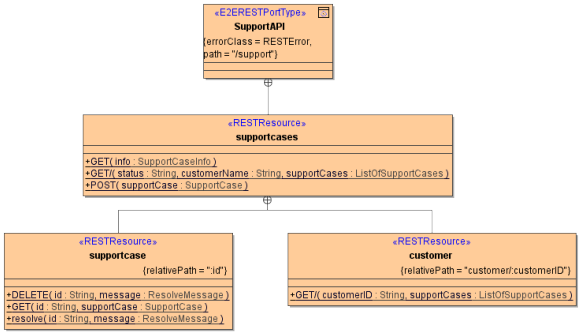
Defining a REST Service Interface

Wanting to implement a REST interface to a service, you first have to figure out the resource structure. Have a look at the structure of the rest example:

Figure: Example Class Diagram of a REST Interface



The rest interface **SupportAPI** has the following structure:

Resource	Methods	
supportcases that accept GET, GET/ and POST.	**GET**	Get some information on the existing support cases.
	GET/	Get all support cases.
	POST	Create a new support case.
Underneath the **supportcases**, there is a single **supportcase** that can be accessed via its **id**.	**DELETE**	Close a support case.
	GET	Get the data of a single support case.
	resolve	Set the status of the support case to "resolved".
Underneath the **supportcases** as well, there is a **customer** that can be accessed via its **customerID**.	**GET/**	Get all support cases of that specific customer.

From the example, you can see implementations of GET, POST and DELETE methods. Of course, you can use the other available methods with the Bridge, as there are PUT, PATCH, HEAD and OPTIONS.

> REST resources are generated to the OpenAPI file with their **class name only**, instead of their fully qualified name (including the xUML package structure, like `urn:Services.Classes.MyRESTResource`). This implicates that their names have to be unique throughout the REST interface. The compiler will report an error, if it encounters REST resources having the same name in different packages.

Defining the REST Port Type

A **REST Port Type** is a class having stereotype <<E2ERESTPortType>>. A REST port type can be deployed just like any other xUML service. It has the following tagged values:

Tagged Value	Description	Allowed Values	
Path (path)	Defines the path to this rest interface. If empty, the path is derived from the package structure.	none	path of the package structure will be used, e.g. `/Services/SupportCase/SupportAPI`
		any valid path string	path string starting with "/", e.g. `/support`

On this Page:

- Defining the REST Port Type
  - Example
- Defining REST Resources
  - Examples
- Defining REST Methods
  - Examples
- Defining REST Parameters
  - Examples
- REST Errors
  - Default Error Class
  - Specific Error Classes

Related Pages:

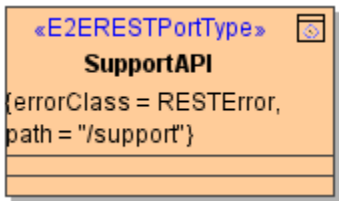
- Implementing REST methods
- REST Service Reference
- RESTful HTTP Service

Error Class (errorClass)	Assigns a user-defined <<RESTError>> class to the REST interface. This class should be set in case of error and given back via the REST response.	any complex type describing the structure of the error
Api Version (apiVersion)	Defines the API version this port type provides (for documentation purposes only).	any string

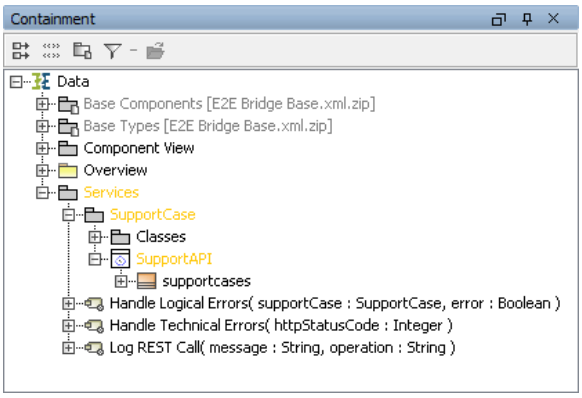
The REST port type can be added to the component diagram just as any other port type, e.g. SOAP port type:

Note the **trace port** that can be defined on the <<RESTService>> component. This is a shadow SOAP port that can be used to test the REST methods with the Analyzer.

Example



The <<E2ERESTPortType>> **SupportAPI** has path `/support` applied. The REST service can be accessed via `/support` instead of `/Services/SupportCase/SupportAPI` as depicted in the containment tree below.



