**COAT**

**Cloud Orchestration Automation Templating**

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

**User Manual**

Ross J. Micheals

Kevin Mangold

# 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 allow templates and configurations to more easily be shared and managed.

## 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

### Machine-Readable Code

### Variables

Portions of this document use the

## 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 clients 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 to finish this

Contrast this with our previous template. Here, we’ve used ${ROOT.lines} instead of ${lines}. This is because the COAT XML processing layer will convert the XML configuration into a (Java) *object*, where each XML element becomes a property (i.e., a pair of get() and set() method

### 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 that resources are 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).
* 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.
* Through a structured URL, we specify a name for the configuration (default.xml) and schema (main.xsd). (For convenience, we use the same names as we did for the files). Because there can only be one template for a template service, we do not need to specify a template name.

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.

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.xml’

> 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 at “execution” time as the HTTP payload. 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**
2. **Get a Previous Version of a Resource.**
3. **Rename a Service or Resource**.
4. **Split an INF/INI File**.
5. **Use the “Upload” Resource.** Resources don’t need to be created with the structured URL. A single URL

## Expected Usage

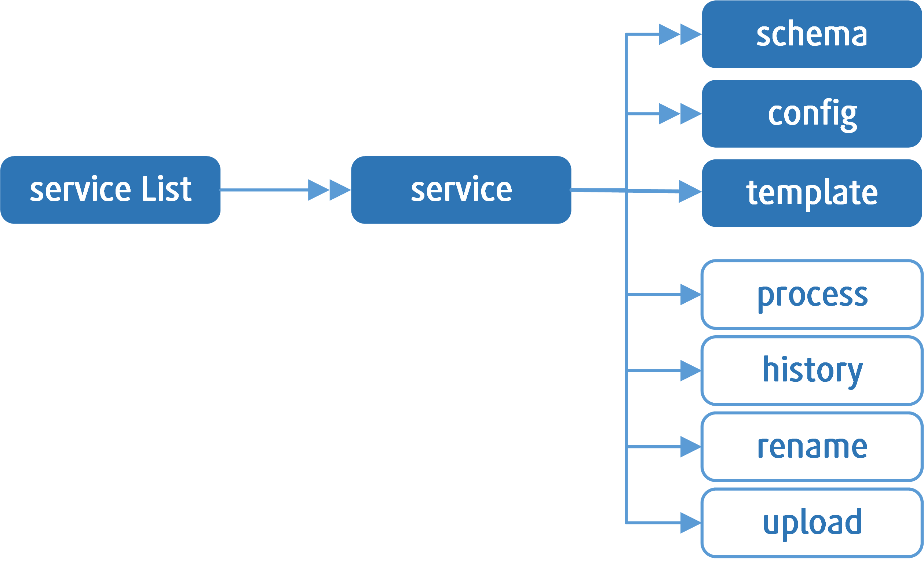
This section more formally describes the expected usage of COAT template services.

### Consumption Style

COAT services are accessible by both structured URLs as well has through the more RESTful *hypertext as the engine of application state* (HATEOS) mechanism. Consumers of COAT services are encouraged to rely primarily on the HATEOS, 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 with which clients will retrieve or activate the various resources. A double arrow represents a *one-to-many* relationship



Describe the expected hateos pattern (service list to service to resources)

Describe that the service will only activate certain links once basic constraints are met; for example no default ‘process’ link without a schema, default config and template;

Described that reousrces (shema/config/template) are scoped to a service; they are not shared between different services. That is config for one service is independent of their other; a configuration does not

## 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 HATEOS) 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 1. 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 result of processing a template against the provided configuration, the service output, 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 will can give deeper insight into which stages are failing. For example, if XML validation is fails, this implies that code generation and compilation are not. Likewise, errors in template processing are not directly attributable to XML validation. (Naturally, this is not a guarantee that previous stages are defect free, 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

Enumerate and describe the X-COAT-\* headers

Describe the CORS headers built-in to the service

## 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 returned by COAT as an XML payload.

### Exception Result

### 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 §3.3.3) |
| 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 document 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. |

**Notes**

1. If the link is to a service, then the values for ‘name’ and ‘serviceName’ are identical.
2. Services that contain resources cannot be deleted

### Parse Error

### Rename Result

### Relationship (Rel)

### Resource History

### Resource List

The relationship, or *rel* describes the type of resources.

