

Last edited by  [Ondřej Vašíček](#) 1 day ago

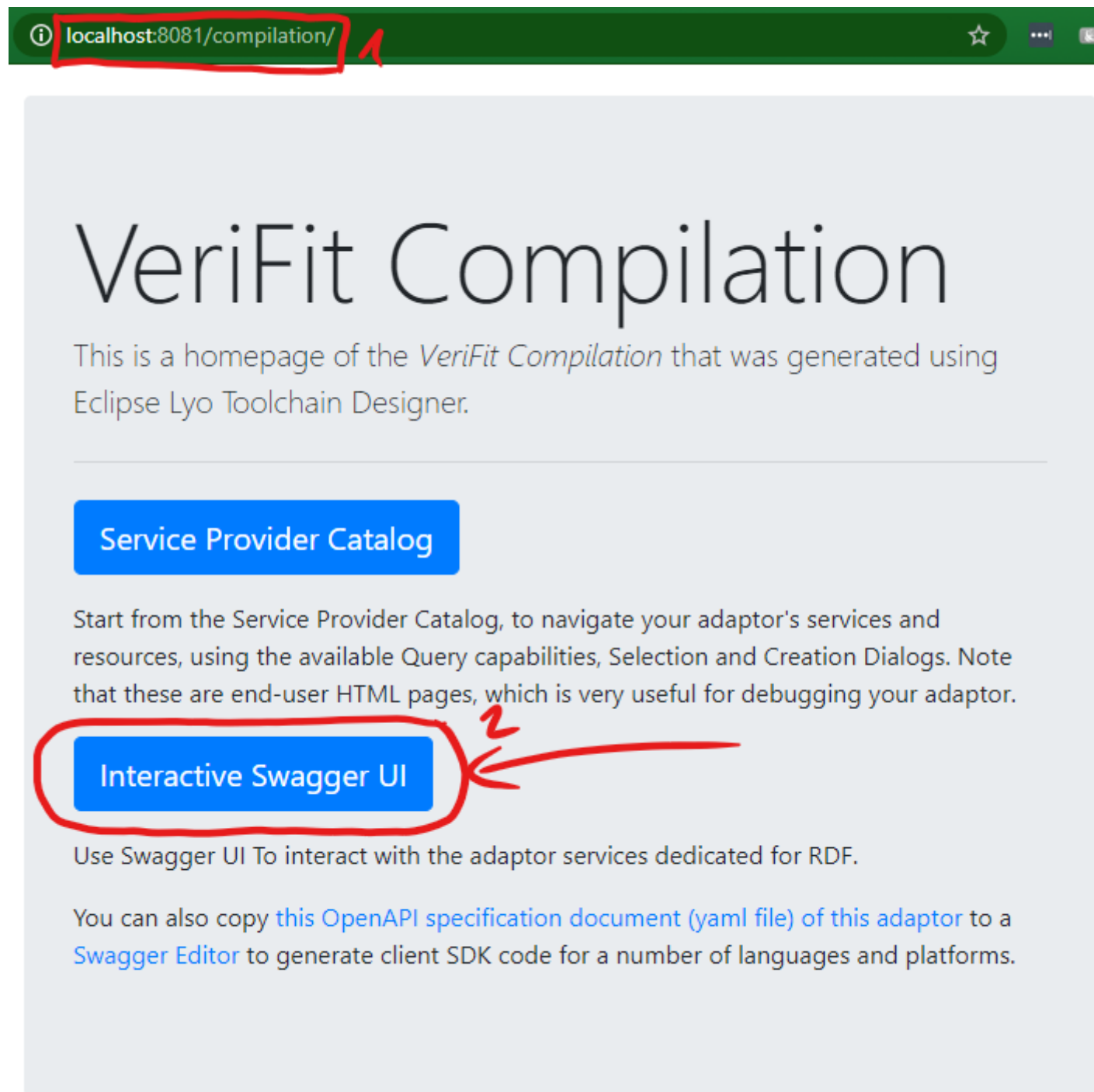
3. Analysis Using SwaggerUI

This guide includes links to the adapter's API. These links will only work if the adapter is running on localhost with default host:port configuration.

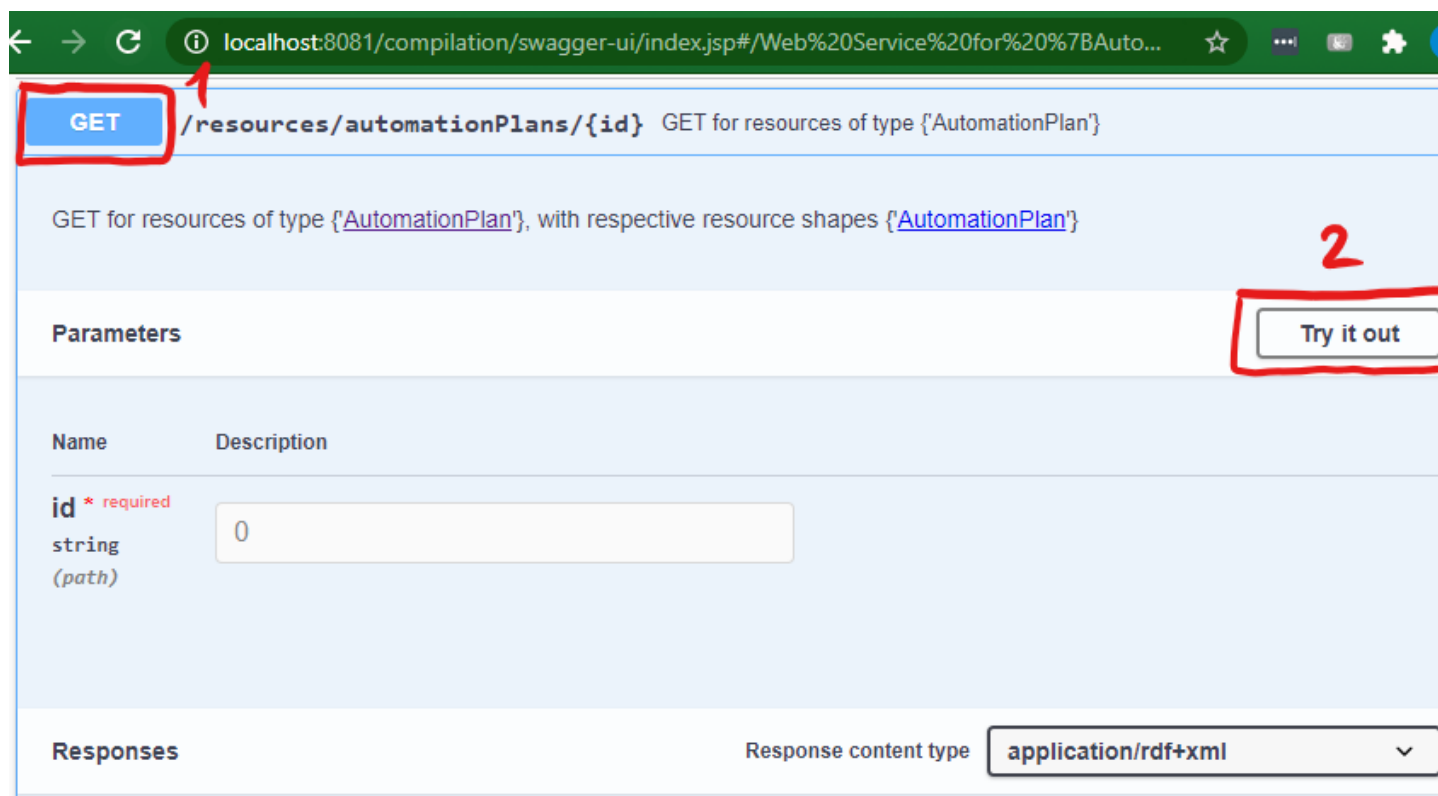
An alternative to this tutorial is the [Postman Tutorial collection](#). That one, however, requires Postman to be installed.

