

REST Developer's Guide

Version 9.7

October 2014

This document applies to webMethods Integration Server Version 9.7 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 2007-2014 Software AG, Darmstadt, Germany and/or Software AG USA Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://documentation.softwareag.com/legal/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". This document is part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

Table of Contents

About this Guide.....	5
Document Conventions.....	5
Documentation Installation.....	6
Online Information.....	6
About Integration Server REST Processing.....	7
Overview.....	8
How REST Processing Works.....	11
About REST Request Messages.....	12
How webMethods Integration Server Processes REST Requests.....	12
Processing Requests Using Partial Matching of URL Aliases.....	14
Sending Responses to the REST Client.....	15
Status Line.....	16
Header Fields.....	16
Message Body.....	16
Setting Up Your REST Application.....	17
Setting Up a REST Application on Integration Server.....	18
Services.....	18
Configuration.....	19
Converting an Existing Application.....	20
Documenting Your Rest Application.....	21
Providing Information About Your Application.....	22
General Information.....	22
Information About Each Request.....	22
Information About Responses.....	24
Index.....	25

About this Guide

This guide is for developers using webMethods Integration Server to create REST applications. This guide assumes basic knowledge of REST concepts and HTTP request processing and familiarity with Software AG Designer and webMethods Integration Server.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies storage locations for services on webMethods Integration Server, using the convention <i>folder.subfolder:service</i> .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
<i>Italic</i>	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
Monospace font	Identifies text you must type or messages displayed by the system.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the symbol.
[]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Documentation Installation

You can download the product documentation using the Software AG Installer. The documentation is downloaded to a central directory named `_documentation` in the main installation directory (SoftwareAG by default).

Online Information

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at <http://documentation.softwareag.com>. The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at <https://empower.softwareag.com>.

To submit feature/enhancement requests, get information about product availability, and download products and certified samples, go to [Products](#).

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the [Knowledge Center](#)

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at <http://techcommunity.softwareag.com>. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

1 About Integration Server REST Processing

■ Overview	8
------------------	---

Overview

Representational State Transfer (REST) is an architectural style used to build distributed hypermedia systems. The World Wide Web is the best known example of such a system.

The focus of REST is on resources rather than services. A resource is a representation of an object or information. A resource can represent:

- A single entity, like a coffee pot you want to purchase from an online shopping site.
- A collection of entities, like records from a database.
- Dynamic information, like real-time status updates from a monitoring site.

That is, resources are the entities or collections of entities in a distributed system that you want to post or retrieve or take action on. In a REST style system, each resource is identified by a universal resource identifier (URI).

Development of REST systems is defined by a series of constraints:

- Clients and servers are separate.
- Communication between clients and servers is stateless.
- Clients can cache responses returned from servers.
- There may be intermediate layers between the client and server.
- Servers can supply code for the clients to execute.
- Clients and servers remain loosely coupled by communicating through a uniform interface.

The uniform interface is the key constraint that differentiates REST from other architectural approaches. The characteristics of this interface are:

- Requests identify resources.
- Responses contain representations of those resources.
- Clients manipulate resources through their representations.
- Messages are self-descriptive.
- The interface employs Hypermedia as the engine of application state (HATEOAS), which enables the client to find other resources referenced in the response.

One strength of REST is that it leverages the well understood methods supported by HTTP to describe what actions should be taken on a resource. To be REST-compliant, an application must support the HTTP GET, POST, PUT, and DELETE methods. Many applications use web browsers to interact with resources on the Internet. web browsers, however, typically support only the HTTP GET and HTTP POST methods. To get around this restriction, you can use Integration Server to build REST-compliant applications that support all four methods.

Integration Server can be a REST server or a REST client. When Integration Server acts as a REST server, it hosts an application that you write. The application includes services that you write that instruct Integration Server to process some or all of the HTTP GET, POST, PUT, and DELETE methods in request messages against resources. When Integration Server acts as a REST client, it sends specially formatted requests to the REST server.