It has a <<RESTError>> class **RESTError** applied as error class.

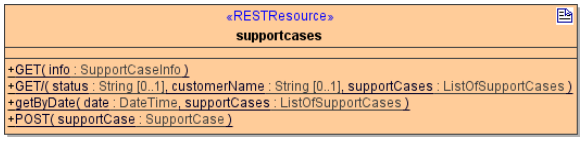
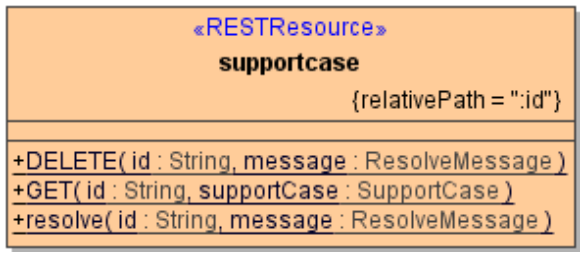
Defining REST Resources

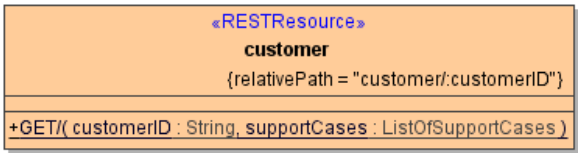
A **REST Resource** is a class having stereotype <<RESTResource>>. This stereotype represents both: collections of resources (e.g. **supportcases**) and single resources (**supportcase**). Both are handled indifferently by the Bridge. It is the modeler who should be aware, that some methods may not make sense on collections.

REST Resources have the following tagged values:

Tagged Value	Description	Allowed Values	
Relative Path (relativePath)	Defines the path of the REST resource or collection in relation to the parent resource . You can provide a static path, or a dynamic path using the notation `:<name of a REST Parameter>`. You may also provide a combination of both.	none	the name of the REST resource will be used, e.g. `/supportcases`
		any valid string	the given name will be used
		a dynamic path supplying a REST parameter	dynamic path, the value of the REST parameter will be passed to the REST methods, e.g. `:id`

Examples

Example REST Resource		Description
supportcases		REST resource **supportcases** has no relative path applied. It will be accessible via `/support/supportcases`. The first part of the URL is coming from the **path** value of the REST port type.
supportcase		REST resource **supportcase** has a dynamic path applied: **:id**. It will be accessible via `/support/supportcases/<a specific id>`, e.g. `/support/supportcase/1234`. **id** must be a REST parameter and accepted by all REST operations related to this resource. For more information on REST parameters refer to Defining REST Parameters.

customer		REST resource **customer** has a combined static and dynamic path applied: **customer/:customerID**. This is necessary to avoid conflicts with **supportcase**, which also has dynamic elements in its path. This resource will be accessible via `/support/supportcases/customer/<a specific customer id>`, e.g. `/support/supportcases/customer/0815`. **customerID** must be a REST Parameter and accepted by all REST operations related to this resource. For more information on REST parameters refer to Defining REST Parameters.

Defining REST Methods

A **REST Method** is an method having the stereotype <<REST>>. REST methods must be static.

> <<REST>> is the stereotype to apply to a REST method. Do not confuse with <<RESTOperation>>, which is used for RESTful HTTP services as described on RESTful HTTP Service.
> The latter approach is recommended only, if you want to use content types different to JSON and XML.

With REST methods, we distinct between **verb** methods and **named** methods.

- **Verb Methods**
  Verb-methods intercept requests issued directly to the resource. Unlike named methods, verb-methods cannot specify path parameters other than the ones defined by the parent resource(s). With the Bridge, you can use all available HTTP methods, as there are GET, POST, PUT, DELETE, PATCH, HEAD, and OPTIONS.
  Example: A `GET` on /support/supportcases will route to the **GET** method of class **supportcases** and give an overview on the existing support cases.
- **Named Methods**
  To call such method, append its name (or **relativePath**) to the parent resource.
  Example: A `PUT` on `/support/supportcases/1234/resolve` will route to the **resolve** method of class **supportcases**.

> **The trailing /**
>
> **Verb** methods (unlike normal methods) can be in form of `GET` or `GET/` - the difference is subtle but significant.
>
> Think about the support manager example.
>
> - Issuing a `GET` on `/support/supportcases/` is expected to return a list of existent support cases.
> - A `GET` on `/support/supportcases` is expected to return information on the support cases in general, e.g number of support cases, list of customers afflicted, ...

