**COAT**

**Cloud Orchestration Automation Templating**

Generate configuration files from anywhere with a web services front-end to Apache Velocity

**User Manual**

# Introduction & Overview

The Cloud Orchestration & Automation Template (COAT) engine is a web services front end to the Apache Velocity Engine. It is can be used to systematically generate text files from anywhere web services are available.

According to the Apache Velocity website,

The Apache Velocity Engine is a mature, free open-source templating engine. Velocity permits you to use a simple yet powerful template language to reference objects defined in Java code. It is written in 100% pure Java and can be easily embedded into your own applications.

COAT wraps that Apache Velocity engine in an XML-centric, web-services front-end. A COAT *template service* takes as input

* an Apache Velocity template
* an XML instance document containing configuration information
* one or more XML schemas (XSD) that the XML configuration conforms to

The cross-platform nature of Java allows COAT to run on a variety of computing platforms. In addition, the use of web services means that Apache Velocity services are now accessible from platforms in which Apache Velocity might not otherwise be deployable; for example due to technological or policy constraints. A central repository of template services also allows templates and configurations to be shared and managed more easily.

## Audience

This document is intended for software developers that are familiar consuming REST-based web services. Readers are assumed to have familiarity with XML, XML schema and the concept of a template engine; specific knowledge about Apache Velocity familiarity is helpful, but not required.

## Request for Feedback

In the spirit of continuous improvement, feedback on COAT is both welcomed and encouraged. NIST and the authors extend an open invitation to participate in the development of COAT by sending related comments to [xe-coat-comments@nist.gov](mailto:xe-coat-comments@nist.gov).

## Documentation Conventions

This document uses the following conventions.

### Quotations

If the inclusion of a period within a quotation might lead to ambiguity as to whether or not the period *should* be included in the quoted material, the period will be placed outside the trailing quotation mark. For example, a sentence that ends in a quotation would have the trailing period “inside the quotation, like this quotation punctuated like this.” However, a sentence that ends in a URL would have the trailing period outside the quotation mark, such as “http://example.com”.

### Machine-Readable Code

With the exception of some reference URLs, machine-readable information will typically be depicted with a mono‐spaced font, such as this.

## References

This document uses the [bracket] shorthand to refer to documents referenced in this manual.

|  |  |
| --- | --- |
| **[FizzBuzz]** | I. Ghory, *Using FizzBuzz to Find Developers who Grok Coding*. http://imranontech.com/2007/01/24/using-fizzbuzz-to-find-developers-who-grok-coding/ |
| **[W3C Charset]** | M. Dűrst. *Setting the HTTP charset parameter.* http://www.w3.org/International/O-HTTP-charset |
| **[XSDPart1]** | Henry Thompson et al., *XML Schema Part 1: Structures Second Edition*, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/, W3C Recommendation 28 October 2004. |
| **[XSDPart2]** | P. Biron, A. Malhotra, *XML Schema Part 2: Datatypes Second Edition,* http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/, W3C Recommendation 28 October 2004. |

# Design Concepts and Architecture

## Service Overview

A COAT server comprises a collection of *template services*. Each template service comprises document and functional resources.

*Document* resources are static records (documents) that are managed through the typical CRUD (create, read, update, delete) operations.

A fully functioning template service comprises the following *document* resources

* one *template* resource — an Apache Velocity template
* one or more schema resources — XML schemas that describe an XML configuration
* one or more *config* resources — XML files that contain configuration instances

Users of the COAT can, through these CRUD operations, create, modify, use, and remove template services to suit their purpose.

Services and document resources can be identified and referenced to by *name*. A *service name* uniquely identifies a service. A *service name* plus *resource name* uniquely identifies a particular resource.

*Functional* resources allow clients to perform a variety of non-CRUD operations. These functions are built-in to COAT and are not extensible at runtime.

* *process* resources run a template against a particular configuration
* *history* resources allows the retrieval of previous versions of the resource.
* *splitter* resources creates a template service, a configuration and a schema from an existing configuration file.[[1]](#footnote-1)
* *rename* resources allow document resources to be renamed.
* *upload* resources create a template, configuration, or schema resource from posted content.

The *rename* and *upload* functional resources are provided for convenience. Templates, configurations, and schemas may also be uploaded to COAT through their respective resource endpoints (e.g., a template can be uploaded by posting to a template resource). The *upload* resource is a convenience wrapper that provides a uniform endpoint and assigns posted content according to the designated filename. Likewise, the *rename* resource is a shorthand for the creation of new resources and deletion of the old.

## Walkthrough

The following is a walkthrough that is intended to give an overview of the COAT service.[[2]](#footnote-2)

In 2007, Imran Ghory [FizzBuzz] proposed the following litmus test to gauge a minimum competency for a programmer:

Write a program that prints the numbers from 1 to 100. But for multiples of three print “Fizz” instead of the number and for the multiples of five print “Buzz”. For numbers which are multiples of both three and five print “FizzBuzz”.

For the sake of illustrating COAT, let’s use the more general requirement, making the number of lines, multiples, and what is printed configurable. Denoting variables like ***this*,** our requirement becomes.

Generate a text file that lists the numbers from 1 to ***lines***; but for multiples of ***fizzFrequency***print ***fizzText*** instead of the number and for the multiples of ***b*** print ***buzzText***. For numbers which are multiples of both three and five print ***fizzText*** concatenated with ***buzzText*.**

If you were to use Apache Velocity alone to generate this file, it might look something like this[[3]](#footnote-3)

#foreach ($line in [1..${lines}])

#if ($line % ${fizzFrequency} == 0 &&

$line % ${buzzFrequency} == 0)

${fizzText}${buzzText}

#elseif ($line % ${fizzFrequency} == 0)

${fizzText}

#elseif ($line % ${buzzFrequency} == 0)

${buzzText}

#else

$line

#end

#end

This template, however, is only part of a larger necessary solution. First, to execute this template, the Apache Velocity engine needs to be instantiated and made available to a client. Second, the variables ***lines, fizzFrequency****,* ***fizzText, buzzFrequency,*** and ***buzzText*** need to be bound to specific values.

The COAT front end provides this (and other) functionality to arbitrary clients through web services and an XML processing layer. Hosting Velocity and providing a web-service front end allows access from any client that can “speak” the web. The XML processing layer allows a client to bind template variables to different sets of values (or *configuration*s). COAT can store different configurations on behalf of a client, or it can use a configuration provided by the client when the template is executed.

Let’s walk through how this works in practice by writing a simple COAT template service. Let’s assume we have

(a) the XML schema for FizzBuzz configuration documents:

**main.xsd**

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="FizzBuzz" type="FizzBuzzType"/>

<xs:complexType name="FizzBuzzType">

<xs:sequence>

<xs:element name="lines" type="xs:int"/>

<xs:element name="fizzText" type="xs:string"/>

<xs:element name="buzzText" type="xs:string"/>

<xs:element name="fizzFrequency" type="xs:int"/>

<xs:element name="buzzFrequency" type="xs:int"/>

</xs:sequence>

</xs:complexType>

</xs:schema>

(b) an XML document (conformant to the above schema) that reflects Ghory’s original problem statement,

**default.xml**

<FizzBuzz>

<lines>100</lines>

<fizzText>Fizz</fizzText>

<buzzText>Buzz</buzzText>

<fizzFrequency>3</fizzFrequency>

<buzzFrequency>5</buzzFrequency>

</FizzBuzz>

and, (c) the Apache Velocity template that we will use for processing.

