# APS External Protocol Extender

User Guide

Version: 0.9.1

Author: Tommy Svensson

Copyright © 2013 Natusoft AB

Table of Contents

# 1 APSExternalProtocolExtender

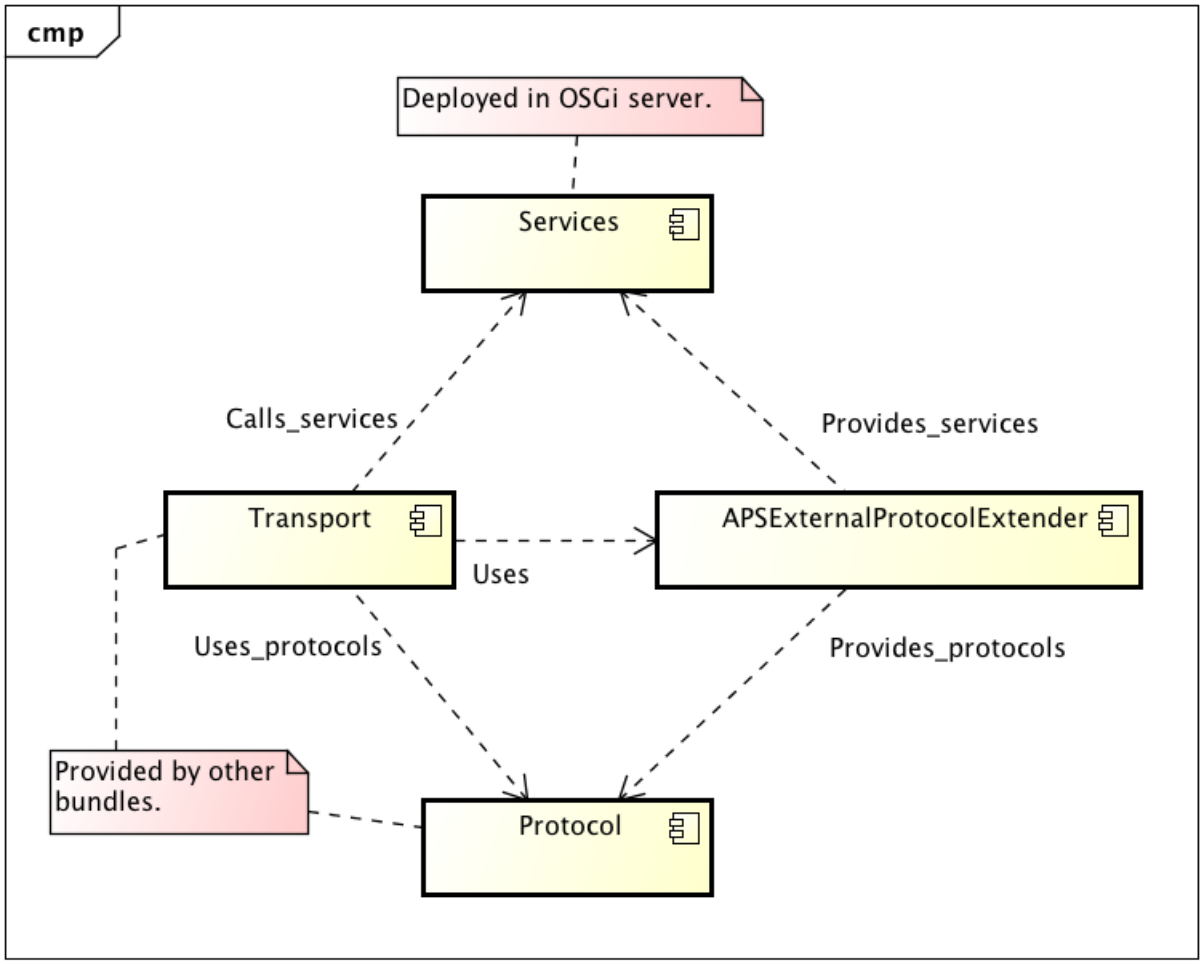
This is an OSGi bundle that makes use of the OSGi extender pattern. It listens to services being registered and unregistered and if the services bundles *MANIFEST.MF* contains ”APS-Externalizable: true” the service is made externally available. If the *MANIFEST.MF* contains ”APS-Externalizable: false” however making the service externally available is forbidden.

The exernal protocol extender also provides a configuration where services can be specified with their fully qualified name to be made externally available. If a bundle however have specifically specified false for the above manifest entry then the config entry will be ignored.