REST methods have the following tagged values:

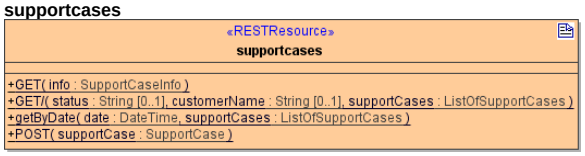
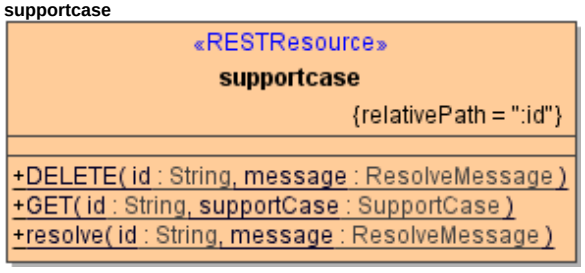
Tagged Value	Description	Allowed Values	
Http Method (httpMethod)	Provide the HTTP method of this REST method should respond to.	a valid HTTP method	GET, POST, PUT, DELETE, PATCH, HEAD, OPTIONS

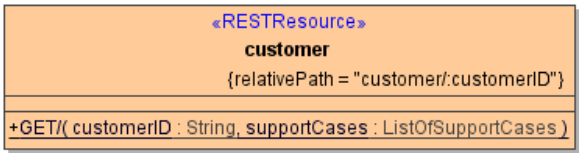
		none	• method name, if it is one of: GET, POST, PUT, DELETE, PATCH, HEAD, OPTIONS (with optional trailing '/') • GET otherwise
Relative Path (relativePath)	Defines the path of the REST method in relation to the parent resource.	none	The name of the REST method will be used.
		any valid string	The given name will be used. The relative path may also contain variables (REST path parameters, specified as `:<variable name>`) and can be segmented like e.g. `/date=:<a date variable>`.
Is Verbatim Path (isVerbatimPath)	This a REST Adapter setting and has no effect on REST service.		
Blob Body Content Type (blobBodyContentType)	Specify a default content type for **Blob** response parameters from this endpoint. This must be a list of valid media ranges as defined in RFC 7231. This information will be generated to the OpenAPI descriptor file (response content type). Refer to Handling Blobs in the REST Interface for a deeper explanation and some examples. This tag must be left unset if no **Blob** output parameters are used. In future versions, the effect of this tag may be extended to other contexts as well.	a list of valid media ranges	e.g. `application/msexcel` Default is `application/octet-stream` if not specified.
Reject Other Response Content Type (rejectOtherResponseContentTypes)	Runtime 2021.6 Builder 7.15.0 The xUML Runtime performs a verification of the content-type header for REST responses. Specify whether to return an error (HTTP 406, not acceptable) on responses with a content type that does not conform with the content types specified in **Blob Body Content Type**. Any mismatch will be logged to the service log on log level **Debug**. Refer to Handling Blobs in the REST Interface for a deeper explanation and some examples.	**true**	• Return HTTP 406 (Not Acceptable, default). • Service log: `RESTLM/47: Client does not accept any of declared response content types` This exception can be suppressed by setting **Ignore Http Errors** to true on the REST adapter alias.
		false	• Accept the request in spite of the mismatch and handle this within the service. • Service log (**Debug**): `RESTLM/10: Cannot generate any of the expected output formats`
Accepted Request Content Type (acceptedRequestContentType)	Runtime 2021.6 Builder 7.15.0 Provide a list of content types this REST endpoint accepts. This must be a list of valid media ranges as defined in RFC 7231. This information will be generated to the OpenAPI descriptor file (parameter content type). Refer to Handling Blobs in the REST Interface for a deeper explanation and some examples. This tag must be left unset if no **Blob** output parameters are used. In future versions, the effect of this tag may be extended to other contexts as well.	a list of valid media ranges	e.g. `application/xhtml+xml` Default is `application/octet-stream` if not specified.