**template**

#foreach ($line in [1..${ROOT.lines}])

#if ($line % ${ROOT.fizzFrequency} == 0 &&

$line % ${ROOT.buzzFrequency} == 0)

${ROOT.fizzText}${ROOT.buzzText}

#elseif ($line % ${ROOT.fizzFrequency} == 0)

${ROOT.fizzText}

#elseif ($line % ${ROOT.buzzFrequency} == 0)

${ROOT.buzzText}

#else

$line

#end

#end

This template is almost similar to original Apache Velocity template beside a difference: in the COAT template, we have been prefixing the variables defined in the configuration with *ROOT.* For example, *${ROOT.lines}* is used instead of *${lines}*. This is because the COAT XML processing layer convert the XML configuration into a Java object, where each XML element becomes a property (i.e., a pair of get() and set() method).

Once these three resources defined, we are in a position interact with the COAT service to create a template service and use it. An example is provided in the next section (see §2.2.1).

### Basic Template Lifecycle

A COAT server allows clients to create and consume individual template services. Suppose we have a new (i.e., empty) installation of COAT running at an endpoint, or *base URL* <http://coat/>. From start to finish, the complete template service lifecycle is (1) create a template service, (2) upload resources to that service and (3) request that COAT run that service’s template. Our worked example uses curl, but any HTTP client will do.

1. **Creating the Template.** To create a template service, simply perform a POST on the URL *{baseUrl}/{serviceName}*. In our case, we create the FizzBuzz template service with the command:

> curl –X POST http://coat/FizzBuzz

1. **Upload Resources**. Then, we upload our configuration file, *default.xml*, schema file (*main.xsd*) and template. (COAT does not require resources to be uploaded in any particular order).

> curl –X POST –H ″Content-Type:text/xml″ –d @default.xml <http://coat/FizzBuzz/config/default.xml>

> curl –X POST –H ″Content-Type:text/xml″ –d @main.xsd <http://coat/FizzBuzz/schema/main.xsd>

> curl –X POST –H ″Content-Type:text/plain″ –d @fizzbuzz.vm <http://coat/FizzBuzz/template>

Notice that:

* The structure of the URL determines the name of the service to upload to (FizzBuzz), the type of resource uploaded (config, schema, or template) and a resource name (*default.xml*, *main.xsd*). Because there can only be one template for a template service, we do not need to specify a template name.
* We specify a “*Content-Type*.” COAT requires ‘*text/xml*’ for XML files (schema and configuration XML) and ‘*text/plain’* for the Apache Velocity templates.
* We give the XML Schema the name *main.xsd*. A COAT service can have multiple schema files. However, COAT requires a *main.xsd* and that the *main.xsd* defines a single root element. As detailed elsewhere in the document, part of the COAT processing pipeline is translating XML instance documents (i.e., configurations) into objects to serve as input for the Apache Velocity template processing. This pipeline depends on being able to unambiguously generate the appropriate objects.

1. **Running the Template**. Run the template by performing a GET on the *{FizzBuzz}/process/{configName}* URL

> curl <http://coat/FizzBuzz/process/default.xml>

This yields

1⮨ 2⮨ Buzz⮨ 4⮨ Fizz⮨ Buzz⮨ 7⮨ 8⮨ Buzz⮨ Fizz⮨ 11⮨ Buzz⮨ 13⮨ 14⮨ FizzBuzz⮨ 16⮨…

where we’ve replaced a newline by the ⮨ symbol so we can show the results on a single line. The result line as been truncated in this document for readability.

By convention, if a name is not specified to a template service’s process URL, then COAT automatically assumes that the desired configuration name is ‘*default.xm*l’

> curl <http://coat/FizzBuzz/process>

### Additional Template Operations

That’s all there is to the fundamentals to running a COAT service. The remainder of the manual gives detailed information about each COAT resource and method, but here are examples of other useful operations.

1. **Running Different Configurations**. As mentioned previously, running a configuration can be done “by name.” This involves uploading a resource,

> curl –X POST –H ″Content-Type:text/xml″ –d @frobozz.xml <http://coat/FizzBuzz/config/frobozz.xml>

and then executing the template by adding the configuration name as a URL path parameter

> curl <http://coat/FizzBuzz/process/frobozz.xml>

Consequently, the contents of *frobozz.xml* are persisted by COAT. If for some reason, however, you would prefer to have COAT never persist the configuration, you can pass it in as the HTTP payload when executing the template. Specifically, the previous command has the same net effect as this:

> curl –X POST –d @frobozz.xml <http://coat/FizzBuzz/process>

The template is exercised with the provided configuration in a single step, but the configuration *frobozz.xml* isn’t stored.

1. **Delete a Resource.** To delete a resource, simply perform a DELETE on the URL that points to the resource. For the FizzBuzz template service, the configuration, schema and template can be deleted:

> curl –X DELETE <http://coat/FizzBuzz/config/default.xml>

> curl –X DELETE <http://coat/FizzBuzz/schema/main.xsd>

> curl –X DELETE <http://coat/FizzBuzz/template>

Once all the resources associated with a template service have been deleted, the template service resource can be deleted:

> curl –X DELETE <http://coat/FizzBuzz>

If the service still has resources associated with it, then trying to delete it will result in an error and generate a *ResourceNotEmptyException* that will be provided in the payload of the response.

1. **Get a Previous Version of a Resource.** COAT maintains a history of its stored resources and allows accessing the various versions of schemas, templates and configurations, and also getting the list of the various versions of a specific resource.

If the specific version of a resource is unknown, the first step consists in requesting the history list, for example the history list for the schema *main.xsd* of the FizzBuzz template service can be retrieved using this command:

> curl <http://coat/history/FizzBuzz/schema/main.xsd>

COAT returns the list of the versions of the template represented using the *ResourceHistory* type (see §3.3.6).

<resourceHistory>

<historicLink>

<serviceName>FizzBuzz</serviceName>

<rel>schema</rel>

<name>history/main.xsd</name>

<uri>http://coat/historic/1426623485028/FizzBuzz/schema/main.xsd</uri>

<sizeInBytes>78</sizeInBytes>

<milliseconds>1426623485028</milliseconds>

</historicLink>

<originalRel>schema</originalRel>

<originalURI>http://coat/FizzBuzz/schema/main.xsd</originalURI >

</resourceHistory>

The response contains the information of the current version of the schema described using the elements *originalRel* and *originalURI*. Each previous version of the schema is also listed using a *historicLink* resource.