So, what is meant by ”made externally available” ? Well what this bundle does is to analyze with reflection all services that are in one way or the other specified as being externalizable (manifest or config) and for all callable methods of the service an *APSExternallyCallable* object will be created and saved locally with the service name. *APSExternallyCallable* extends *java.util.concurrent.Callable*, and adds the possibility to add parameters to calls and also provides meta data for the service method, and the bundle it belongs to.

## 1.1 The overall structure

The complete picture for making services externally callable looks like this:




This bundle provides the glue between the services and the protocols. Transports and protocols have to be provided

by other bundles.

The flow is like this:

1. Transport gets some request and an InputStream.
2. Transport gets some user selected protocol (The APSExtProtocolHTTPTransportProvider allows specification of both protocol, protocol version, and service to call in the URL).
3. Transport calls APSExternalProtocolService to get requested protocol.
4. Transport calls protocol to parse InputStream and it returns an RPCRequest.
5. Transport uses the information in the RPCRequest to call a service using APSExternalProtocolService.
6. Transport takes the result from the call and passes to the protocol along with an OutputStream to write response on.

## 1.2 APSExternalProtocolService

This bundle registers an *APSExternalProtocolService* that will provide all *APSExternallyCallable* instances (or rather copies of them since you can modify the one you get back by providing arguments). This service also provides getters for available remote protocols and you can register with it to receive information about chages for services and protocols.

### 1.2.1 Protocols

There is a base API for protocols: RPCProtocol. APIs for different types of protocols should extend this. There is currently only one type of protocol available: *StreamedRPCProtocol*. The protocol type APIs are service APIs and services implementing them must be provided by other bundles. This bundle looks for and keeps track of all such service providers.

The *StreamedRPCProtocol* provides a method for parsing a request from an InputStream returning an RPCRequest object. This request object contains the name of the service, the method, and the parameters. This is enough for using *APSExternalProtocolService* to do a call to the service. The request object is also used to write the call response on an OutputStream. There is also a method to write an error response.

It is the responsibility of the transport provider to use a protocol to read and write requests and responses and to use the request information to call a service method.

### 1.2.2 Getting information about services and protocols.

A transport provider can register themselves with the *APSExternalProtocolService* by implementing the *APSExternalProtocolListener* interface. They will then be notified when a new externalizable service becomes available or is leaving and when a protocol becomes available or is leaving.

## 1.3 See also

*APSExtProtocolHTTPTransportProvider* - Provides a HTTP transport.

*APSStreamedJSONRPCProtocolProvider* - Provides version 1.0 and 2.0 of JSONRPC.

## 1.4 APIs

public *interface* **APSExternalProtocolService** [se.natusoft.osgi.aps.api.external.extprotocolsvc] {

*This service makes the currently available externalizable services available for calling. It should be used by a bundle providing an externally available way of calling a service (JSON over http for example) to translate and forward calls to the local service. The locally called service is not required to be aware that it is called externally. __Never cache any result of this service!__ Always make a new call to get the current state. Also note that it is possible that the service represented by an APSExternallyCallable have gone away after it was returned, but before you do call() on it! In that case an APSNoServiceAvailableException will be thrown. Note that you can register as an APSExternalProtocolListener to receive notifications about externalizable services coming and going, and also protocols coming and going to keep up to date with the current state of things.*

**public Set<String> getAvailableServices()**

*Returns all currently available services.*

**public List<APSExternallyCallable> getCallables(String serviceName) throws RuntimeException**

*Returns all APSExternallyCallable for the named service object.*

*Parameters*

*serviceName* *- The name of the service to get callables for.*

*Throws*

*RuntimeException* *- If the service is not available.*

**public boolean isRESTCallable(String serviceName) throws RuntimeException**

*Returns true if the service has _put*(...)_, _get*(...)_, and/or _delete*(...)_ methods. This is to help HTTP transports support REST calls.*

*Parameters*

*serviceName* *- The service to check if it has any REST methods.*

**public APSRESTCallable getRESTCallable(String serviceName)**

*Returns an APSRESTCallable containing one or more of post, put.get, and delete methods. This is to help HTTP transports support REST calls.*

*Parameters*

*serviceName* *- The name of the service to get the REST Callables for.*

**public Set<String> getAvailableServiceFunctionNames(String serviceName)**

*Returns the names of all available functions of the specified service.*

*Parameters*

*serviceName* *- The service to get functions for.*

**public APSExternallyCallable getCallable(String serviceName, String serviceFunctionName)**

*Gets an APSExternallyCallable for a specified service name and service function name.*

*Returns*

*An APSExternallyCallable instance or null if the combination of service and serviceFunction is not available.*

