OpenAPI 3

This page lists all supported **OpenAPI 3** elements and rules.

Supported OpenAPI 3 versions are 3.0.0, 3.0.1, 3.0.2, and 3.0.3. The OpenApi version 3.1 is not supported at the moment.

Supported REST Service Elements And Rules

Services are defined using the following elements:

Element	Supported by Connector	Description	More Information at ...
`openapi`	✓	Supported versions are 3.0.0, 3.0.1, 3.0.2, and 3.0.3. 3.1.0 is not supported.	
`info`	✓	Provides metadata about the API.	
`title`	✓	Used as name for the REST interface. This is the name of the connector. If you create a connector with the same title it will replace the connector.	
`description`	✗	Used as documentation for the REST interface.	
`termsOfService`	✗	A URL to the Terms of Service for the API. It must be in the format of a URL.	
`contact`	✗	Contact information for the exposed API.	
`license`	✗	License information for the exposed API.	
`version`	✗	Version of the OpenAPI document (which is distinct from the OpenAPI Specification version or the API implementation version).	
`server`	✓	The host (name or IP) serving the API.	
`url`	✓	A URL to the target host. Only the first **url** will be used. The URL will be split to the following attributes of the REST alias: - Protocol - Port - Host - Base Path If the URL is relative, then **Host** is empty. The REST adapter only supports HTTP and HTTPS.	REST Adapter Reference
`description`	✗	An optional string describing the host designated by the URL.	
`variables`	✗	The variables will not be expanded.	
`produces`	✗	A list of MIME types the APIs can produce. The REST adapter only parses JSON and XML.	

On this Page:

Related Documentation:

- OpenAPI Specification

<table>
<tr><td>paths</td><td>✓</td><td>Available paths and operations for the API. REST resource classes are created to reproduce each paths structure.
Paths that contain encoded characters are imported like follows:
<table>
<tr><th>Encoded Characters</th><th colspan="2">Replaced By</th></tr>
<tr><td><b>: (colon) %3A (encoded colon)</b></td><td>class name</td><td>_ (underscore)</td></tr>
<tr><td></td><td>relative path</td><td><code>::</code></td></tr>
<tr><td><b><a parameter>={<a parameter value>}</b></td><td>parameter</td><td>curly brackets ignored, result is <a <code>parameter>=<a parameter value></code></td></tr>
</table>
</td><td></td></tr>
<tr><td>$ref</td><td>✗</td><td>Allows for an external definition of this path item.</td><td></td></tr>
<tr><td>summary</td><td>✗</td><td>Optional string summary, intended to apply to all operations in this path.</td><td></td></tr>
<tr><td>description</td><td>✗</td><td>Optional string description, intended to apply to all operations in this path.</td><td></td></tr>
<tr><td>server</td><td>✗</td><td>Alternative <code>server</code> array to service all operations in this path.</td><td></td></tr>
<tr><td>methods</td><td>✓</td><td>HTTP methods defined for this path. A REST operation is created for each methods. Supported HTTP methods are <b>get</b> , <b>put</b>, <b>post</b>, <b>delete</b>, <b>options</b>, <b>head</b>, <b>patch</b>, <b>trace</b>.</td><td></td></tr>
<tr><td>tags</td><td>✗</td><td>List of tags for API documentation control.</td><td></td></tr>
<tr><td>summary</td><td>✓</td><td>Short summary of what the operation does. If the description if empty, the summary is used as documentation for the operation.</td><td></td></tr>
<tr><td>description</td><td>✓</td><td>Verbose explanation of the operation behavior. The description is used as documentation for the operation.</td><td></td></tr>
<tr><td>externalDocs</td><td>✗</td><td>Additional external documentation for this operation<b>.</b></td><td></td></tr>
<tr><td>operationId</td><td>✗</td><td>Unique string used to identify the operation<b>.</b></td><td></td></tr>
<tr><td>requestBody</td><td>✓</td><td>Request body applicable for this operation.</td><td></td></tr>
<tr><td>description</td><td>✗</td><td>Brief description of the request body.</td><td></td></tr>
<tr><td>content</td><td>✓</td><td>Content of the request body. The key is a media type or media type range and the value describes it.</td><td>Media Type Object</td></tr>
<tr><td>required</td><td>✗</td><td>Determines if the request body is required in the request.</td><td></td></tr>
<tr><td>produces</td><td>✓</td><td>List of MIME types the operation can produce. As the REST adapter only support JSON and XML if produces is defined and none of these are in the list the responses are ignored.</td><td></td></tr>
<tr><td>parameters</td><td>✓</td><td>List of parameters of parameter reference that are applicable for this operation. A REST parameter is created for each parameter object.</td><td>Parameter Object</td></tr>
<tr><td>responses</td><td>✓</td><td>List of possible responses as they are returned from executing this operation. An output parameter is created for the default response status code (201 for POST, 200 for the others).</td><td></td></tr>
</table>