Reject Other Request Content Types (rejectOtherRequestContentTypes)	Runtime 2021.6 Builder 7.15.0 Specify whether to return an error on requests with a content type that does not conform with the content types specified in **Accepted Request Content Type**. Any mismatch will be logged to the service log on log level **Debug**. Refer to Handling Blobs in the REST Interface for a deeper explanation and some examples.	**true**	• Return HTTP 415 (Unsupported Media Type) if the request content type of a **Blob** input parameter does not match the requirements (default). • Service log (**Debug**): `RESTLM/10: Cannot generate any of the expected output formats` This exception can be suppressed by setting **Ignore Http Errors** to true on the REST adapter alias.
		false	• Perform the adapter call in spite of the "content-type" header mismatch and handle this within the service. • Service log: `RESTLM/48: Request content type not declared as accepted by the service`

If the method name is one of GET, POST, PUT, DELETE, PATCH, HEAD, OPTIONS (with optional trailing '/'), it will be invoked automatically on its parent resource when an corresponding request is received.

Examples

Example REST Resource	Method		Descripton
supportcases 	**GET** and **GET/**	verb method	Both methods have no **httpMethod** applied as GET is the default method. They will be invoked, when accessed via a `GET` on `/support/supportcases` or `/support/supportcases/`.
	getByDate	named method	This method has **httpMethod** GET applied. It will be invoked on a `GET` on `/support/supportcases/date=<a valid date>`.
	POST	verb method	This method has **httpMethod** POST applied. It will be invoked on a `POST` on `/support/supportcases`.
supportcase 	**DELETE**	verb method	This method has **httpMethod** DELETE applied. It will be invoked, when accessed via a `DELETE` on `/support/supportcases/<a support case id>`, because its parent resource has a relative path :id applied.
	GET	verb method	This method has no **httpMethod** applied as GET is the default method. It will be invoked, when accessed via a `GET` on `/support/supportcases/<a support case id>`.
	resolve	named method	This method has **httpMethod** PUT applied. It will be invoked, when accessed via a `PUT` on `/support/supportcases/<a support case id>/resolve`.

customer  	**GET/**	verb method	This method has no **httpMethod** applied as GET is the default method. It will be invoked, when accessed via a GET on `/support/supportcases/customer/<a customer id>`, because its parent resource has a relative path :customerID applied.

Defining REST Parameters

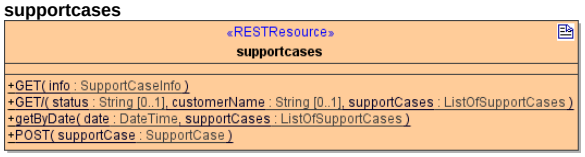
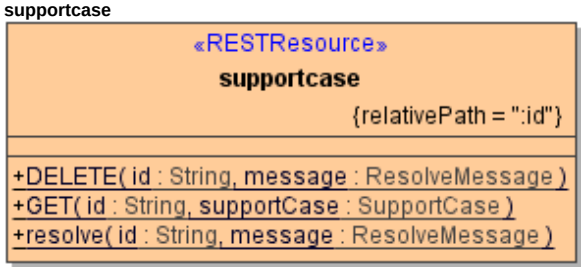
A **REST Parameter** is an input parameter of a <<REST>> method having the stereotype <<RESTParameter>>. This defines that this parameter will be provided via path, query, body, or header of the HTTP request. This has to be indicated on the parameter by setting tagged value in:

Tagged Value	Description	Allowed Values		Allowed REST Methods	Allowed Types	Hints and Limitations
External Name (externalName)	Defines an external name for the REST parameter	any string				Use this, when wanting to access a REST service that has parameter names with special characters. In this case, set this name (e.g. `ugly@parameter-name`) to **externalName** and give a better name. So you will not have to escape the parameter every time you use it.
In (in)	Defines how the parameter will be passed to the REST method. This tag is **mandatory**.	query	via a query string	all	all simple types and **Array** of simple type	Unknown parameters will be ignored, known will be passed to the method after being URL-decoded.
		path	via the REST resource path	all	**Integer**, **Float**, **String**, **Boolean**, **DateTime**	Path parameters are all required. All path parameters must be consumed by the called method and the parameter names must be the same as the path segment identifiers (without colon).
		body	via the REST call body	POST, PUT, PATCH	a complex type and **Array**	A REST method can have only one body parameter.
		header	via the REST call header	all	all simple types and **Array** of simple type	Unknown parameters will be ignored, known will be passed to the method.
Multiplicity (multiplicity)	Defines whether the parameter is required, or not.	0..1				Parameter is not required. Path parameters are always required.
		1				Parameter is required.

All path parameters are required. For all other parameters, use the **multiplicity** to specify whether they are required or not.