*Parameters*

*serviceName* *- The name of the service object to get callable for.*

*serviceFunctionName* *- The name of the service function of the service object to get callable for.*

**public List<RPCProtocol> getAllProtocols()**

*Returns*

*All currently deployed providers of RPCProtocol.*

**public RPCProtocol getProtocolByNameAndVersion(String name, String version)**

*Returns an RPCProtocol provider by protocol name and version.*

*Returns*

*Any matching protocol or null if nothing matches.*

*Parameters*

*name* *- The name of the protocol to get.*

*version* *- The version of the protocol to get.*

**public List<StreamedRPCProtocol> getAllStreamedProtocols()**

*Returns*

*All currently deployed providers of StreamedRPCProtocol.*

**public StreamedRPCProtocol getStreamedProtocolByNameAndVersion(String name, String version)**

*Returns a StreamedRPCProtocol provider by protocol name and version.*

*Returns*

*Any matching protocol or null if nothing matches.*

*Parameters*

*name* *- The name of the streamed protocol to get.*

*version* *- The version of the streamed protocol to get.*

**public void addExternalProtocolListener(APSExternalProtocolListener externalServiceListener)**

*Add a listener for externally available services.*

*Parameters*

*externalServiceListener* *- The listener to add.*

**public void removeExternalProtocolListener(APSExternalProtocolListener externalServiceListener)**

*Removes a listener for externally available services.*

*Parameters*

*externalServiceListener* *- The listener to remove.*

}

---

public *interface* **APSExternallyCallable<ReturnType>** extends Callable<ReturnType> [se.natusoft.osgi.aps.api.external.extprotocolsvc.model] {

*This API represents one callable service method.*

**public String getServiceName()**

*Returns*

*The name of the service this callable is part of.*

**public String getServiceFunctionName()**

*Returns*

*The name of the service function this callable represents.*

**public DataTypeDescription getReturnDataDescription()**

*Returns*

*A description of the return type.*

**public List<ParameterDataTypeDescription> getParameterDataDescriptions()**

*Returns*

*A description of each parameter type.*

**public Bundle getServiceBundle()**

*Returns*

*The bundle the service belongs to.*

**public void setArguments(Object... value)**

*Provides parameters to the callable using a varags list of parameter values.*

*Parameters*

*value* *- A parameter value.*

**ReturnType call() throws Exception**

*Calls the service method represented by this APSExternallyCallable.*

*Returns*

*The return value of the method call if any or null otherwise.*

*Throws*

*Exception* *- Any exception the called service method threw.*

}

---

public *interface* **APSExternalProtocolListener** [se.natusoft.osgi.aps.api.external.extprotocolsvc.model] {

*A listener for externally available services. Please note that this means that the service is available for potential external protocol exposure! For it to be truly available there also has to be a protocol and transport available. It is probably only transports that are interested in this information!*

**public void externalServiceAvailable(String service, String version)**

*This gets called when a new externally available service becomes available.*

*Parameters*

*service* - *The fully qualified name of the newly available service.*

*version* - *The version of the service.*

**public void externalServiceLeaving(String service, String version)**

*This gets called when an externally available service no longer is available.*

*Parameters*

*service* - *The fully qualified name of the service leaving.*

*version* - *The version of the service.*

**public void protocolAvailable(String protocolName, String protocolVersion)**

*This gets called when a new protocol becomes available.*

*Parameters*

*protocolName* - *The name of the protocol.*

*protocolVersion* - *The version of the protocol.*

**public void protocolLeaving(String protocolName, String protocolVersion)**

*This gets called when a new protocol is leaving.*

*Parameters*

*protocolName* - *The name of the protocol.*

*protocolVersion* - *The version of the protocol.*

}

---

}

---

public *class* **APSRESTException** extends APSRuntimeException [se.natusoft.osgi.aps.api.net.rpc.errors] {

*This is a special exception that services can throw if they are intended to be available as REST services through the APSExternalProtocolExtender + APSRPCHTTPTransportProvider. This allows for better control over status codes returned by the service call.*

**public APSRESTException(int httpStatusCode)**

*Creates a new APSRESTException.*

*Parameters*

*httpStatusCode* *- The http status code to return.*

**public APSRESTException(int httpStatusCode, String message)**

*Creates a new APSRESTException.*

*Parameters*

*httpStatusCode* *- The http status code to return.*

*message* *- An error message.*

**public int getHttpStatusCode()**

*Returns the http status code.*

}

---

public *enum* **ErrorType** [se.natusoft.osgi.aps.api.net.rpc.errors] {

*This defines what I think is a rather well though through set of error types applicable for an RPC call. No they are not mine, they come from Matt Morley in his JSONRPC 2.0 specification at http://jsonrpc.org/spec.html. I did add SERVICE_NOT_FOUND since it is fully possible to try to call a service that does not exist.*