For example, the previous version of the schema main.xsd can now be requested, as its URI is now known from the response of our previous request

> curl <http://coat/historic/1426623485028/FizzBuzz/schema/main.xsd>

1. **Rename a Service or Resource**. COAT offers a method to rename a service, a configuration or a schema. This method is provided for convenience as the same result can be achieved by creating a new resource and then deleting an old one.

Renaming the schema *main.xsd* to *custom.xsd* of Fizzbuzz is requested using the following command:

> curl –X POST <http://coat/rename/FizzBuzz/schema/main.xsd?newName=custom.xsd>

And confirmed by a message of *RenameResult* type (see §3.3.5) containing information about the current and former names of the resource.

<renameResult>

<newLink>

<serviceName>FizzBuzz</serviceName>

<rel>schema</rel>

<name>custom.xsd</name>

<uri>http://coat/FizzBuzz/schema/custom.xsd</uri>

<historyUri>http://coat/history/FizzBuzz/schema/custom.xsd</historyUri>

<renameUri>http://coat/rename/FizzBuzz/schema/custom.xsd</renameUri>

<isDeletable>true</isDeletable>

</newLink>

<oldLink>

<serviceName>FizzBuzz</serviceName>

<rel>schema</rel>

<name>main.xsd</name>

<uri>http://coat/FizzBuzz/schema/main.xsd</uri>

<historyUri>http://coat/history/FizzBuzz/schema/main.xsd</historyUri>

<renameUri>http://coat/rename/FizzBuzz/schema/main.xsd</renameUri>

<isDeletable>true</isDeletable>

</oldLink>

</renameResult>

This method is especially useful when renaming a service that already has resources associated with it as it recursively renames all of its resources.

1. **Split an INF/INI File**. COAT offers the capability to create a service, its template, a schema and a configuration as a one-time operation by providing a single file following the Microsoft INF/INI file structure.

> curl –X POST –H ″Content-Type:text/plain″ –d @fizzbuzz.ini <http://coat/splitter/ini?serviceName=FizzBuzz>

Once the operation has been successfully executed, it is also possible to execute the template right away.

> curl http://coat/FizzBuzz/process

The default configuration, schema and the template can also be retrieved. This is possible, as they have been inferred by COAT. From the INF/INI file that was provided at the service creation, COAT is able to create:

* a schema that describes the data structure
* a configuration that contains the list of the variables and their values
* a template that captures the structure of the result files generated after processing

1. **Use the “Upload” Resource.** Resources don’t have to be created solely with the structured URLs. A single URL can be used to provide a uniform endpoint.   
     
   Uploading a new configuration *config1.xml* for the *FizzBuzz* service can be achieved using:

> curl –X POST –H ″Content-Type:text/xml″ –d @config1.xml <http://coat/FizzBuzz/upload?rel=config&name=config1.xml>

Or this request:

> curl –X POST –H ″Content-Type:text/xml″ –d @config1.xml <http://coat/FizzBuzz/config/config1.xml>

There are differences in the responses of these two requests. The response on the *upload* request returns a message of type *ResourceList* (§3.3.7) in the payload if the request is successful.

<resourceList>

<link>

<serviceName>FizzBuzz</serviceName>

<rel>config</rel>

<name>config1.xml</name>

<uri>http://coat/FizzBuzz/config/config1.xml</uri>

<historyUri>http://coat/history/FizzBuzz/config/config1.xml</historyUri>

<renameUri>http://coat/rename/FizzBuzz/config/config1.xml</renameUri>

<isDeletable>true</isDeletable>

</link>

</resourceList>

As opposed to the response of a successful request that used the structured URL that contains no payload.

## Expected Usage

This section more formally describes the expected usage of COAT template services.

### Consumption Style

COAT services are accessible by both structured URLs as well has through the more RESTful *hypertext as the engine of application state* (HATEOAS) mechanism. Consumers of COAT services are encouraged to rely primarily on the HATEOAS, falling back to structured URLs if necessary. This decoupling of syntax and semantics allows for clients to better handle future versions or distributed variants of COAT. Regardless, both styles are supported in this initial version.

### Cardinality & Expected Flow

The following diagram simultaneously describes the cardinality of the relationships between the various COAT resources (how many of what kind of resource belongs to another), and the expected order in which clients will retrieve or activate the various resources. A double arrow represents a *one-to-many* relationship.

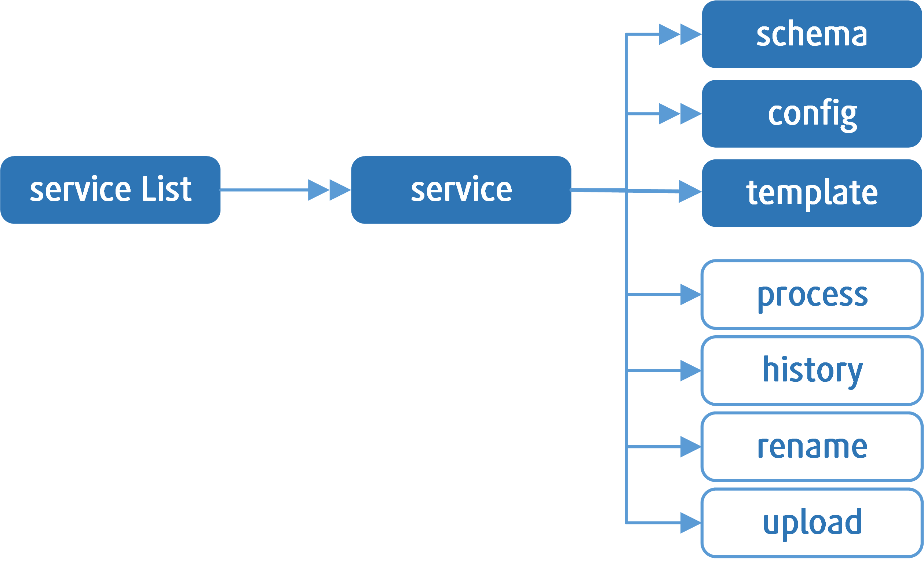


Figure Relationships between the COAT resources

COAT services follow the HATEOAS pattern: no prior knowledge is required to interact with the service. For example, the following request

> curl <http://coat/>

returns the list of all services available on the COAT server described using an XML message in the payload of the response. Each service is listed and described including its URI. By using this URI, it is possible to request the detailed description of the service from the COAT server. In the following example we are requesting the description of the *FizBuzz* template service using its URI retrieved from the service list.

> curl <http://coat/FizzBuzz>

The service description returned in the response contains the list of resources associated with the service. Each of these service resources has as well an URI that can be used to get a detailed information on the resource.

Not only the HATEOAS pattern allows to dynamically discovering services and their resources (described using *links*) but also permits to communicate which operations (described also using *links*) can be used once some constraints have been met. For example, a *process* link won’t be provided if the service is without a schema, default configuration and template. But if constraints are met, the link is provided in the resource description as it is shown below in this XML fragment.