1) Create an SUT

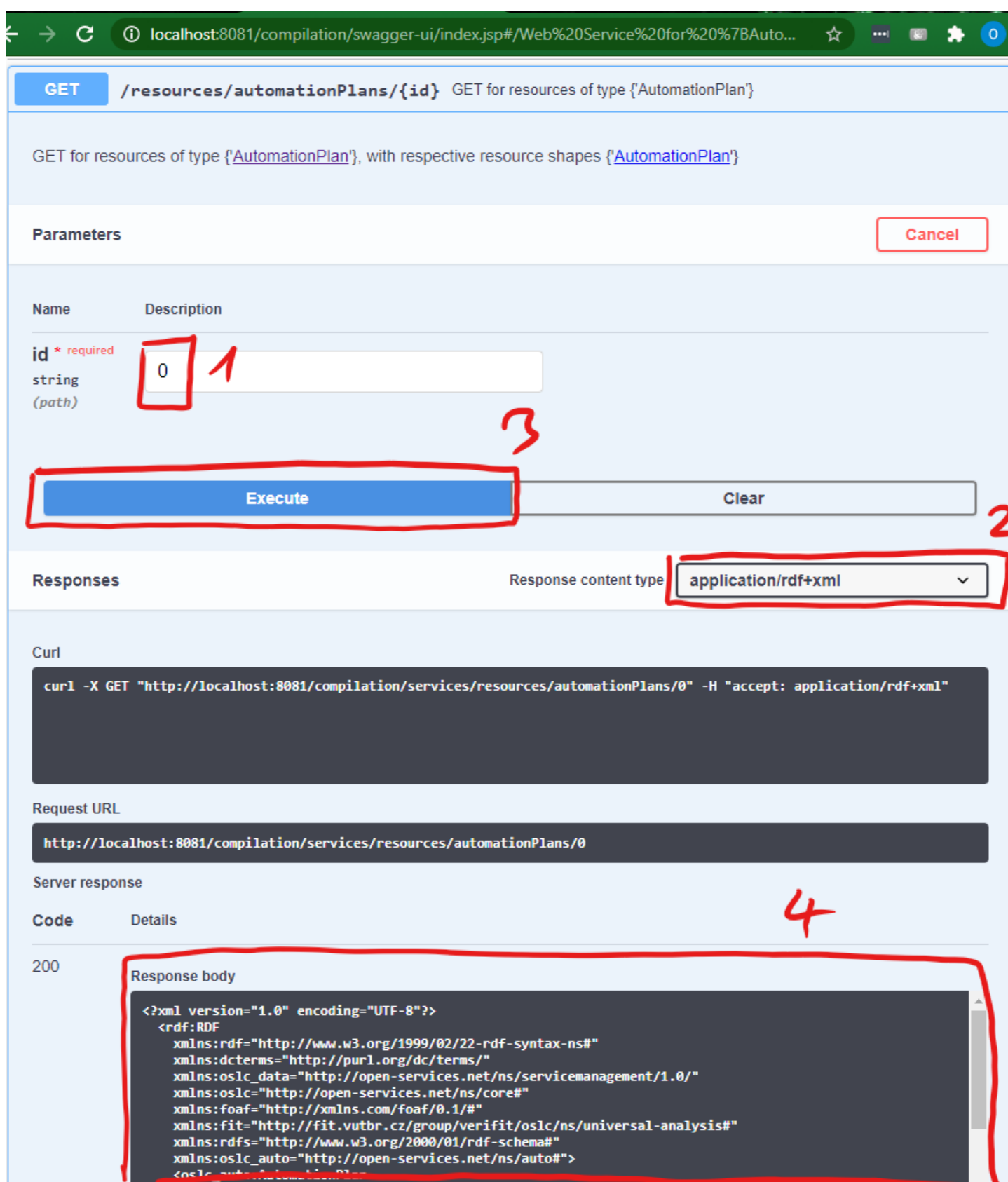
Open the Compilation adapter [home page](#). And navigate to the [swagger UI](#).