**PARSE_ERROR**

*Invalid input was received by the server. An error occurred on the server while parsing request data.*

**INVALID_REQUEST**

*The request data sent is not a valid.*

**METHOD_NOT_FOUND**

*The called method does not exist / is not available.*

**SERVICE_NOT_FOUND**

*The called service does not exist / is not available.*

**INVALID_PARAMS**

*The parameters to the method are invalid.*

**INTERNAL_ERROR**

*Internal protocol error.*

**SERVER_ERROR**

*Server related error.*

}

---

public *interface* **HTTPError** extends RPCError [se.natusoft.osgi.aps.api.net.rpc.errors] {

*Extends RPCError with an HTTP status code. HTTP transports can make use of this information.*

**public int getHttpStatusCode()**

*Returns*

*Returns an http status code.*

}

---

public *interface* **RPCError** [se.natusoft.osgi.aps.api.net.rpc.errors] {

*This represents an error in servicing an RPC request.*

**public ErrorType getErrorType()**

*The type of the error.*

**public String getErrorCode()**

*A potential error code.*

**public String getMessage()**

*Returns an error message. This is also optional.*

**public boolean hasOptionalData()**

*True if there is optional data available. An example of optional data would be a stack trace for example.*

**public String getOptionalData()**

*The optional data.*

}

---

public *class* **RequestedParamNotAvailableException** extends APSException [se.natusoft.osgi.aps.api.net.rpc.exceptions] {

*This exception is thrown when a parameter request cannot be fulfilled.*

**public RequestedParamNotAvailableException(String message)**

*Creates a new APSException instance.*

*Parameters*

*message* *- The exception message.*

**public RequestedParamNotAvailableException(String message, Throwable cause)**

*Creates a new APSException instance.*

*Parameters*

*message* *- The exception message.*

*cause* *- The cause of this exception.*

}

---

public *abstract class* **AbstractRPCRequest** implements RPCRequest [se.natusoft.osgi.aps.api.net.rpc.model] {

*This contains a parsed JSONRPC request.*

**public AbstractRPCRequest(String method)**

*Creates a new AbstractRPCRequest.*

*Parameters*

*method* *- The method to call.*

**public AbstractRPCRequest(RPCError error)**

*Creates a new AbstractRPCRequest.*

*Parameters*

*error* *- An RPCError indicating a request problem, most probably of ErrorType.PARSE_ERROR type.*

**public AbstractRPCRequest(String method, Object callId)**

*Creates a new AbstractRPCRequest.*

*Parameters*

*method* *- The method to call.*

*callId* *- The callId of the call.*

**protected Map<String, Object> getNamedParameters()**

*Returns*

*The named parameters.*

**protected List<Object> getParameters()**

*Returns*

*The sequential parameters.*

**public void setServiceQName(String serviceQName)**

*Sets the fully qualified name of the service to call. This is optional since not all protocol delivers a service name this way.*

*Parameters*

*serviceQName* *- The service name to set.*

**public void addParameter(Object parameter)**

*Adds a parameter. This is mutually exclusive with addParameter(name, parameter)!*

*Parameters*

*parameter* - *The parameter to add.*

}

---

public *interface* **RPCRequest** [se.natusoft.osgi.aps.api.net.rpc.model] {

**boolean isValid()**

*Returns true if this request is valid. If this returns false all information except getError() is invalid, and getError() should return a valid RPCError object.*

**RPCError getError()**

*Returns an RPCError object if isValid() == false, null otherwise.*

**String getServiceQName()**

*Returns a fully qualified name of service to call. This will be null for protocols where service name is not provided this way. So this cannot be taken for given!*

**String getMethod()**

*REST protocol where it will be part of the URL.*

*Returns*

*The method to call. This can return null if the method is provided by other means, for example a*

**boolean hasCallId()**

*Returns true if there is a call id available in the request.*

*A call id is something that is received with a request and passed back with the response to the request. Some RPC implementations will require this and some wont.*

**Object getCallId()**

*Returns the method call call Id.*

*A call id is something that is received with a request and passed back with the response to the request. Some RPC implementations will require this and some wont.*