<link>

<serviceName>FizzBuzz</serviceName>

<rel>process</rel>

<name>process</name>

<uri>http://coat/FizzBuzz/process</uri>

</link>

It should be noted that resources (template, configuration and schema) are associated with only one service and cannot be shared among services. If a COAT user wishes to “re-use” an existing template to create a new service, then the user needs first to retrieve the template from the existing service and then in a second step to post it as the template for the new service.

## Template Processing Pipeline

*Template* *processing* is the core function and purpose of COAT. Template processing may be initiated by a variety of mechanisms:

* **Default Processing**.
* **Named Processing**.
* **On Demand (Payload) Processing**.

The effective difference among these methods are (1) the URLs (structured or HATEOAS) that are used to trigger processing and (2) the parameters that are passed either explicitly or implicitly into the method.

The following diagram depicts the flow of information into, within, and finally out of a COAT template service.



Figure 2. Visualization of the COAT processing pipeline

A schema, configuration, and template serve as inputs and are shown in the diagram as solid, filled rectangles. The service output, which is the result of processing a template against the provided configuration, is depicted as the filled rectangle with rounded corners. Intermediate stages of processing are shown as unfilled rectangles. These stages are internal to the COAT service and are currently not exposed to service consumers.

Upon receiving a request to process a template, COAT generates Java source code from each available schema ➊ and compiles it into Java byte code ➋. If that succeeds, COAT then validates the configuration XML against the service’s schema(s) ➌. After validation, the service converts the XML into java objects through *deserialization* ➍. Next, COAT examines the main schema and looks for a single, unique root element ➎. Finally, the Apache Velocity template is processed against the configuration object ➏.

Understanding the flow of the processing step is useful when troubleshooting errors, since the nature of errors returned can give deeper insight to troubleshoot the stages that are failing. For example, if the XML validation fails, this implies that code generation and compilation were successful as the validation comes only once code generation has been successfully completed. Likewise, errors in template processing are not directly attributable to XML validation. (Naturally, this is not a guarantee that previous stages are free of defect and is only an indicator).

# Data Dictionary

This section contains descriptions of the data elements that comprise the COAT data model. Each data type is described via an accompanying XML Schema type definition.

If the instance of a data type is returned by a resource, then that resource will return an XML document that confirms to the accompanying document schema. Consequently, all documents that consist of a single instance of a single type will all share the same root element name.

## HTTP Headers

COAT uses custom HTTP Headers when returning responses. The information stored in the header is also available in the payload of the message under XML format. CORS is also supported by COAT and allows to access resources out of the domain of the COAT server.

### X-COAT Headers

The following table provides the list and a description of the customs HTTP headers are used by COAT.

|  |  |
| --- | --- |
| Field Name | Description |
| X-COAT-Exception-Message | The message associated with the exception that was thrown (See note 1). |
| X-COAT-Exception-Type | The name of the class that throw the exception result (See note 2). |
| X-COAT-Historic-Rel-Of | The type (i.e., *rel*ationship type) of the resource whose location is specified in *X-COAT-Historic-Version-Of.* |
| X-COAT-Historic-Version-Of | The location of the last version (i.e. most recent instance) of this resource. |
| X-COAT-New-Name | The current name of the resource after a renaming operation has been performed. |
| X-COAT-Old-Name | The former name of the resource after a renaming operation has been performed. |
| X-COAT-Rel | The type (i.e., *rel*ationship type) of the resource specified in *X-COAT-ResourceName.* |
| X-COAT-ResourceName | A label that describes or otherwise names the resource. |
| X-COAT-ServiceName | The name of the service to which the resource belongs. |
| X-COAT-Uploaded-Rel | The type (i.e., *rel*ationship type) of the resource that has been uploaded using the *upload* service method. |

**Notes**

1. This header field contains the same value as the XML element *classLiteral* provided in the response payload. More details are provided in §3.3.1.
2. This header field contains the same value as the XML element *message* provided in the response payload. More details are provided in §3.3.1.

### CORS Headers

The following table provides the list and a description of the CORS fields returned in the header of the response by the COAT server.

|  |  |
| --- | --- |
| Field Name | Description |
| Access-Control-Allow-Methods | A list of the HTTP methods allowed by the server (See note 1). |
| Access-Control-Allow-Origin | A list of the remote URLs (origins) that are allowed to be reached by the client browser and the COAT server. |
| Access-Control-Expose-Headers | A list of the custom HTTP field names that are made available by the COAT server (See note 2). |

**Notes**

1. Methods that could be allowed include GET, POST, OPTIONS, DELETE and HEADS.
2. Values from this field can be among the X-COAT-\* header field names described in §3.1.1.

## Namespaces

The following namespaces and corresponding namespace prefixes are used throughout this document.

|  |  |  |
| --- | --- | --- |
| Prefix | Namespace | Remarks |
| xs | http://www.w3.org/2001/XMLSchema |  |
| xsi | http://www.w3.org/2001/XMLSchema-instance |  |
| coat | http://coat.xe.nist.gov |  |

## Data Types

The following is an alphabetically ordered list of each of the data types that may be returned by COAT as an XML payload.

### Exception Result

An *exception result* provides information about a Java exception that was triggered when processing a client request. Example of requests triggering the generation of those messages includes using reserved name or illegal characters when creating a service, trying to use a resource that doesn’t exist, etc. Exception result instances conform to the XML Schema:

<xs:complexType name="ExceptionResultType">

<xs:sequence>

<xs:element name="classLiteral" type="xs:string" minOccurs="0"/>

<xs:element name="message" type="xs:string" minOccurs="0"/>

</xs:sequence>  
</xs:complexType>

|  |  |  |
| --- | --- | --- |
| Element | Type | Remarks |
| classLiteral | xs:string | The name of the class that throw the exception result. |
| message | xs:string | The message associated with the exception that was thrown. |

### Historic Link

A *historic link* provides a hypermedia pointer to a version of a resource. This type is an extension on the *link* type (described in §3.3.3) that provides two additional elements: the size of the link at its timestamp expressed in seconds since 1/1/1970 (Epoch time). Historic link instances conform to the XML Schema:

<xs:complexType name="HistoricLinkType">  
 <xs:complexContent>  
 <xs:extension base="LinkType">

<xs:sequence>

<xs:element name="sizeInBytes" type="xs:long" minOccurs="0"/>

<xs:element name="milliseconds" type="xs:long" minOccurs="0"/>

</xs:sequence>  
 </xs:extension>  
 </xs:complexContent>  
</xs:complexType>

|  |  |  |
| --- | --- | --- |
| Element | Type | Remarks |
| sizeInBytes | xs:long | The size (expressed in bytes) of this version of the resource. |
| milliseconds | xs:long | A timestamp represented using Epoch time, i.e. the number of seconds elapsed since 1/1/1970. |

### Link

A *link* provides a hypermedia pointer to another resource. Link instances conform to the XML Schema:

<xs:complexType name="LinkType">

<xs:sequence>

<xs:element name="serviceName" type="xs:string" minOccurs="0"/>

<xs:element name="rel" type="xs:string" minOccurs="0"/>

<xs:element name="name" type="xs:string" minOccurs="0"/>

<xs:element name="uri" type="xs:string" minOccurs="0"/>

<xs:element name="historyUri" type="xs:string" minOccurs="0"/>

<xs:element name="renameUri" type="xs:string" minOccurs="0"/>

<xs:element name="uploadUri" type="xs:string" minOccurs="0"/>

<xs:element name="isDeletable" type="xs:boolean" minOccurs="0"/>

</xs:sequence>  
</xs:complexType>

|  |  |  |
| --- | --- | --- |
| Element | Type | Remarks |
| serviceName | xs:string | The name of the service to which the resource belongs (See note 1) |
| rel | xs:string | The type (i.e., *rel*ationship type) of resource the link points to (See note 2) |
| name | xs:string | A label that describes or otherwise names the resource |
| uri | xs:string | Location/URI of the resource |
| historyUri | xs:string | Location/URI of a resource that can be used to retrieve previous versions of the template service’s resources |
| renameUri | xs:string | Location/URI of a resource that can be used to change the name of the template service’s document resources |
| uploadUri | xs:string | Location/URI of a generic endpoint that can be used to upload document resources |
| isDeleteable | xs:boolean | Indicates if the resource pointed to by the link can be deleted (See note 3) |

**Notes**

1. If the link is to a service, then the values for *name* and *serviceName* are identical.
2. The value of *rel* indicates the type of the resource*.*
3. Services that contain resources cannot be deleted.

### Parse Error

A *parse error* provides information on a parse error that occurred when running a template against a configuration. Parse Error instances conform to the XML Schema:

<xs:complexType name="ParseErrorType">

<xs:sequence>

<xs:element name="columnNumber" type="xs:int"/>

<xs:element name="lineNumber" type="xs:int"/>

<xs:element name="link" type="LinkType" minOccurs="0"/>

<xs:element name="message" type="xs:string" minOccurs="0"/>

</xs:sequence>  
</xs:complexType>

|  |  |  |
| --- | --- | --- |
| Element | Type | Remarks |
| columnNumber | xs:int | The column number in the file where the error occurred |
| lineNumber | xs:int | The line number in the file where the error occurred |
| link | LinkType | A hypermedia pointer to the resource that triggered the parse error. (See note 1) |
| message | xs:string | A specific message providing detailed information about the parse error that occurred |

**Notes**

1. The *link* points to the file that triggered the parse error. It could be a configuration, a template, or a schema file: the element *name* from the *link* element provides the name of the resource. The filename is provided when *rel* is of type ‘config’ or ‘schema’. When the *rel* value is ‘template’, the *name* value is also ‘template’.

### Rename Result

A *rename result* provides information on the result of a renaming operation. When a resource that is being renamed is associated with other resource(s), then the relevant fields of the other resource(s) are also being renamed. Each one of those associated resources is described using a *renameResource* element. Rename result instances conform to the XML Schema:

<xs:complexType name="RenameResultType">

<xs:sequence>

<xs:element name="newLink" type="LinkType" minOccurs="0"/>

<xs:element name="oldLink" type="LinkType" minOccurs="0"/>

<xs:element name="renamedResource" type="RenameResultType" minOccurs="0" maxOccurs="unbounded"/>

</xs:sequence>  
</xs:complexType>

|  |  |  |
| --- | --- | --- |
| Element | Type | Remarks |
| newLink | LinkType | A hypermedia pointer that provides information on the resource after the renaming operation has been completed. |
| oldLink | LinkType | A hypermedia pointer that provides information on the resource before the renaming operation has been completed. |
| renamedResource | RenameResultType | Provides information on a resource associated with the resource that has been renamed. |

### Resource History

A *resource history* provides the list of the previous versions including details for a resource. Each previous version is represented using a *Historic Link*. The last version of the resource is not listed in the list and instead is described by its relationship type and URI using dedicated fields. *Resource History* instances conform to the XML Schema:

<xs:complexType name="ResourceHistoryType">

<xs:sequence>

<xs:element name="historicLink" type="HistoricLinkType" minOccurs="0" maxOccurs="unbounded"/>

<xs:element name="originalRel" type="xs:string" minOccurs="0"/>

<xs:element name="originalUri" type="xs:string" minOccurs="0"/>

</xs:sequence>  
</xs:complexType>

|  |  |  |
| --- | --- | --- |
| Element | Type | Remarks |
| historicLink | HistoricLinkType | A hypermedia pointer that provides information on the resource historic (See note 1 & 2) |
| originalRel | xs:string | The type (i.e., *rel*ationship type) of resource the link points to (see §3.3.4) |
| originalUri | xs:string | A label that describes or otherwise names the resource |

**Notes**

1. The last version of the resource is not included in the list.
2. Versions of a resource are called *Historics*.

### Resource List

A *resource list* provides a collection of resources represented using *LinkType*. This type is provided to list the resources of a service after a successful *upload* operation (see §4.10). Resource list instances conform to the XML Schema:

<xs:complexType name="ResourceListType">

<xs:sequence>

<xs:element name="link" type="LinkType" minOccurs="0" maxOccurs="unbounded"/>

</xs:sequence>  
</xs:complexType>

|  |  |  |
| --- | --- | --- |
| Element | Type | Remarks |
| link | LinkType | A hypermedia pointer to another resource. |

### Service List

A *service list* provides a list of the services available on the COAT server. Service List instances conform to the XML Schema:

<xs:complexType name="ServiceListType">

<xs:sequence>

<xs:element name="iniSplitter" type="LinkType" minOccurs="0"/>

<xs:element name="link" type="LinkType" minOccurs="0" maxOccurs="unbounded"/>

</xs:sequence>  
</xs:complexType>

|  |  |  |
| --- | --- | --- |
| Element | Type | Remarks |
| iniSplitter | LinkType | A link on a resource that can be used to split INF/INI files to create a new service (See note 1) |
| link | LinkType | A hypermedia pointer to a resource that describes a service that is not an INF/INI splitter |

**Notes**

1. The INF/INI splitter is a service provided by COAT directly. More information can be found in §2.2.2

### Service Resources

A *service resources* provides the list of resources associated with a service, i.e. a detailed description of the service. Service resources instances conform to the XML Schema:

<xs:complexType name="ServiceResourcesType">

<xs:complexContent>  
 <xs:extension base="ResourceListType">

<xs:sequence>

</xs:sequence>  
 </xs:extension>  
 </xs:complexContent>  
</xs:complexType>

**Notes**

1. This type is an extension of the *ResourceList* type described in §3.3.7.

### Template Generation Error

A *template generation error* provides information that describes errors generated when COAT tries to execute an Apache Velocity template. Template generation error instances conform to the XML Schema:

<xs:complexType name="TemplateGenerationErrorType">

<xs:sequence>

<xs:element name="error" type="ParseErrorType" minOccurs="0" maxOccurs="unbounded"/>

<xs:element name="resourceName" type="xs:string" minOccurs="0"/>

<xs:element name="serviceName" type="xs:string" minOccurs="0"/>

</xs:sequence>  
</xs:complexType>

|  |  |  |
| --- | --- | --- |
| Element | Type | Remarks |
| error | ParseErrorType | Detailed information on the error. See §3.3.4 for details. |
| resourceName | xs:string | The name of the resource that made the template generation failed. |
| serviceName | xs:string | The name of the service that was used to generate the template. |

# Resources

## Documentation Conventions

### Nouns and Verbs in URL templates

COAT is following guidelines to aim toward REST compliance. There are however exceptions that have been introduced in order for favor readability of URI.