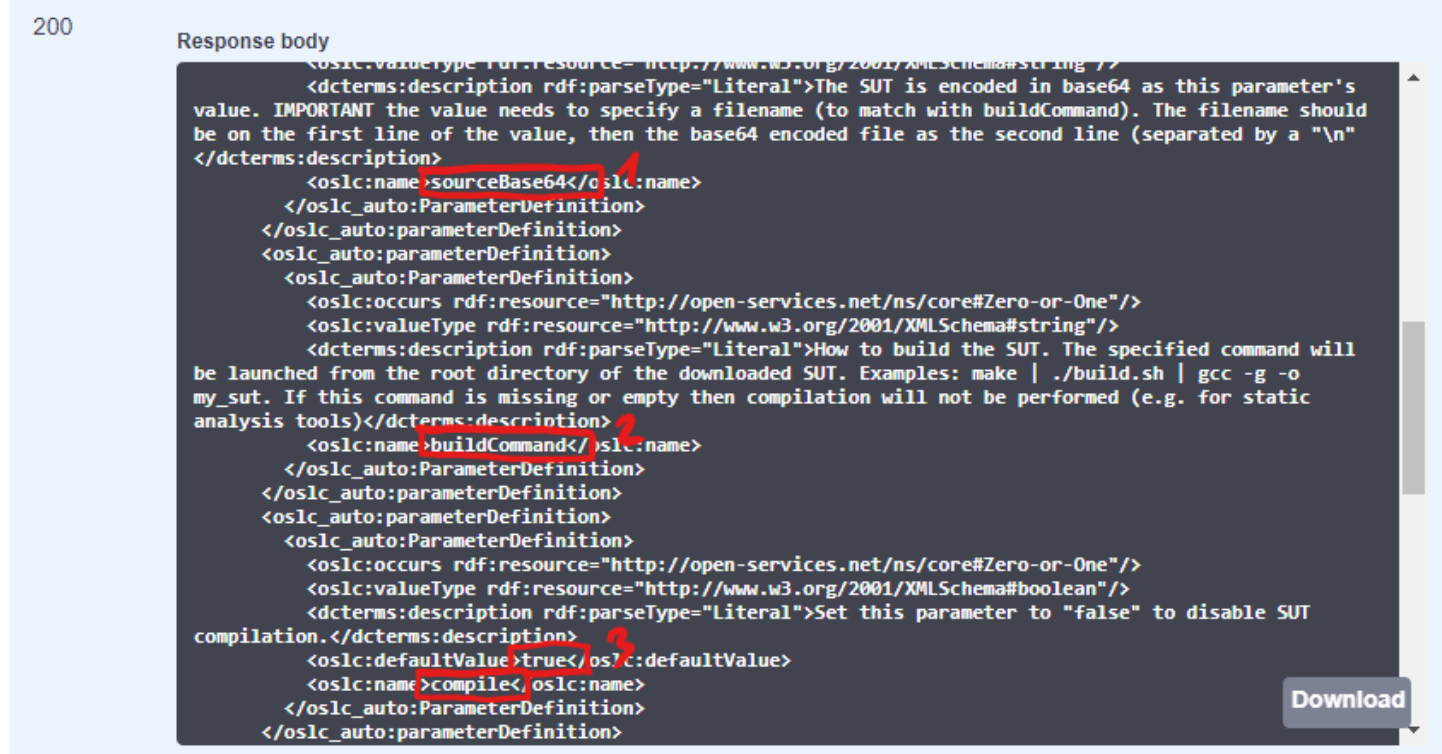
Find and expand the [Automation Plan GET](#) (or [alt](#)) capability (1) and click the "try it out" button (2).



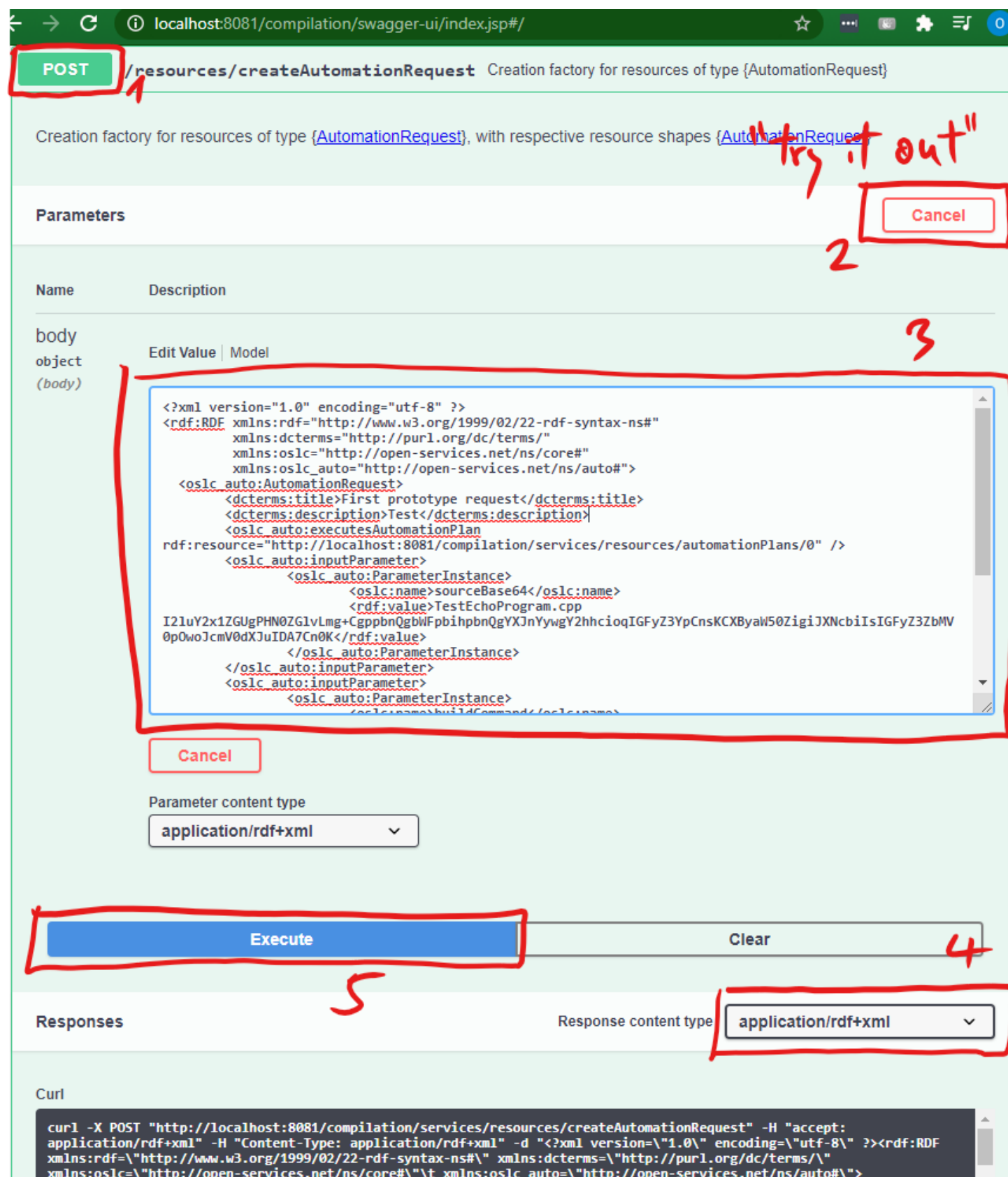
Set the ID to 0 (zero) (1) and pick your desired content type (2). Execute the request (3) and then look at the result (4) to learn about input parameters accepted by the Automation Plan which are defined using ParameterDefinition properties. The Automation Plan will look something like [this](#).



Input parameters will include a number of source* parameters (1), a build command and a launch command for the SUT (2), and other functional parameters like the compile parameter (3) which has a default value of true.



