFileUtils

The *IndexedObjectFileUtils* is a collection of utility functions that can be used to create indexed files with the given data.

Other Implementation Details

WS-Enumeration WSDL and schema changes

The following changes have been made to the WS-Enumeration WSDL and schema files:

1. The WS-Addressing namespace used by the specification was changed to <http://schemas.xmlsoap.org/ws/2004/03/addressing> in order to work with the existing tooling.
2. The *EnumerationEndOp* operation was commented out as it violates WS-I Basic Profile 1.1 and is not supported by the tooling. Therefore, this part of WS-Enumeration functionality is not supported by the current implementation.
3. Since the *EnumerateOp* operation is an optional operation, it was moved into a separate port type called *DataSourceStart*. All other operations remain in the *DataSource* port type.
4. The *EnumerationContextType* type was simplified to an equivalent form in order to be properly recognized by the tooling.

Other comments on the WS-Enumeration WSDL and schema files:

1. The schema file uses the *xsd:union* type which makes it hard for the tooling to figure out which value was actually serialized. The *xsd:choice* type might be better for such cases.
2. Currently there is no way to ask the data service if it has any more elements without actually retrieving some elements.
3. The *EndOfSequence* element in schema file should be defined with *xsd:boolean* type. Right now it defaults to *xsd:anyType*.