2 How REST Processing Works

- About REST Request Messages 12
- How webMethods Integration Server Processes REST Requests 12
- Processing Requests Using Partial Matching of URL Aliases 14
- Sending Responses to the REST Client 15

About REST Request Messages

REST clients send specially formatted requests to your REST application. The format of REST requests is determined by the webMethods Integration Server REST implementation and your specific application, but essentially it conveys the following information, or tokens, to the REST server:

- The HTTP method to execute
- The directive
- The name of the resource

A simple REST request looks like this:

```
METHOD /directive/resource_type/resource_id HTTP/1.1
```

Where...	Is the...
<i>METHOD</i>	HTTP request method.
<i>directive</i>	The type of processing to perform.
<i>resource_type/ resource_id</i>	Resource to act upon.

More complex request messages can contain more explicit information about the resource.

How webMethods Integration Server Processes REST Requests

When Integration Server processes a REST request, it parses the tokens and identifies the HTTP method to execute, locates the resource to act upon, and passes additional information as input parameters to the services you wrote for your application. On Integration Server the resources of your application are represented as folders within a package. For each resource, you will write individual services for the HTTP methods that you want Integration Server to execute against the resource. Those services must be named `_get`, `_post`, `_put`, and `_delete`, and they are stored in the folder for the resource. For more information, see ["Services" on page 18](#).

Consider a Discussion application that maintains a database of discussions about different topics. The following examples show how Integration Server would parse these REST requests.

Example 1

Here is a request to obtain a list of all topics contained in the database, and how Integration Server would parse the request:

```
GET /rest/discussion/topic HTTP/1.1
```

Where...	Is the...
GET	Type of HTTP method to perform. Integration Server maps this value to the corresponding service on Integration Server, in this case, the <code>_get</code> service.
rest	Type of processing to perform, in this case, Integration Server REST processing. Note: For more information about directives, see <i>webMethods Integration Server Administrator's Guide</i> .
discussion/topic	Location of the <code>_get</code> service for this resource on Integration Server. In this example, the <code>_get</code> service resides in the topic folder in the discussion folder (<code>discussion.topic</code>).

Example 2

Here is a request to display information about topic number 3419, and how Integration Server would parse the request:

```
GET /rest/discussion/topic/3419 HTTP/1.1
```

Where...	Is...
3419	An instance of a resource passed into a service as the <code>\$resourceID</code> variable. In the example, the <code>\$resourceID</code> variable narrows the focus of the GET request to topic 3419. Note: Integration Server assigns the first token after the folder(s) to the <code>\$resourceID</code> parameter. To determine whether a token represents a folder or the <code>\$resourceID</code> , Integration Server looks in the current namespace for a folder that has the same name as the token. If it does not find a folder with this name, Integration Server assigns the token to the <code>\$resourceID</code> variable. In other words, the first token (after the directive) that does not correspond to a folder becomes the <code>\$resourceID</code> .

Example 3

Here is a request to display information about a particular comment, 17 for example, and how Integration Server would parse the request:

```
GET /rest/discussion/topic/3419/comment/17 HTTP/1.1
```

Where...	Is...
comment/17	Additional information that further narrows the information about the resource. This information is passed into a service as the <i>\$path</i> variable. In the example, <code>comment/17</code> further narrows the focus of the GET request to comment 17.

Example 4

Here is a request to display information contributed by participant Robertson in 2009 about topic 17, and how Integration Server would parse the request:

```
GET /rest/discussion/topic/3419/comment/17?year=2009&name=Robertson
HTTP/1.1
```

Where...	Are...
year and name	Input variables that are specific to your application. Tokens specified after the <code>?</code> must be entered as name/value pairs. In this example, <code>year=2009</code> and <code>name=Robertson</code> narrow the focus of the GET request to entries that participant Robertson added to comment 17 in 2009.

Processing Requests Using Partial Matching of URL Aliases

REST URL requests usually include the identifier for a particular resource. However, because the identifier varies for each instance of a resource, REST requests often do not exactly match any of the defined URL aliases for a particular resource. To enable you to define URL aliases for REST resources, Integration Server can use partial matching to process REST requests. A *partial match* occurs when a REST request includes only part of a URL alias. For more information about URL aliases, see *webMethods Integration Server Administrator's Guide*.