Next find and expand the [Automation Request POST](#) capability (1) and again click "try it out" (2). Paste your Automation Request into the input box (3), select your desired content type (4), and execute the request (5). The Automation Request needs to contain input parameters base on the executed Automation Plan and have a link to the executedAutomationPlan. Input parameters in our case are an SUT source (sourceBase64), an SUT launchCommand and an SUT buildCommand. Try this [example request](#)



The response to your request will look something like [this](#). The created Automation Request will have a URI with a numerical ID at the end (1). And its most important property for us now is a link to the producedAutomationResult (2). Notice that the numerical ID of the A.Result is the same as of the A.Request this can be used as a shortcut for finding A.Results.

Server response

Code	Details
201	<p>Response body</p> <pre> <?xml:namespace prefix="oslc_auto"="http://open-services.net/ns/auto#" /> <oslc_auto:AutomationRequest rdf:about="http://localhost:8081/compilation/services/resources/automationRequest/1"> <dcterms:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime" >2021-04-26T14:03:12.975Z</dcterms:created> <oslc_auto:inputParameter> <oslc_auto:ParameterInstance> <rdf:value>gcc -g TestEchoProgram.cpp -o my_echo</rdf:value> <oslc:name>buildCommand</oslc:name> </oslc_auto:ParameterInstance> </oslc_auto:inputParameter> <dcterms:identifier>1</dcterms:identifier> <dcterms:modified rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime" >2021-04-26T14:03:12.975Z</dcterms:modified> <oslc_auto:inputParameter> <oslc_auto:ParameterInstance> <rdf:value>./my_echo</rdf:value> <oslc:name>launchCommand</oslc:name> </oslc_auto:ParameterInstance> </oslc_auto:inputParameter> <oslc_auto:executesAutomationPlan rdf:resource="http://localhost:8081/compilation/services/resources/automationPlans/0"/> <oslc_auto:desiredState rdf:resource="http://open-services.net/ns/auto#complete"/> <oslc_auto:producedAutomationResult rdf:resource="http://localhost:8081/compilation/services/resources/automationResults/1"/> <oslc_auto:state rdf:resource="http://open-services.net/ns/autoInProgress"/> <oslc_auto:inputParameter> <oslc_auto:ParameterInstance> <rdf:value>TestEchoProgram.cpp </pre> <p>Download</p>

Now find and expand the [Automation Result GET](#) (or [alt](#)) capability (1) and again click "try it out" (2). Set the ID to match the ID of your Automation Result (3) (from the producedAutomationResult property in the previous screenshot), select your desired content type (4), and execute the request (5).

GET /resources/automationResults/{id} GET for resources of type {AutomationResult}

GET for resources of type {AutomationResult}, with respective resource shapes {AutomationResult}

Parameters

Name	Description
id * required	
string (path)	

Execute Clear

Responses

Response content type: application/rdf+xml

The most important properties right now are the state (2) and verdict (3) properties which say if the SUT creation was finished and if it was successful. The A.Result also contains a backward link to its associated A.Request (1). The A.Result will look something like [this](#).

Code

Code	Details
200	<p>Response body</p> <pre> <oslc_auto:producedByAutomationRequest rdf:resource="http://localhost:8081/compilation/services/resources/automationRequests/1"/> <oslc_auto:contribution> <oslc_auto:Contribution> <oslc:valueType rdf:resource="http://localhost:8081/compilation/services/resourceShapes/SUT"/> <dcterms:title rdf:parseType="Literal">SUT</dcterms:title> <dcterms:description rdf:parseType="Literal">Created SUT resource. Also linked to by the createdSUT property.</dcterms:description> <rdf:value>http://localhost:8081/compilation/services/resources/SUTs/1</rdf:value> </oslc_auto:Contribution> </oslc_auto:contribution> <dcterms:modified rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime" >2021-04-26T14:03:17.968Z</dcterms:modified> <oslc_auto:state rdf:resource="http://open-services.net/ns/auto#complete"/> <oslc_auto:inputParameter> <oslc_auto:ParameterInstance> <rdf:value>gcc -g TestEchoProgram.cpp -o my_echo</rdf:value> <oslc:name>buildCommand</oslc:name> </oslc_auto:ParameterInstance> </oslc_auto:inputParameter> <oslc_auto:outputParameter> <oslc_auto:ParameterInstance> <rdf:value>false</rdf:value> <oslc:name>unpackZip</oslc:name> </oslc_auto:ParameterInstance> </oslc_auto:outputParameter> <dcterms:identifier>1</dcterms:identifier> <oslc_auto:verdict rdf:resource="http://open-services.net/ns/auto#passed"/> </pre> <p>Download</p>

The A.Result will contain logs of the SUT creation process.

```
<oslc_auto:contribution>
  <oslc_auto:Contribution>
    <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <dcterms:title rdf:parseType="Literal">statusMessage</dcterms:title>
    <dcterms:description rdf:parseType="Literal">Status messages from the adapter about the execution.
  </dcterms:description>
  <rdf:value>SUT fetch successful
Executing: gcc -g TestEchoProgram.cpp -o my_echo
as: powershell.exe ./adapter/exec_compilation_1.ps1 ""
In dir: SUT\1
Compilation completed successfully
SUT resource created
</rdf:value>
</oslc_auto:Contribution>
</oslc_auto:contribution>
```

And most importantly the A.Result will contain a link to the createdSUT

CodeDetails

200

Response body

```
<oslc_auto:contribution>
  <oslc_auto:Contribution>
    <fit:filePath>C:\power42\matrix\universal-analysis-
adapter\compilation\SUT\1\adapter\stdout_compilation_1</fit:filePath>
    <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <dcterms:title rdf:parseType="Literal">stdout</dcterms:title>
    <dcterms:description rdf:parseType="Literal">Standard output of the compilation.
  </dcterms:description>
  <rdf:value>
    </rdf:value>
  </oslc_auto:Contribution>
</oslc_auto:contribution>
<oslc_auto:contribution>
  <oslc_auto:Contribution>
    <fit:filePath>C:\power42\matrix\universal-analysis-
adapter\compilation\SUT\1\adapter\stderr_compilation_1</fit:filePath>
    <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <dcterms:title rdf:parseType="Literal">stderr</dcterms:title>
    <dcterms:description rdf:parseType="Literal">Error output of the compilation.
  </dcterms:description>
  <rdf:value>
    </rdf:value>
  </oslc_auto:Contribution>
</oslc_auto:contribution>
<fit:createdSUT rdf:resource="http://localhost:8081/compilation/services/resources/sUTs/1"/>
<oslc_auto:outputParameter>
  <oslc_auto:ParameterInstance>
    <rdf:value>true</rdf:value>
  </oslc_auto:ParameterInstance>
  <oslc:name>compile</oslc:name>
```

Download

To retrieve the created SUT resource navigate to the [SUT GET capability](#) (or [alt](#)) (1), click (2), set the ID based on the ctedSUT property from the last screenshot (3), pick your content type (4), execute the request (5), and look at the SUT resource in the received response (6).