**int getNumberOfParameters()**

*Returns*

*The number of parameters available.*

**<T> T getIndexedParameter(int index, Class<T> paramClass) throws RequestedParamNotAvailableException**

*Returns the parameter at the specified index.*

*Returns*

*The parameter object or null if indexed parameters cannot be delivered.*

*Parameters*

*index* *- The index of the parameter to get.*

*paramClass* *- The expected class of the parameter.*

*Throws*

*RequestedParamNotAvailableException* *- if requested parameter is not available.*

}

---

public *interface* **RPCProtocol** [se.natusoft.osgi.aps.api.net.rpc.service] {

*This represents an RPC protocol provider. This API is not enough in itself, it is a common base for different protocols.*

**String getServiceProtocolName()**

*Returns*

*The name of the provided protocol.*

**String getServiceProtocolVersion()**

*Returns*

*The version of the implemented protocol.*

**String getRequestContentType()**

*Returns*

*The expected content type of a request. This should be verified by the transport if it has content type availability.*

**String getResponseContentType()**

*Returns*

*The content type of the response for when such can be provided.*

**String getRPCProtocolDescription()**

*Returns*

*A short description of the provided service. This should be in plain text.*

**RPCError createRPCError(ErrorType errorType, String errorCode, String message, String optionalData, Throwable cause)**

*Factory method to create an error object.*

*Returns*

*An RPCError implementation or null if not handled by the protocol implementation.*

*Parameters*

*errorType* *- The type of the error.*

*errorCode* *- An error code representing the error.*

*message* *- An error message.*

*optionalData* *- Whatever optional data you want to pass along or null.*

*cause* *- The cause of the error.*

}

---

public *interface* **StreamedHTTPProtocol** extends StreamedRPCProtocol [se.natusoft.osgi.aps.api.net.rpc.service] {

*This is a marker interface indicating that the protocol is really assuming a HTTP transport and is expecting to be able to return http status codes. This also means it will be returning an HTTPError (which extends RPCError) from _createError(...)_. It might be difficult for non HTTP transports to support this kind of protocol, and such should probably ignore these protocols. For example a REST implementation of this protocol will not be writing any error response back, but rather expect the transport to deliver the http status code it provides. A non HTTP transport*

*will not be able to know how to communicate back errors in this case since it will not know anything about the protocol itself.*

**boolean supportsREST()**

*Returns*

*true if the protocol supports REST.*

}

---

public *interface* **StreamedRPCProtocol** extends RPCProtocol [se.natusoft.osgi.aps.api.net.rpc.service] {

*This represents an RPC protocol provider that provide client/service calls with requests read from an InputStream or having parameters passes as strings and responses written to an OutputStream. HTTP transports can support both _parseRequests(...)_ and _parseRequest(...)_ while other transports probably can handle only _parseRequests(...)_. __A protocol provider can return null for either of these!__ Most protocol providers will support _parseRequests(...)_ and some also _parseRequest(...)_.*

**List<RPCRequest> parseRequests(String serviceQName, String method, InputStream requestStream) throws IOException**

*Parses a request from the provided InputStream and returns 1 or more RPCRequest objects.*

*Returns*

*The parsed requests.*

*Parameters*

*serviceQName - A fully qualified name to the service to call. This can be null if service name is provided on the stream.*

*method - The method to call. This can be null if method name is provided on the stream.*

*requestStream - The stream to parse request from.*

*Throws*

*IOException - on IO failure.*

**RPCRequest parseRequest(String serviceQName, String method, Map<String, String> parameters) throws IOException**

*Provides an RPCRequest based on in-parameters. This variant supports HTTP transports.*

*Returns*

*The parsed requests.*

*Parameters*

*serviceQName* *- A fully qualified name to the service to call. This can be null if service name is provided on the stream.*

*method* *- The method to call. This can be null if method name is provided on the stream.*

*parameters* *- parameters passed as a*

*Throws*

*IOException* *- on IO failure.*

**void writeResponse(Object result, RPCRequest request, OutputStream responseStream) throws IOException**

*Writes a successful response to the specified OutputStream.*

*Parameters*

*result* *- The resulting object of the RPC call or null if void return. If is possible a non void method also returns null!*

*request* *- The request this is a response to.*

*responseStream* *- The OutputStream to write the response to.*

*Throws*

*IOException* *- on IO failure.*

**boolean writeErrorResponse(RPCError error, RPCRequest request, OutputStream responseStream) throws IOException**

*Writes an error response.*

*Returns*

*true if this call was handled and an error response was written. It returns false otherwise.*

*Parameters*

*error* *- The error to pass back.*

*request* *- The request that this is a response to.*

*responseStream* *- The OutputStream to write the response to.*

*Throws*

*IOException* *- on IO failure.*

}

---