When partial matching is enabled and Integration Server receives a REST request URL, an alias is considered a match if the entire alias matches all or part of the request URL, starting with the first character of the request URL's path.

For example, assume the following URL aliases are defined:

URL Alias	URL Path
a1	rest/purchasing/order

URL Alias	URL Path
a2	rest/purchasing/invoice
a22	rest/purchasing/admin
a3	invoke/pub.flow/debugLog

When partial matching is enabled, the following request URLs would get different results:

- A request URL of `http://MyHost:5555/a1` matches URL alias a1 exactly. The resulting URL is `http://MyHost:5555/rest/purchasing/order`.
- A request URL of `http://MyHost:5555/a2/75909` matches alias a2 because the request URL's path begins with "a2". The trailing characters of the request URL are retained and the resulting URL is `http://MyHost:5555/rest/purchasing/invoice/75909`.
- A request URL of `http://MyHost:5555/a1/75909/customer/0122?terms=net7` matches alias a1 because the request URL's path begins with "a1". The trailing characters of the request URL are retained and the resulting URL is `http://MyHost:5555/rest/purchasing/order/75909/customer/0122?terms=net7`.

Note: When a URL alias starts with the same characters as another alias, Integration Server will not match the longer alias while configured to perform partial matching. This is because Integration Server evaluates the request URL in ascending alphabetical order. For this reason, you should never define an alias that begins with another alias (example, creating aliases named a2 and a22). For example, a request URL of `http://MyHost:5555/a22` matches alias a2 because the request URL's path begins with "a2". The resulting URL is `http://MyHost:5555/rest/purchasing/invoice`. In this example, alias a22 is not matched to the correct URL alias because Integration Server evaluates the request URL in ascending alphabetical order and makes a partial match to URL alias a2.

Note: In some cases, a partial match can result in an invalid request. For example, a request URL of `http://host:5555/a3456` matches alias a3 because the request URL's path begins with "a3". The trailing characters of the request URL are retained and the resulting URL is `http://host:5555/invoke/pub.flow/debugLog456`. Since there is no `pub.flow:debugLog456` service, this would be an invalid request.

For instructions on enabling partial matching, see *webMethods Integration Server Administrator's Guide*.

Sending Responses to the REST Client

When Integration Server responds to an HTTP request, the response contains a status line, header fields, and a message body.

Status Line

The status line consists of the HTTP version followed by a numeric status code and a reason phrase. The reason phrase is a brief textual description of the status code. Integration Server will always set the HTTP version to match the version of the client that issued the request. You cannot change the HTTP version.

You can, however, use the `pub.flow:setResponseCode` service to set the status code and reason phrase. For more information about this service, refer to the *webMethods Integration Server Built-In Services Reference*. If you do not explicitly set the status code, Integration Server will set it to 200 for successfully completed requests and an appropriate error code for unsuccessful requests.

HTTP/1.1 defines all the legal status codes in Section <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10>. Examine these codes to determine which are appropriate for your application.

Header Fields

You can communicate information about the request and the response through header fields in the HTTP response. Integration Server will generate some header fields, such as Set-Cookie, WWW-Authenticate, Content-Type, Content-Length, and Connection. You can use the `pub.flow:setResponseHeader` to set Content-Type and other header fields. For more information about this service, refer to the *webMethods Integration Server Built-In Services Reference*.

HTTP/1.1 defines the header fields that can appear in a response in three sections of RFC 2616: [4.5](#), [6.2](#), and [7.1](#). Examine these codes to determine which are appropriate for your application.

Message Body

The message body usually contains a representation of the requested resource, one or more URLs that satisfy the request, or both. In some cases, the message body should be empty, as specified in [RFC 2616, Section 4.3](#)

You can use the `pub.flow:setResponse` service to explicitly set the message body. See the *webMethods Integration Server Built-In Services Reference* for details. If you do not explicitly set the message body, the output pipeline of the top-level service will be returned to the client in the message body.

For more information about how Integration Server builds HTTP responses, refer to the section that describes how Integration Server chooses content handlers in *webMethods Integration Server Administrator's Guide*.