GET

/resources/sUTs/{id}

GET for resources of type {SUT}

GET for resources of type {SUT}, with respective resource shapes {SUT}

Parameters

Cancel

Name	Description
id * required string (path)	

Execute

Clear

Responses

Response content type

application/rdf+xml

Curl

curl -X GET "http://localhost:8081/compilation/services/resources/sUTs/1" -H "accept: application/rdf+xml"

Request URL

http://localhost:8081/compilation/services/resources/sUTs/1

Server response

Code

Details

200

Response body

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dcterms="http://purl.org/dc/terms/"
 xmlns:oslc_data="http://open-services.net/ns/servicemanagement/1.0/"
 xmlns:oslc="http://open-services.net/ns/core#"
 xmlns:foaf="http://xmlns.com/foaf/0.1/"
 xmlns:fit="http://fit.vutbr.cz/group/verifit/oslc/ns/universal-analysis#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:oslc_auto="http://open-services.net/ns/auto#">

The SUT resource will have a URI (1) which needs to be remembered by the client in order to be used as an input parameter when requesting analysis. It will have properties which are then used by the analysis adapter such as the build command (2) and the launch command (4), a compilation flag (3), and a directory (5). The SUT resource will look something like [this](#).

Code

Details

200

Response body

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dcterms="http://purl.org/dc/terms/"
 xmlns:oslc_data="http://open-services.net/ns/servicemanagement/1.0/"
 xmlns:oslc="http://open-services.net/ns/core#"
 xmlns:foaf="http://xmlns.com/foaf/0.1/"
 xmlns:fit="http://fit.vutbr.cz/group/verifit/oslc/ns/universal-analysis#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:oslc_auto="http://open-services.net/ns/auto#">
 <fit:SUT rdf:about="http://localhost:8081/compilation/services/resources/sUT/1">
 <fit:buildCommand rdf:parseType="Literal">gcc -g TestEchoProgram.cpp -o my_echo</fit:buildCommand>
 <dcterms:identifier>1</dcterms:identifier>
 <dcterms:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
 2021-04-26T14:03:17.856Z</dcterms:created>
 <oslc_auto:producedByAutomationRequest
 rdf:resource="http://localhost:8081/compilation/services/resources/automationRequests/1"/>
 <fit:compiled rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
 true</fit:compiled>
 <dcterms:modified rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
 2021-04-26T14:03:17.856Z</dcterms:modified>
 <fit:launchCommand rdf:parseType="Literal">./my_echo</fit:launchCommand>
 <fit:SUTdirectoryPath rdf:parseType="Literal">C:\power42\matrix\universal-analysis-
 adapter\compilation\SUT\1</fit:SUTdirectoryPath>
 <dcterms:title rdf:parseType="Literal">SUT - First prototype request</dcterms:title>
 </fit:SUT>
</rdf:RDF>

Download

2) Execute Analysis

Open the Analysis adapter [home page](#). And navigate to the [swagger UI](#).

VeriFit Analysis

This is a homepage of the *VeriFit Analysis* that was generated using Eclipse Lyo Toolchain Designer.

Service Provider Catalog

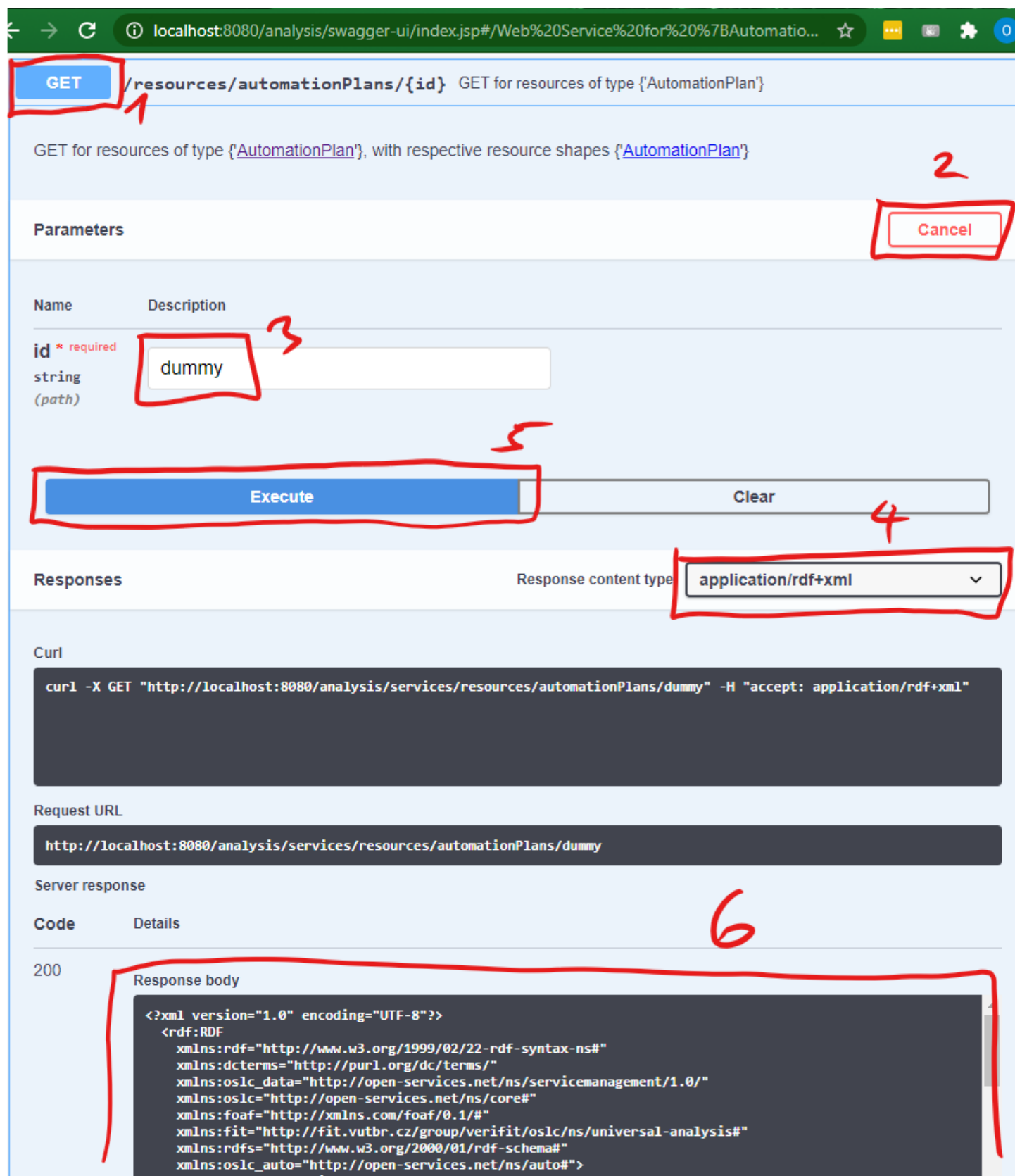
Start from the Service Provider Catalog, to navigate your adaptor's services and resources, using the available Query capabilities, Selection and Creation Dialogs. Note that these are end-user HTML pages, which is very useful for debugging your adaptor.