Some services could have been descripted using a name but instead a language shorthand was used to build the URI template to invoke the services. For example, services that should have been named “renamer” and “uploader” are instead using the shorthand and are called “rename” and “upload”.

### Input Parameters

A detailed description of each service including input parameters is provided in §4.4. The input parameters described for each services belongs to one of the following three categories:

* **Path parameter:** a URL template omitting the protocol and the name of the server used. For example, the <http://coat/version> URI is represented using the shorthand [/version](http://coat/version). Path parameters follow the hierarchical path structure.
* **Query string parameter**: a parameter that allows to provide additional information when using a service. Query string parameters do not follow the hierarchical path structure and are following the structure ***key=value***. Each parameter of the list is separated from the other using the ***&*** symbol and the full list is prefixed using the ***?*** symbol.
* **Header parameter**: a parameter declared in the HTTP header of the request. An example of a header parameter commonly used by COAT is the *Content type* parameter.

## Shared Functionality

### Encoding of message

COAT can preserve the encoding charset used if it specified in the request. This can be done by appending the desired charset to the Content-Type. The payload of the response will then be encoded using the charset specified in the request.

If the payload encoding is not specified, COAT will attempt to determine the charset automatically.

### Output Headers

Output headers are used by COAT to provide information back to the user. A detailed description of each header fields is provided in §3.1.

For each service method described in §4.4, the output headers generated in the response are listed in the **‘Output Parameters’** row.

## Service & Resource Names

Service and resource names must start with a letter, a digit, an underscore, a period, a left parenthesis or a right parenthesis. Each subsequent character must be a letter (“a-z”, “A-Z”), a digit (“0-9”), an underscore (“\_”), a period (“.”), a left parenthesis (“(”), a right parenthesis (“)”), a hyphen (“-“), or a backslash (“\”).

The following are reserved name and neither valid service names nor valid resource names: *config, process, null, schema, test-connection, template, undefined, upload, version.*

## General Operations

### Version

|  |  |
| --- | --- |
| Description | Get the version COAT server |
| URL Template | /version |
| Verb | GET |
| Input Parameters | None |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | version | |
| Output Payload | Version number of COAT as raw text (i.e., not XML) |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

**Usage Notes**

The *version* resource returns the version of COAT according to the following format: *x.x.x.x*

### Test Connection

|  |  |
| --- | --- |
| Description | Test the a client connection to the COAT server |
| URL Template | /test-connection |
| Verb | GET, POST, or DELETE |
| Input Parameters | None |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | test-connection | |
| Output Payload | Version number of COAT as raw text (i.e., not XML) |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

**Usage Notes**

This resource is intended for clients to determine if they are able to connect to COAT using a GET, POST, or DELETE. Depending on the constraints placed up a client (such as a web browser that implements [CORS]), not every HTTP verb leveraged by COAT will be accessible to that client. This resource can also be used for systematic testing.

The *test-connection* resource returns the same information as *version*. However, clients should use *version* for version information, not *test-connection*.

### Service List

|  |  |
| --- | --- |
| Description | Get a list of all the available template services |
| URL Template | / |
| Verb | GET |
| Input Parameters | None |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | serviceList | |
| Output Payload | Service List (§3.3.8) |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

**Usage Notes**

This resource is intended for clients to determine the list of available template services.

## Service Resources

*Service* resources can be queried, created or deleted.

### Get Service Info

|  |  |
| --- | --- |
| Description | Get detailed information about a particular template service |
| URL Template | {serviceName} |
| Verb | GET |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the service to query | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | service | | X-COAT-ResourceName | *{serviceName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | Service Resources (§3.3.9) upon success, *or*  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

### Create Service

|  |  |
| --- | --- |
| Description | Create a template service |
| URL Template | {serviceName} |
| Verb | POST |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the service to create | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | serviceList | | X-COAT-ResourceName | *{serviceName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | None upon success, *or*  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 204 | No Content | Success | | 400 | Bad Request | Error | | 500 | Internal Server Error | Error | |

**Usage Notes**

1. If a service with the same name already exists, then the method will still succeed.
2. If *serviceName* is a reserved name (§4.3), then the response status will be an Internal Server Error (500) and the response body will contain an *IllegalResourceName* exception in the form of an Exception Result (§3.3.1).

### Delete Service

|  |  |
| --- | --- |
| Description | Delete a template service |
| URL Template | {serviceName} |
| Verb | DELETE |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the service to delete | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | serviceList | | X-COAT-ResourceName | *{serviceName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | None upon success, *or*  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 204 | No Content | Success | | 500 | Internal Server Error | Error | |

**Usage Notes**

1. If the service contains document resources, then the response will return an Internal Server Error (500) and the response body will contain a *ResourceNotEmptyException* in the form of an Except Result (§3.3.1).
2. Once a service is deleted, it cannot be retrieved from the COAT web service.
3. This method returns success if the service has already been deleted or otherwise does not exist.
4. This method will return success even given an illegal service name (§4.3).

## Processors

### Process the Default Configuration

|  |  |
| --- | --- |
| Description | Run the template service using the configuration *defaults.xml* |
| URL Template | {serviceName}/process |
| Verb | GET |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the target template service | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | process | | X-COAT-ResourceName | process | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | The results of running the template service using the default configuration |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

**Usage Notes**

As the configuration name is not specified, COAT automatically assumes that the desired configuration name is ‘*default.xml*’.

### Process a Payload

|  |  |
| --- | --- |
| Description | Run the template service using the posted configuration |
| URL Template | {serviceName}/process |
| Verb | POST |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the target template service | | Header | *Content-Type* | Must start with ‘text/xml’ | |
| Input Payload | Configuration that will be used to run the template service |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | process | | X-COAT-ResourceName | process/process | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | The results of running the template service using the posted configuration |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

**Usage Notes**

The payload’s encoding may be specified by appending the desired charset to the Content-Type. For example, consider the HTTP header Content-Type: text/xml; charset=ISO-8859-1. If the payload encoding is not specified, COAT will attempt to determine the charset automatically.

### Process a Named Configuration

|  |  |
| --- | --- |
| Description | Run the template service using the named configuration |
| URL Template | {serviceName}/process/{configName} |
| Verb | GET |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the target template service | | Path | *configName* | Name of the target configuration to use | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | process | | X-COAT-ResourceName | process/*{configName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | The results of running the template service against the named configuration |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

**Usage Notes**

The configuration specified must already exist and be associated with the template service.