			`description`	✓	Short description of the response. The description is used as a documentation for the parameter or the usage.	
			`content`	✓	Definition or definition reference of the response structure. ⓘ The adapter does not support primitives as response therefore a responses of these types are ignored.	Media Type Object
			`headers`	✓	List of headers that are sent with the response. Will be generated as out parameters.	
			`links`	✗	Map of operations links that can be followed from the response.	
		`callbacks`		✗	Map of possible out-of band callbacks related to the parent operation.	
		`deprecated`		✗	Declares this operation to be deprecated.	
		`security`		✗	Declaration of which security schemes are applied to this operation.	
		`servers`		✗	Alternative `server` array to service this operation.	
	`parameters`			✗	List of parameters that are applicable to all the operations described under this path.	
`components`				✓	Holds a set of reusable objects for different aspects of the OpenAPI specification.	
	`schemas`			✓	List to hold data types produced and consumed by operations. A class is created for each schema object.	Schema Object
	`responses`			✓	List to hold responses that can be used across operations.	
	`parameters`			✓	List to hold parameters that can be used across operations.	Parameter Object
	`examples`			✗	List to hold parameters that can be used across operations.	
	`requestBodies`			✓	List to hold requestBodies that can be used across operations.	
	`headers`			✓	List to hold headers that can be used across operations.	
	`securitySchemes`			✗	List to hold securitySchemes that can be used across operations.	
	`links`			✗	List to hold links that can be used across operations.	
	`callbacks`			✗	List to hold callbacks that can be used across operations.	

Supported REST Parameter Elements And Rules

Element	Used By Connector	Description	More Information at ...
`name`	✓	The name of the parameter. Parameter names are case sensitive. • If **in** is "path", the **name** field must correspond to the associated path segment from the path. • For all other cases, the **name** corresponds to the parameter name based on the **in** property.	See Path Templating for further information.
`in`	(✓)	The location of the parameter. Possible values are "query", "header" or "path". ⓘ "cookie" is not supported by the importer and ignored	
`description`	✓	A brief description of the parameter	

`required`	✗	Determines whether this parameter is mandatory.	
deprecated	✗	Specifies that a parameter is deprecated and SHOULD be transitioned out of usage.	
`allowEmptyValue`	✗	Sets the ability to pass empty-valued parameters.	
`style`	✗	Describes how the parameter value will be serialized depending on the type of the parameter value.	
`explode`	✗	When this is true, parameter values of type `array` or `object` generate separate parameters for each value of the array or key-value pair of the map.	
`allowReserved`	✗	Determines whether the parameter value SHOULD allow reserved characters.	
`schema`	✓	The schema defining the type used for the parameter.	Schema Object
`example`	✗	Example of the parameter's potential value.	
`examples`	✗	Examples of the parameter's potential value.	

Supported REST Media Type Elements and Rules

Element	Used By Connector	Description	More Information at ...
`schema`	✓	The schema defining the type used for the parameter.	Schema Object
`example`	✗	Example of the parameter's potential value.	
`examples`	✗	Examples of the parameter's potential value.	
`encoding`	✗	A map between a property name and its encoding information.	

Supported REST Schema Elements And Rules

Element	Supported By Connector	Description	More Information at ...
`$ref`	✓	A reference to a schema defined in the definitions list. the schema object is replaced by the definition. ⓘ If the schema reference defines a schema object without any properties, and the `type` of this object is a simple type (not **object**), the importer generates this simple type to the parameter (see description of `type` below).	