Interactive Swagger UI

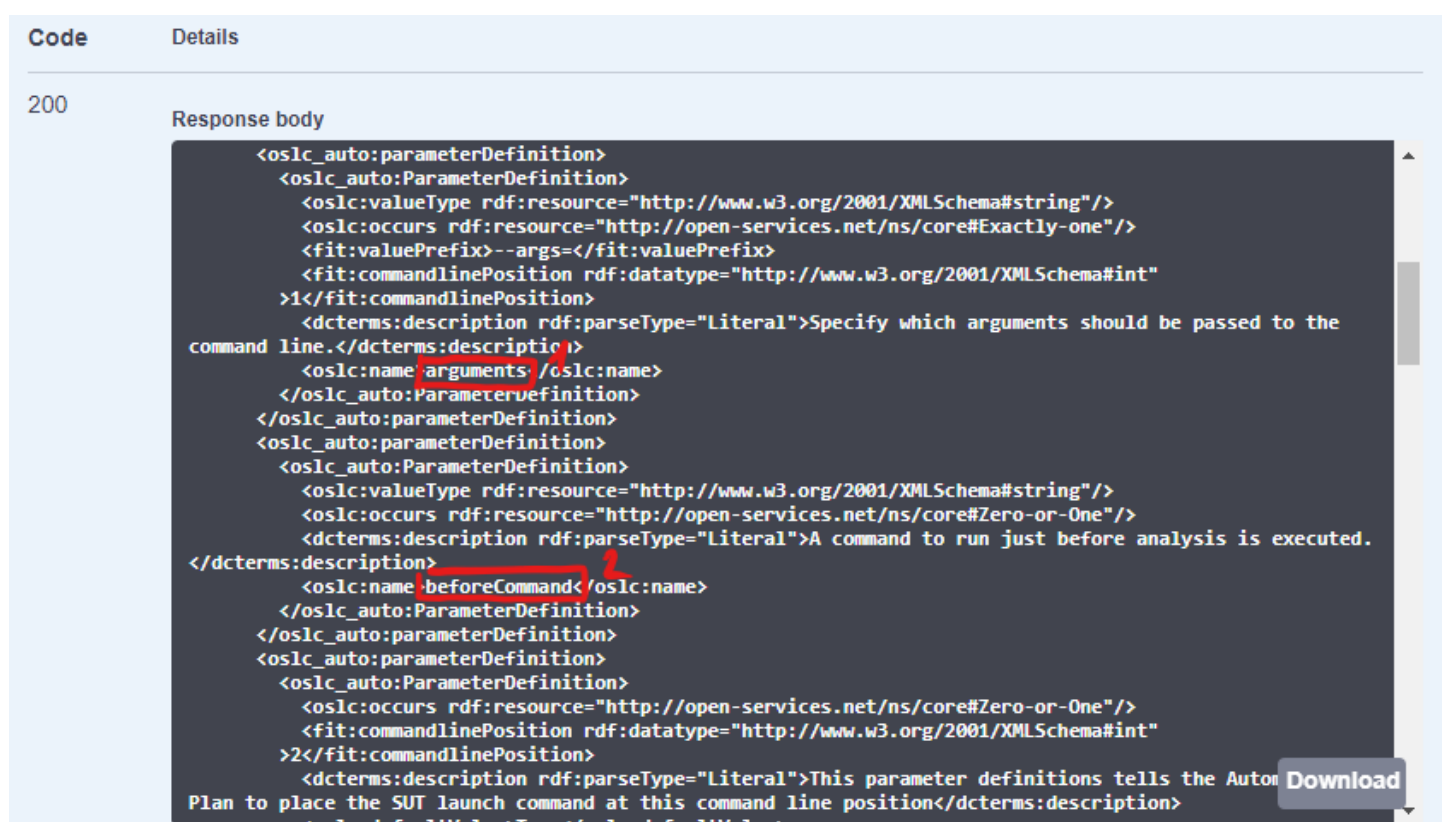
Use Swagger UI To interact with the adaptor services dedicated for RDF.

You can also copy [this OpenAPI specification document \(yaml file\)](#) of [this adaptor](#) to a [Swagger Editor](#) to generate client SDK code for a number of languages and platforms.

Find and expand the [Automation Plan GET](#) (or [alt](#)) capability (1) and click the "try it out" button (2). Set the ID to the ID of your analysis tool (3) and pick your desired content type (4). Execute the request (5) and then look at the result (6) to learn about input parameters accepted by the Automation Plan which are defined using ParameterDefinition properties. The Automation Plan will look something like [this](#)



Parameter Definitions will match the ones defined by your for the analysis tool (e.g. 1) and there will be common adapter input parameters as well (e.g. 2).



Next find and expand the [Automation Request POST](#) capability (1) and again click "try it out" (2). Paste your Automation Request into the input box (3), set the content type of the request body (4), select your desired content type (5), and execute the request (6). The Automation Request needs to contain input parameters base on the executed Automation Plan and have a link to the executedAutomationPlan. Input parameters in our case are a link to an SUT resource to be analysed and arguments for the analysis tool. Try this [🔗 example request](#)

POST /resources/createAutomationRequest Creation factory for resources of type {AutomationRequest}

Creation factory for resources of type {AutomationRequest}, with respective resource shapes {AutomationRequest}

Parameters

Name Description

body
object
(body)

Edit Value | Model

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:oslc="http://open-services.net/ns/core#"
  xmlns:oslc_auto="http://open-services.net/ns/auto#">

  <oslc_auto:AutomationRequest>
    <dcterms:title>Tutorial title</dcterms:title>
    <dcterms:description>Tutorial description</dcterms:description>
    <oslc_auto:executesAutomationPlan
      rdf:resource="http://localhost:8080/analysis/services/resources/automationPlans/dummy" />
    <dcterms:creator rdf:resource="Title-creator"/>

    <oslc_auto:inputParameter>
      <oslc_auto:ParameterInstance>
        <oslc:name>arguments</oslc:name>
        <rdf:value>HelloWorld</rdf:value>
      </oslc_auto:ParameterInstance>
    </oslc_auto:inputParameter>