### Service List

### Service Resources

### Template Generation Error

A *template generation error* provides information that describes errors generated during the processing pipline when COAT tries to execute an Apache Velocity template (§).

# Resources

## Documentation Conventions

Language shorthand (“rename” vs “renamer”)

Describe input parameters

Path (url template)

Query (?rel=foo;)

Header (inside the HTTP header)

## Shared Functionality

* Describe that encoding is preserved
* Describe common Output headers

## 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 neither valid service names nor valid resource names: *config, process, null, schema, test-connection, template, undefined, upload, version.*

## Services

### General Operations

#### Version

|  |  |
| --- | --- |
| Description | Get the version COAT server |
| URL Template | /version |
| Verb | GET |
| Input Parameters | None |
| Input Payload | None |
| Output Parameters | None |
| 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**

#### 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 | None |
| 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 be used to by a client 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 | None |
| Output Payload | Service List (3.3.8) |
| Output Status | |  |  |  | | --- | --- | --- | | Code | Name | Condition | | 200 | OK | Success | |

### Service

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

#### Get Template Service Info

|  |  |
| --- | --- |
| Description | Get detailed information about a particular service |
| URL Template | {serviceName} |
| Verb | GET |
| Input Parameters | None |
| Input Payload | None |
| Output Parameters | None |
| Output Payload | Service Resources (§3.3.9) 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 | |

#### Create Template 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 | None |
| 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**

1. If the 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 Template Service

|  |  |
| --- | --- |
| Description | Delete a service |
| URL Template | {serviceName} |
| Verb | DELETE |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the service to create | |
| Input Payload | None |
| Output Parameters | None |
| 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

### Default Processor

|  |  |
| --- | --- |
| Description | Run the service’s template against the default 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 against which to run the template |
| Output Parameters | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-ServiceName | *{serviceName}* | | X-COAT-ResourceName | process/process | |
| Output Payload |  |

**Usage Notes**

### Payload Processor

|  |  |
| --- | --- |
| Description | Run the service’s template against the posted configuration |
| URL Template | {serviceName}/process |
| Verb | POST |
| Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | serviceName | Name of the target template service | | Header | Content-Type | Must start with ‘text/xml’ | |
| Input Payload | Configuration against which to run the template |
| Output | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Header | X-COAT-ServiceName | *{serviceName}* | | Header | X-COAT-ResourceName | process/process | | Header | Content-Type | Starts with ‘text/plain’ | | Payload | n/a | Output of the processing step | |

**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/plain; charset=ISO-8859-1. If the payload encoding is not specified, COAT will attempt to determine the charset automatically.

### Named Processor

|  |  |
| --- | --- |
| Description | Run the service’s template against the named configuration |
| URL Template | {serviceName}/process/{configName} |
| Verb | GET |
| Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | serviceName | Name of the target template service | | Path | configName | Name of the configuration to process | | Header | Content-Type | Must start with ‘text/xml’ | |
| Input Payload | None |
| Output Payload | |  |  | | --- | --- | | HTTP Header | Value | | X-COAT-ServiceName | *{serviceName}* | | X-COAT-ResourceName | process/*{configName}* | |

## Templates

### Template

#### Create Template

**Usage Notes**

1. Each template service has at most one template

#### Delete Template

|  |  |
| --- | --- |
| Description | Delete a template |
| URL Template | {serviceName}/template |
| Verb | DELETE |
| Input Parameters | |  |  |  | | --- | --- | --- | | Style | Name | Description | | Path | *serviceName* | Name of the service containing template to delete | |
| Input Payload | None |
| Output Parameters | None |
| 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**

### Template History

### Historic Template

## Schemas

### Schema

#### Create Schema

#### Delete Schema

### Schema History

### Historic Schema

## Configurations

### Configuration

#### Create Configuration

#### Delete Configuration

### Configuration History

### Historic Configuration

## Renamers

### Schema Renamer

### Config Renamer

## Upload

|  |  |
| --- | --- |
| Description |  |
| URL Template | {serviceName}/upload?rel={rel}&name={name} |
| Verb | GET |
| 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 | contentType |  | |
| Input Payload | Contents of resource to be created |
| Hypermedia Referer | Get Service Info |
| Returns |  |

## Splitters

### INF Splitter

1. Currently, COAT only supports the “splitting” of INI/INF 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)