APS External Protocol HTTP Transport Provider

User Guide

Version: 1.0.0

Author: Tommy Svensson

Table of Contents

1 APSExtProtocolHTTPTransportProvider

This provides an http transport for simple remote requests to OSGi services that have "APS-Externalizable: true" in their META-INF/MANIFEST.MF. This follows the OSGi extender pattern and makes any registered OSGi services of bundles having the above manifest entry available for remote calls over HTTP. This transport makes use of the aps-external-protocol-extender which exposes services with the above mentioned manifest entry with each service method available as an APSExternallyCallable.The aps-ext-protocol-http-transport-provider for example acts as a mediator between the protocol implementations and aps-external-protocol-extender for requests over HTTP.

Please note that depending on protocol not every service method will be callable. It depends on its arguments and return value. It mostly depends on how well the protocol handles types and can convert between the caller and the service.

This does not provide any protocol, only transport! For services to be able to be called at least one protocol is needed. Protocols are provided by providing an implementation of se.natusoft.osgi.aps.api.net.rpc.service.StreamedRPCProtocol and registering it as an OSGi service. The StreamedRPCProtocol API provides a protocol name and protocol version getter which is used to identify it. A call to an RPC service looks like this:

    http://host:port/apsrpc/*protocol*/*version*[/*service*][/*method*]

protocol - This is the name of the protocol to use. An implementation of that protocol must of course be available for this to work. If it isn't you will get a 404 back! The protocol service (RPCProtocol<-StreamedRPCProtocol) provides a name for each protocol. It is this name that is referenced.

version - This is the version of the protocol. If this doesn't match any protocols available you will also get a 404 back.

service - This is the service to call. Depending on the protocol you might not need this. But for protocols that only provide method in the stream data like JSONRPC for example, then this is needed. When provided it has to be a fully qualified service interface class name.

method - This is the method to call. The need for this also depends on the protocol. A REST protocol would need it. The JSONRPC protocol does not. When this is specified in the URL then it will be used even if the protocol provides the method in the request! Please note that a method can be specified on two ways:

- method(type,...)
- method

The method will be resolved in that order. The parameter type specifying version is required when there are several methods with the same name but different parameters. The method name only will give you the last one in that case.

1.1 Examples

See examples under the **APSStreamedJSONRPCProtocolProvider** section.

1.2 Authentication

Authentication for services are provided in 2 ways. Both require a userid and a password and both validate the user using the APSAuthService.

The 2 alternatives are:

- http://.../apsrpc/**auth:user:password**/protocol/...
- Basic HTTP authentication using header: 'Authorization: Basic {*base 64 encoded user:password*}'.

One of these will be required if the *requireAuthentication* configuration have been enabled.

1.3 The help web

Opening the *http://.../apsrpc/_help/* URL will give you a web page that provides a lot of information. This page requires authentication since it register itself with the APSAdminWeb (/apsadminweb) as ”Remote Services” and appears there as a tab, and thus joins in with the APSAdminWeb authentication.

In addition to much of the same information as in this documentation it also lists all protocols tracked by the *APSExternalProtocolExtender* with their name, version, description, and other properties. Next it lists all services that *APSExternalProtocolExtender* provides as callable. Each of these services are a link that can be clicked. Clicking on a service will show all the methods of the service and then list the call url for each method per protocol. Each method listed is also a link, and clicking that link will give you a page where you can provide arguments and then press execute to call the service. The result will be displayed as JSON on the same page. This is very useful for testing and debugging services.

1.4 See Also

Also look at the documentation for APSExternalProtocolExtender.