<table>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td><code>type</code></td><td>✓</td><td>The type of the object.
<table>
<tr><th>OpenAPI Type</th><th colspan="2">Base Type</th></tr>
<tr><td><b>integer</b></td><td colspan="2">Integer</td></tr>
<tr><td><b>number</b></td><td colspan="2">Float</td></tr>
<tr><td><b>boolean</b></td><td colspan="2">Boolean</td></tr>
<tr><td rowspan="4"><b>string</b></td><td colspan="2">Type depends on the format:</td></tr>
<tr><td>byte or binary</td><td>Blob</td></tr>
<tr><td>date-time or date</td><td>• PAS 23.1.1 DateTime
• PAS 23.1 String</td></tr>
<tr><td>Other formats</td><td>String</td></tr>
<tr><td><b>array</b></td><td colspan="2">If used as type of a parameter or property, it will be an array of <b>items</b> (see below).</td></tr>
<tr><td><b>object</b></td><td colspan="2">A class and properties are created regarding the allOf and properties values.</td></tr>
</table>
ⓘ If <code>type</code> is not given at all, and the definition has no properties, it will be imported as <code>urn:Base Types.Any</code>.</td><td></td></tr>
<tr><td><code>description</code></td><td>✓</td><td>Description of the class or property.</td><td></td></tr>
<tr><td><code>format</code></td><td>✓</td><td>The <b>format</b> of the object is used to be more specific when the type is <b>string</b>.</td><td></td></tr>
<tr><td><code>items</code></td><td>✓</td><td>The <b>item</b> is a schema object to defined the type of element in the array. Only used if type is <b>array</b>.</td><td></td></tr>
<tr><td><code>allOf</code></td><td>✓</td><td>List of schema object. The resulting class will contain all properties of all listed schema objects.</td><td></td></tr>
<tr><td><code>oneOf</code></td><td>✗</td><td>Type Any is used.</td><td></td></tr>
<tr><td><code>anyOf</code></td><td>✗</td><td>Type Any is used.</td><td></td></tr>
<tr><td><code>not</code></td><td>✗</td><td>Type Any is used.</td><td></td></tr>
<tr><td><code>properties</code></td><td>✓</td><td>List of properties. Each property is define by a name and a schema object. A property is created on the resulting class for each properties defined.</td><td></td></tr>
<tr><td><code>additionalProperties</code></td><td>✗</td><td>This should allow to define a map/dictionary.</td><td></td></tr>
<tr><td><code>default</code></td><td>✗</td><td>The default value represents what would be assumed by the consumer of the input as the value of the schema if one is not provided.</td><td></td></tr>
<tr><td><code>multipleOf</code></td><td>✗</td><td></td><td></td></tr>
<tr><td><code>maximum</code></td><td>✗</td><td></td><td></td></tr>
<tr><td><code>exclusiveMaximum</code></td><td>✗</td><td></td><td></td></tr>
<tr><td><code>minimum</code></td><td>✗</td><td></td><td></td></tr>
<tr><td><code>exclusiveMinimum</code></td><td>✗</td><td></td><td></td></tr>
<tr><td><code>maxLength</code></td><td>✗</td><td></td><td></td></tr>
<tr><td><code>minLength</code></td><td>✗</td><td></td><td></td></tr>
<tr><td><code>pattern</code></td><td>✗</td><td>This string must be a valid regular expression, according to the Ecma-262 Edition 5.1 regular expression dialect.</td><td></td></tr>
<tr><td><code>maxItems</code></td><td>✗</td><td></td><td></td></tr>
</table>

`minItems`	❌		
`uniqueItems`	❌		
`maxProperties`	❌		
`minProperties`	❌		
`required`	❌		
`enum`	❌		
`nullable`	❌	A `true` value adds `"null"` to the allowed type specified by the `type` keyword, only if `type` is explicitly defined within the same Schema Object.	
`discriminator`	❌	Adds support for polymorphism. The discriminator is an object name that is used to differentiate between other schemas which may satisfy the payload description.	
`readOnly`	❌	Declares the property as read only. This means that • it may be sent as part of a response • but should not be sent as part of the request.	
`writeOnly`	❌	Declares the property as write only. This means that • it may be sent as part of a request • but should not be sent as part of the response.	
`xml`	❌	May be used on properties schemas only. It has no effect on root schemas. Adds additional metadata to describe the XML representation of this property.	
`externalDocs`	❌	Additional external documentation for this schema.	
`example`	❌	Example.	
`deprecated`	❌	Specifies that a schema is deprecated and should be transitioned out of usage.	