    <oslc_auto:inputParameter>

  </oslc_auto:AutomationRequest>

</rdf:RDF>
```

Cancel

Parameter content type
application/rdf+xml

Execute Clear

Responses
Response content type
application/rdf+xml

The response to your request will look something like [this](#). The created Automation Request will have a URI with a numerical ID at the end (1). And its most important property for us now is a link to the producedAutomationResult (3). Other important properties include the desired state (2), state (4), and your input parameters (5). Notice that the numerical ID of the A.Result is the same as of the A.Request this can be used as a shortcut for finding A.Results.

Code Details

201

Response body

```

rdf:about="http://localhost:8080/analysis/services/resources/automationRequests/1"
  <oslc_auto:desiredState rdf:resource="http://open-services.net/ns/auto:complete">
  <dcterms:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
    >2021-04-26T16:05:01.341Z</dcterms:created>
  <oslc_auto:executesAutomationPlan
    rdf:resource="http://localhost:8080/analysis/services/resources/automationPlans/dummy"/>
  <oslc_auto:inputParameter>
    <oslc_auto:ParameterInstance>
      <rdf:value>HelloWorld</rdf:value>
      <oslc:name>arguments</oslc:name>
    </oslc_auto:ParameterInstance>
  </oslc_auto:inputParameter>
  <dcterms:description rdf:parseType="Literal">Tutorial description</dcterms:description>
  <dcterms:identifier>1</dcterms:identifier>
  <oslc_auto:inputParameter>
    <oslc_auto:ParameterInstance>
      <rdf:value>http://localhost:8081/compilation/services/resources/sUTs/1</rdf:value>
      <oslc:name>SUT</oslc:name>
    </oslc_auto:ParameterInstance>
  </oslc_auto:inputParameter>
  <dcterms:title rdf:parseType="Literal">Tutorial title</dcterms:title>
  <dcterms:modified rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
    >2021-04-26T16:05:01.341Z</dcterms:modified>
  <dcterms:creator rdf:resource="file:///C:/power42/matrix/universal-analysis-adapter/analysis/Title-creator"/>
  <oslc_auto:producedAutomationResult
    rdf:resource="http://localhost:8080/analysis/services/resources/automationResults/1">
  <oslc_auto:state rdf:resource="http://open-services.net/ns/auto:InProgress">

```

Download

Now find and expand the [Automation Result GET](#) (or [alt](#)) capability (1) and again click "try it out" (2). Set the ID to match the ID of your Automation Result (3) (from the producedAutomationResult property in the previous screenshot), select your desired content type (4), and execute the request (5). The result will be shown in (6).

GET /resources/automationResults/{id} GET for resources of type {AutomationResult}

GET for resources of type {AutomationResult}, with respective resource shapes {AutomationResult}

Parameters

Name	Description
id * required	
string	
(path)	

Execute Clear

Responses Response content type application/rdf+xml

Curl