## Templates

*Template* resources can be queried, created or deleted. The interface also allows querying for the list of previous versions of a template or for a specific older *template* resource.

### Get Template

|  |  |
| --- | --- |
| Description | Get the template associated with the template service resource |
| URL Template | {serviceName}/template |
| Verb | GET |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the target template service | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | template | | X-COAT-ResourceName | template | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | The template associated with the service upon success,  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 404 | Not Found | Error | | 500 | Internal Server Error | Error | |

**Usage Notes**

Each template service has at most one template.

### Create Template

|  |  |
| --- | --- |
| Description | Create a template resource and associate it with a template service resource |
| URL Template | {serviceName}/template |
| Verb | POST |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the target template service | | Header | *Content-Type* | Must start with ‘text/plain’ | |
| Input Payload | The template to associate with the template service resource represented as plain text |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | template | | X-COAT-ResourceName | template | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | None upon success, *or* Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 400 | Bad Request | Error | | 500 | Internal Server Error | Error | |

**Usage Notes**

1. Each template service has at most one template.
2. If the template service has already a template associated, the existing template is overwritten
3. The payload’s encoding may be specified by appending the desired charset to the Content-Type. For example, consider the HTTP header Content-Type: text/plain; charset=ISO-8859-1. If the payload encoding is not specified, COAT will attempt to determine the charset automatically.

### Delete Template

|  |  |
| --- | --- |
| Description | Delete a template resource |
| URL Template | {serviceName} |
| Verb | DELETE |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the service containing template to delete | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | template | | X-COAT-ResourceName | template | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | None |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 204 | No Content | Success | | 500 | Internal Server Error | Error | |

### Template History

|  |  |
| --- | --- |
| Description | Return a list of historic templates associated with the template service resource |
| URL Template | history/{serviceName}/template |
| Verb | GET |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the service resource | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | history/template | | X-COAT-ResourceName | template | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | Resource History (§3.3.6) upon success, *or*  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 400 | Bad Request | Error | | 500 | Internal Server Error | Error | |

### Historic Template

|  |  |
| --- | --- |
| Description | Get a historic template resource, i.e. a specific older version of the template resource |
| URL Template | historic/{template}/{serviceName}/template |
| Verb | GET |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the service resource | | Path | *template* | A timestamp expressed in millisecond identifying the historic instance | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Historic-Rel-Of | template | | X-COAT-Historic-Version-Of | http://*serverName*/*{serviceName}*/template | | X-COAT-Rel | historic/template | | X-COAT-ResourceName | template | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | The template associated with the service upon success,  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

## Schemas

*Schema* resources can be queried, created or deleted. The interface also allows querying for the list of previous versions of a schema or for a specific older *schema* resource.

### Get Schema

|  |  |
| --- | --- |
| Description | Get a schema resource (XSD) associated with the template service resource |
| URL Template | {serviceName}/schema/{schemaName} |
| Verb | GET |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the target template service | | Path | *schemaName* | Name of the target schema | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | schema | | X-COAT-ResourceName | *{schemaName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | The named schema associated with the service upon success,  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 204 | No Content | Success | | 404 | Not Found | Error | | 500 | Internal Server Error | Error | |

**Usage Notes**

1. The schema returned is an XML schema (XSD).
2. If the schema designed by *schemaName* doesn’t exist, then the response status will be a Not Found (404) and the response body will contain an *ResourceNotFound* exception in the form of an Exception Result (§3.3.1)

### Create Schema

|  |  |
| --- | --- |
| Description | Create a schema resource and associate it with a template service resource |
| URL Template | {serviceName}/schema/{schemaName} |
| Verb | POST |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *schemaName* | Name of the schema to create | | Path | *serviceName* | Name of the target template service | | Header | *Content-Type* | Must start with ‘text/xml’ | |
| Input Payload | A schema to associate with the template service resource |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | schema | | X-COAT-ResourceName | *{schemaName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | None upon success, *or* Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 204 | No Content | Success | | 400 | Bad Request | Error | | 500 | Internal Server Error | Error | |

**Usage Notes**

1. The schema is created, or overwritten if a schema with the same name already exists.
2. If *schemaName* is a reserved name or contains illegal characters (§4.3), then the response status will be a Bad Request (400) and the response body will contain an *IllegalResourceName* exception in the form of an Exception Result (§3.3.1)
3. The payload’s encoding may be specified by appending the desired charset to the Content-Type. For example, consider the HTTP header Content-Type: text/xml; charset=ISO-8859-1. If the payload encoding is not specified, COAT will attempt to determine the charset automatically.

### Delete Schema

|  |  |
| --- | --- |
| Description | Delete a schema resource |
| URL Template | {serviceName}/schema/{schemaName} |
| Verb | DELETE |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the target template service | | Path | *schemaName* | Name of the target schema | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | schema | | X-COAT-ResourceName | *{schemaName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | None |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 204 | No Content | Success | | 500 | Internal Server Error | Error | |

### Schema History

|  |  |
| --- | --- |
| Description | Return a list of historic schemas associated with the template service resource |
| URL Template | history/{serviceName}/schema/{schemaName} |
| Verb | GET |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the service resource | | Path | *schemaName* | Name of the target schema | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | history/schema | | X-COAT-ResourceName | *{schemaName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | Resource History (§3.3.6) upon success, *or*  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

### Historic Schema

|  |  |
| --- | --- |
| Description | Get a historic schema resource, i.e. a specific older version of the schema resource |
| URL Template | historic/{template}/{serviceName}/schema/{schemaName} |
| Verb | GET |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *schemaName* | Name of the target schema | | Path | *serviceName* | Name of the service resource | | Path | *template* | A timestamp expressed in millisecond identifying the historic instance | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Historic-Rel-Of | schema | | X-COAT-Historic-Version-Of | http://*serverName*/*{serviceName}*/schema/*{schemaName}* | | X-COAT-Rel | historic/schema | | X-COAT-ResourceName | *{schemaName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | The specific version of the schema associated with the service upon success,  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 400 | Bad Request | Error | | 500 | Internal Server Error | Error | |

## Configurations

*Configuration* resources can be queried, created or deleted. The interface also allows querying for the list of previous versions of a configuration or for a specific older *configuration* resource.