Examples

Example REST Resource	REST Parameter	Tagged Value "in"	Remark

supportcases  	status, customerName	**query**	**status** and **customerName** are provided via the query string: `/support/supportcases/?status=in%20progress`. In this case, the xUML Runtime will automatically assign the parameters coming with the query string to the REST parameters.
	supportCase	**body**	For posting a new support case, the support case data **supportCase** is provided through the HTTP **body**. In this case, the xUML Runtime will automatically assign the data from the embodied JSON or XML document to the REST parameter class.
supportcase  	id	**path**	REST resource **supportcase** has a dynamic path **:id** applied. For this reason, all methods of this resource must have a REST parameter with the same name **id** that will receive the value from the URL.

REST Errors

REST services in general return errors via the HTTP status code, so first of all, you should carefully choose the status code you are returning on a service call. Besides the HTTP status code there is no standard way of how to provide additional error information with REST service implementations. Developers can return additional information in HTTP headers or body, though.

With the Bridge REST implementation, we decided to provide error information via the HTTP body by an error class or a **Blob**.

Default Error Class

Each REST port type should have a default <<RESTError>> class assigned. The Bridge will use this class as a default output in case of error.

Figure: Example REST Error Class

In case of error, this class should be

- filled with some error information and
- assigned to the REST HTTP response (so the error information will be returned to the caller)

The xUML Runtime will recognize attributes as error code and/or error message under the following conditions:

- if you applied the names **code** and/or **message** to these attribute(s)
- if you applied the stereotypes <<RESTErrorCode>> and/or <<RESTErrorMessage>> to these attribute(s)

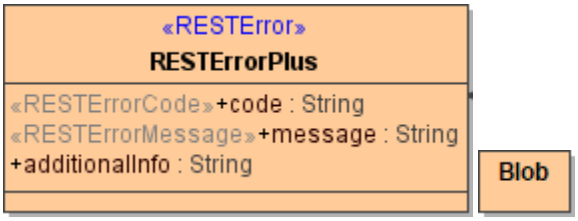
In this case, Runtime error codes and/or messages will automatically by assigned to these attributes in case of error.

Refer to Implementing REST Operations for more information on error handling.

Specific Error Classes

You can define specific error classes for specific HTTP errors to provide more information on the error, or just return a **Blob**.

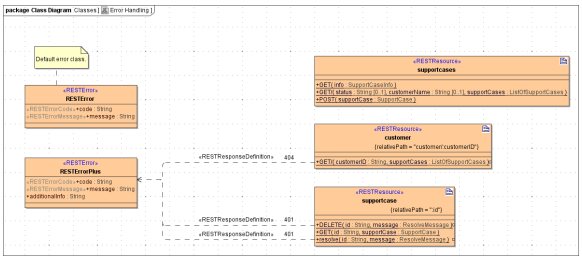
Figure: Specific Error Class



> Use this feature carefully. Having multiple error responses will possibly make your service confusing and will make it harder to implement service calls for a potential client.

There is no difference between using an error class or a **Blob**. Assign the specific error class to related operations or REST resources via a <<use>> dependency having stereotype <<RESTResponseDefinition>>.

Figure: Assigning Error Class to REST Operations



On these <<use>> dependencies, you have to specify an HTTP status code on the **name** tag. For this status code, the default error class will be overwritten by the specific class.

> You cannot overwrite HTTP response codes using REST Parameter classes (like e.g. 200).

You can apply the following name templates:

Example Name	Description	
`401`	**A specific status code.**	The specific error class will only be used, if exactly this HTTP status code is send back.
`40?, 4??`	**Defining a status code pattern.**	The specific error class will only be used, if the HTTP status code that is send back matches the pattern.
`???`	**All status codes.**	This pattern defines a new default error class for the resource or operation. The specific error class is valid for all HTTP status codes.

The definitions above are reflected in the OpenAPI service description (see REST Response Definitions).

> Response definitions using patterns (like e.g. 40? or 4??) can not be generated to the OpenAPI file, so it is not recommended to use them. A response definition having pattern ??? will be generated as **default** response of the operation. Refer to REST Response Definitions for more information on this.

For responses of type **Blob**, you can additionally specify a blob body content type on the <<RESTResponseDefinition>> (tag **blobBodyContentType**). This information will be generated to the OpenAPI descriptor file and will set the the "Content-Type" header to this content type. Default content type is "application/octet-stream".

Refer to Implementing REST Operations for more information on error handling.