```
curl -X GET "http://localhost:8080/analysis/services/resources/automationResults/1" -H "accept: application/rdf+xml"
```

Request URL

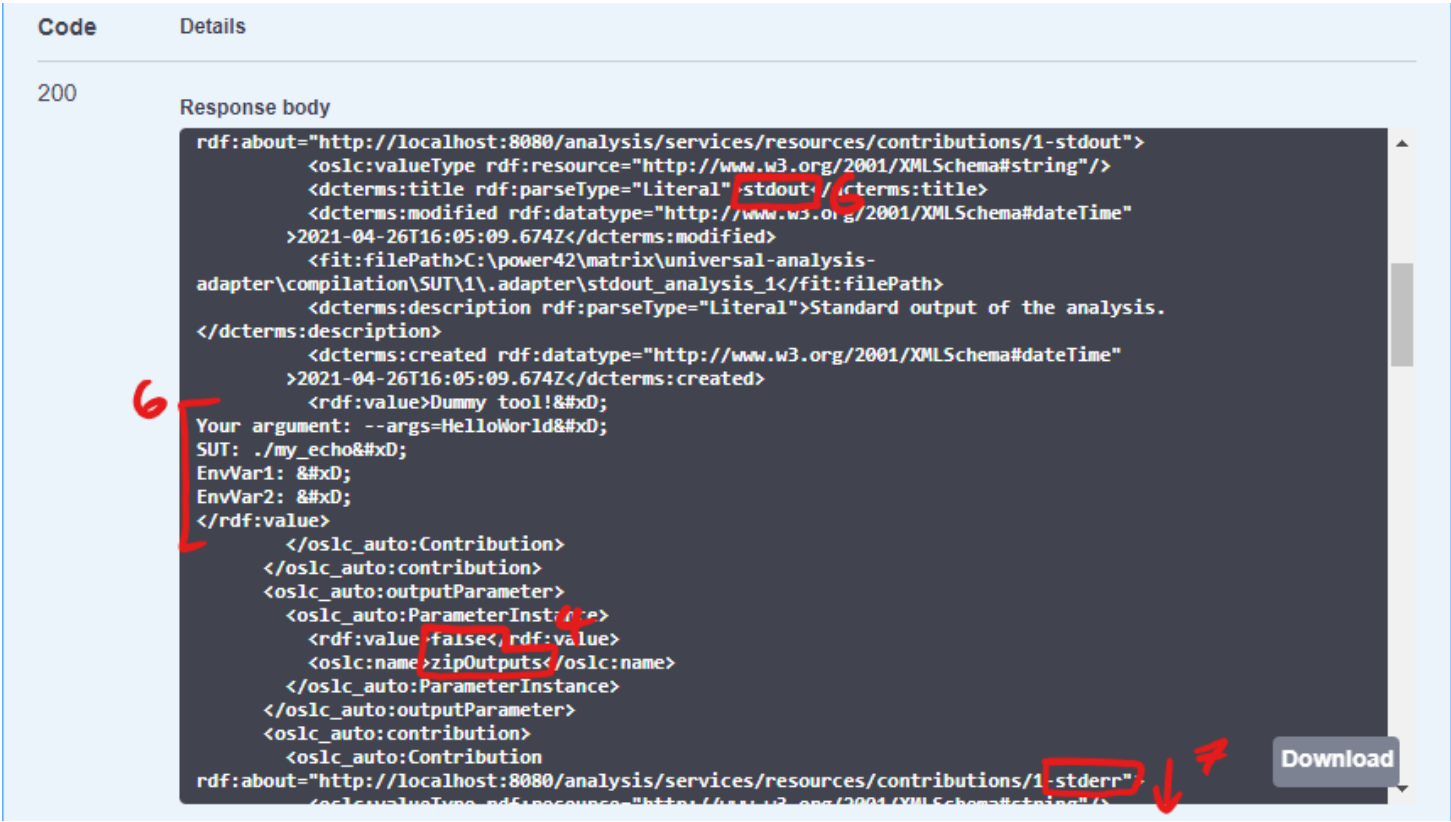
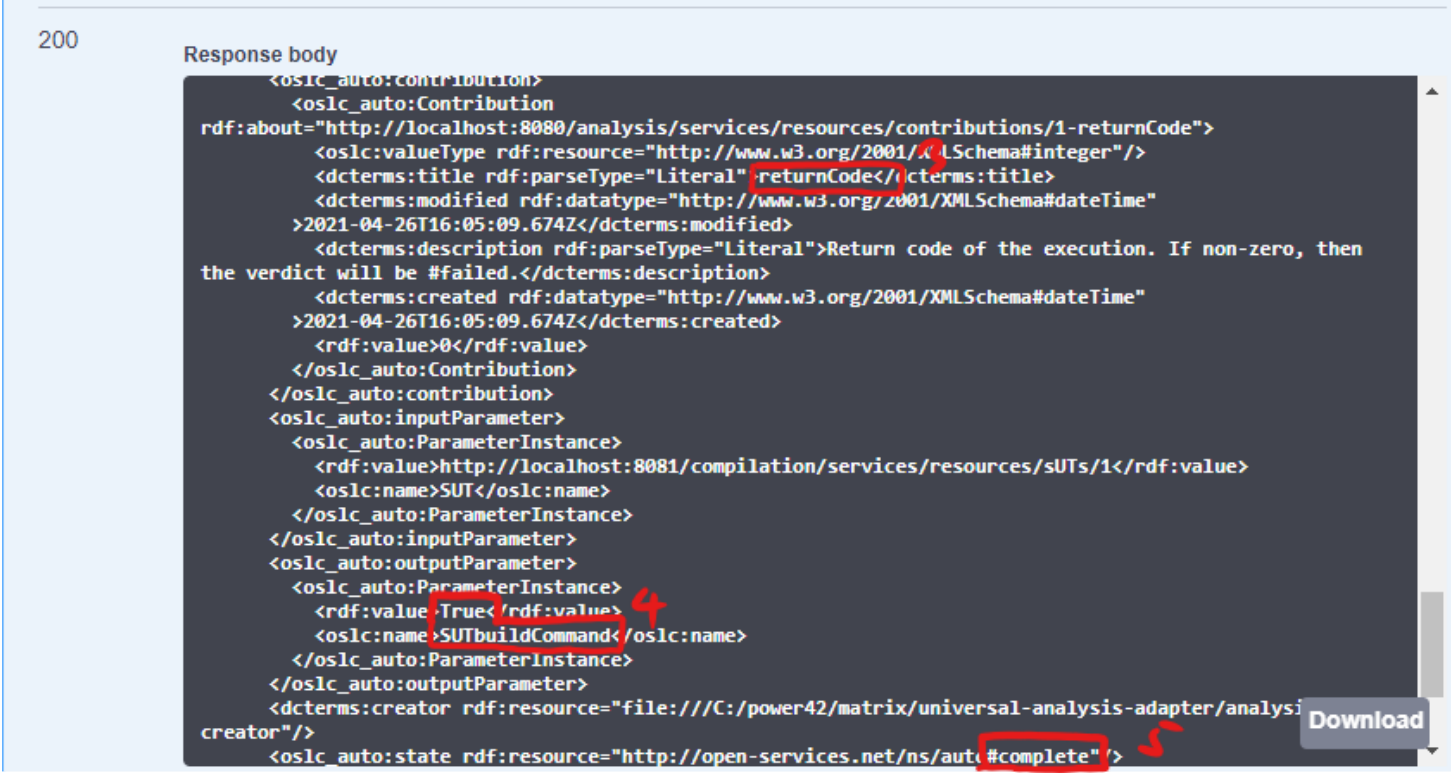
```
http://localhost:8080/analysis/services/resources/automationResults/1
```

Server response

Code	Details
200	<p>Response body</p> <pre><?xml version="1.0" encoding="UTF-8"?> <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:oslc_data="http://open-services.net/ns/servicemanagement/1.0/" xmlns:oslc="http://open-services.net/ns/core#" xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:fit="http://fit.vutbr.cz/group/verifit/oslc/ns/universal-analysis#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:oslc_auto="http://open-services.net/ns/auto#"></pre>

The A.Result will look something like [this](#). The most important properties right now are the state (5) and verdict (1) properties which say if the SUT creation was finished and if it was successful. Other important properties are output parameters which inform the client about parameters the adapter used on its own with default values (4). There will be status messages from the adapter about the execution process (2), standard output logs of the analysis (6, 7), return code of the analysis (3), and potential other contributions depending on the tool and the input parameters used.

Code	Details
200	<p>Response body</p> <pre><dcterms:description rdf:parseType="Literal">Total execution time of the analysis in milliseconds.</dcterms:description> <dcterms:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2021-04-26T16:05:09.674Z</dcterms:created> <rdf:value>6490</rdf:value> </oslc_auto:Contribution> </oslc_auto:contribution> <oslc_auto:verdict rdf:resource="http://open-services.net/ns/auto#">passed</oslc_auto:contribution> <oslc_auto:Contribution rdf:about="http://localhost:8080/analysis/services/resources/contributions/1-statusMessage"> <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/> <dcterms:title rdf:parseType="Literal">statusMessage</dcterms:title> <dcterms:modified rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2021-04-26T16:05:09.674Z</dcterms:modified> <dcterms:description rdf:parseType="Literal">Status messages from the adapter about the execution.</dcterms:description> <dcterms:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2021-04-26T16:05:09.674Z</dcterms:created> <rdf:value>Executing analysis: C:\power42\matrix\universal-analysis-adapter\analysis\tests\resources\dummy_tool.ps1 --args=HelloWorld ./my_echo gcc -g TestEchoProgram.cpp -o my_echo as: powershell.exe ./adapter/exec_analysis_1.ps1 "" In dir: C:\power42\matrix\universal-analysis-adapter\compilation\SUT\1 Analysis completed successfully File Contributions added </rdf:value> </oslc_auto:Contribution> </oslc_auto:contribution></pre>



3) Links to example files for quick access

Compilation / SUT creation

- [Automation Plan GET response](#)
- [Automation Request POST](#)
- [Automation Request POST response](#)
- [Automation Result GET response](#)
- [SUT GET response](#).

Analysis

- [Automation Plan GET response](#)
- [Automation Request POST](#)
- [Automation Request POST response](#)
- [Automation Result GET response](#)