### Get Configuration

|  |  |
| --- | --- |
| Description | Get a configuration resource (XML) associated with the template service resource |
| URL Template | {serviceName}/config/{configName} |
| Verb | GET |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the target template service | | Path | *configName* | Name of the target configuration | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | config | | X-COAT-ResourceName | *{configName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | The named configuration associated with the service upon success,  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 404 | Not Found | Error | | 500 | Internal Server Error | Error | |

**Usage Notes**

1. The configuration returned is an XML document.
2. If the configuration designed by *configName* doesn’t exist, then the response status will be a Not Found (404) and the response body will contain an *ResourceNotFound* exception in the form of an Exception Result (§3.3.1)

### Create Configuration

|  |  |
| --- | --- |
| Description | Create a configuration resource and associate it with a template service resource |
| URL Template | {serviceName}/config/{configName} |
| Verb | POST |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the target template service | | Path | *configName* | Name of the configuration to create | | Header | *Content-Type* | Must start with ‘text/xml’ | |
| Input Payload | A configuration to associate with the template service resource |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | config | | X-COAT-ResourceName | *{configName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | None upon success, *or* Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 204 | No Content | Success | | 400 | Bad Request | Error | | 500 | Internal Server Error | Error | |

**Usage Notes**

1. The configuration is created, or overwritten if a configuration with the same name already exists.
2. If *configName* is a reserved name or contains illegal characters (§4.3), then the response status will be a Bad Request (400) and the response body will contain an *IllegalResourceName* exception in the form of an Exception Result (§3.3.1)
3. The payload’s encoding may be specified by appending the desired charset to the Content-Type. For example, consider the HTTP header Content-Type: text/xml; charset=ISO-8859-1. If the payload encoding is not specified, COAT will attempt to determine the charset automatically.

### Delete Configuration

|  |  |
| --- | --- |
| Description | Delete a configuration resource |
| URL Template | {serviceName}/config/{configName} |
| Verb | DELETE |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the target template service | | Path | *configName* | Name of the configuration to delete | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | config | | X-COAT-ResourceName | *{configName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | None |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 204 | No Content | Success | | 500 | Internal Server Error | Error | |

### Configuration History

|  |  |
| --- | --- |
| Description | Return a list of historic configurations associated with the template service resource |
| URL Template | history/{serviceName}/config/{configName} |
| Verb | GET |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the service resource | | Path | *configName* | Name of the target configuration | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | history/config | | X-COAT-ResourceName | *{configName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | Resource History (§3.3.6) upon success, *or*  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

### Historic Configuration

|  |  |
| --- | --- |
| Description | Get a historic configuration resource, i.e. a specific older version of the named configuration resource |
| URL Template | historic/{template}/{serviceName}/config/{configName} |
| Verb | GET |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *template* | A timestamp expressed in millisecond identifying the historic instance | | Path | *serviceName* | Name of the service resource | | Path | *configName* | Name of the target configuration | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Historic-Rel-Of | config | | X-COAT-Historic-Version-Of | http://*serverName*/*{serviceName}*/config/*{configName}* | | X-COAT-Rel | historic/config | | X-COAT-ResourceName | *{configName}* | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | The specific version of the named configuration associated with the service upon success,  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

## Renamers

*Rename* resources allow document resources including services, schemas and configuration, to be renamed.

### Rename Service

|  |  |
| --- | --- |
| Description | Rename a service to the name specified using the query parameter |
| URL Template | rename/service/{serviceName}?newName={newName} |
| Verb | POST |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the service resource | | Query | *newName* | New name for the service | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-New-Name | *{newName}* | | X-COAT-Old-Name | *{serviceName}* | | X-COAT-Rel | renamer/service | | X-COAT-ResourceName | *{newName}* | | X-COAT-ServiceName | *{newName}* | |
| Output Payload | Rename Result (§3.3.5) upon success, *or*  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

**Usage Notes**

1. If the service is successfully renamed, then the location of the renamed service is returned as a *Rename Result*

### Rename Schema

|  |  |
| --- | --- |
| Description | Rename a schema to the name specified using a query parameter |
| URL Template | rename/{serviceName}/schema/{schemaName}?newName={newName} |
| Verb | POST |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the target service | | Path | *schemaName* | Current schema name | | Query | *newName* | New name for the schema | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-New-Name | *{newName}* | | X-COAT-Old-Name | *{schemaName}* | | X-COAT-Rel | renamer/schema | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | Rename Result (§3.3.5) upon success, *or*  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

**Usage Notes**

1. If the schema is successfully renamed, then the location of the renamed schema is returned as a *Rename Result*

### Rename Configuration

|  |  |
| --- | --- |
| Description | Rename a configuration to the name specified using a query parameter |
| URL Template | rename/{serviceName}/config/{configName}?newName={newName} |
| Verb | POST |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the target service | | Path | *configName* | Current configuration name | | Query | *newName* | New name for the configuration | |
| Input Payload | None |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-New-Name | *{newName}* | | X-COAT-Old-Name | *{configName}* | | X-COAT-Rel | renamer/config | | X-COAT-ServiceName | *{serviceName}* | |
| Output Payload | Rename Result (§3.3.5) upon success, *or*  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

**Usage Notes**

1. If the configuration is successfully renamed, then the location of the renamed configuration is returned as a *Rename Result*

## Upload

The *upload* resource can be used create a template, configuration, or schema resource from posted content. This convenience wrapper provides a uniform endpoint and assigns posted content according to the designated filename.

|  |  |
| --- | --- |
| Description | Create a template, configuration, or schema resource from posted content |
| URL Template | {serviceName}/upload?rel={rel}&name={name} |
| Verb | POST |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the service to upload contents into | | Query | *rel* | Type of resource being uploaded | | Query | *name* | Name of the resource | | Header | *Content-Type* | Must start with ‘text/plain’ or ‘text/xml’ | |
| Input Payload | A template, a configuration, or a schema resource to associate with the service resource |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-Rel | upload | | X-COAT-ResourceName | *{name}* | | X-COAT-ServiceName | *{serviceName}* | | X-COAT-Uploaded-Rel | *{rel}* | |
| Output Payload | Resource List (§3.3.7) upon success, *or*  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

**Usage Notes**

1. The *rel* parameter can take the following values: ‘config’, ‘template’ or ‘schema’
2. If the *rel* parameter is 'template' then the *name* parameter is ignored
3. If the resource is successfully uploaded, then the location of the renamed configuration is returned as a *Rename Result*

## Splitters

*A splitter* resource creates a template service, a configuration and a schema from an existing configuration file.

### INF Splitter

|  |  |
| --- | --- |
| Description | Create a template service, a configuration and a schema by “splitting” an INF/INI file |
| URL Template | splitter/ini?serviceName={serviceName} |
| Verb | POST |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Query | *serviceName* | Name of the service to create | | Header | *Content-Type* | Must start with ‘text/plain’ | |
| Input Payload | The file that will be “split” to create the template service |
| Output Parameters | None |
| Output Payload | Service List (§3.3.8) upon success, *or*  Exception Result (§3.3.1) upon error |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Status Text | Condition | | 200 | OK | Success | | 500 | Internal Server Error | Error | |

1. Currently, COAT only supports the “splitting” of INF/INI files. [↑](#footnote-ref-1)
2. We expect that the COAT implementation will be updated more frequently than this documentation. Therefore, we encourage readers to use this section simply as a way to get started quickly with COAT. The walkthrough was generated from real data (and checked for accuracy), but it is not a verbatim transcript. In other words, we recommend reading the walkthrough to get acquainted with the concepts of COAT; not using it as an interactive tutorial against which you can check your own installation. [↑](#footnote-ref-2)
3. Highlighted and indented for clarity. Leaving these indentations in the actual Apache Velocity template will generate spaces in the output. [↑](#footnote-ref-3)