3 Setting Up Your REST Application

- Setting Up a REST Application on Integration Server 18
- Converting an Existing Application 20

Setting Up a REST Application on Integration Server

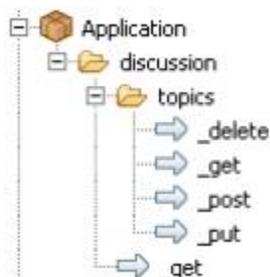
Integration Server can act as a REST server or REST client. For Integration Server to act as a REST server, it must host services that perform the GET, PUT, POST, and DELETE methods. These services, which you provide, perform processing that is specific to your application. For example, the service you write to perform the GET method might access your application database, perform a query, and return a result.

Services

When you build a REST application on your Integration Server, you must include services that correspond to the HTTP methods you want to provide for each resource. These services must be named as follows:

Service	Description
<code>_get</code>	Performs the GET method.
<code>_put</code>	Performs the PUT method.
<code>_post</code>	Performs the POST method.
<code>_delete</code>	Performs the DELETE method.

These services reside in folders on your Integration Server in a directory structure that is specific to your application. For example, the Discussion application described above might have the following structure as viewed from Software AG Designer:



In addition to the `_get`, `_put`, `_post`, and `_delete` services, you can also place a special service named `_default` in one or more of the application folders. Integration Server executes this service if a REST request specifies an HTTP method that is not represented by a service in the folder. For example, suppose the folder contains the `_get`, `_put`, and `_post` services, but no `_delete` service. If the client issues a DELETE request, Integration Server will execute the `_default` service, and pass "DELETE" to it in the `$httpMethod` variable.

If a request specifies an HTTP request method that is not represented by a service in the folder and there is no `_defaultservice` in the folder, the request fails with error “404 Not Found” or “405 Method Not Allowed.” Integration Server issues 404 if the first token in the URI does not exist in the namespace, or 405 if one or more tokens in the URI identify elements in the namespace but the URI does not correctly identify a folder and a service to execute.

Example 1

A resource’s folder contains the `_get`, `_post`, and `_default` services:

<u>If the client sends a...</u>	<u>Integration Server responds by...</u>
GET request	Executing the <code>_get</code> service
POST request	Executing the <code>_post</code> service
DELETE request	Executing the <code>_default</code> service

Example 2

A resource’s folder contains the `_get`, `_put`, and `_delete` services:

<u>If the client sends a...</u>	<u>Integration Server responds by...</u>
GET request	Executing the <code>_get</code> service
PUT request	Executing the <code>_put</code> service
POST request	Issuing error “405 Method Not Allowed”

Additional possible uses for the `_default` service are:

- Direct all REST requests through common code before branching off to individual GET, PUT, POST, or DELETE methods.
- Make PUT and POST processing the same by directing PUT and POST requests to the same code.

Configuration

There are a few things you can configure with respect to REST processing:

- Name of the REST directive

If you want to allow clients to specify a name other than “rest” for the REST directive, you can do so with the `watt.server.RESTDirective` configuration parameter.

For example, to allow clients to specify “process” for the REST directive, you would change the property to the following:

```
watt.server.RESTDirective=process
```

With this setting, clients can specify “rest” or “process” for the REST directive. In the following example, the two requests are equivalent:

```
METHOD /process/discussion/topic/9876 HTTP/1.1
```

```
METHOD /rest/discussion/topic/9876 HTTP/1.1
```

For more information about the `watt.server.RESTDirective` property, refer to *webMethods Integration Server Administrator’s Guide*.

- Which ports will accept the rest directive

By default, all Integration Server ports except the proxy port allow use of the rest directive. You can limit which ports will allow this directive by specifying them on the `watt.server.allowDirective` configuration parameter. For more information about this property, refer to the *webMethods Integration Server Administrator’s Guide*.

Converting an Existing Application

If you have an existing application that you want to transform into a REST application, one approach is to refactor your existing services into `_get`, `_put`, `_post` and `_delete` services. There is, however, another approach that might be faster and easier.

The `invoke` directive, like the `rest` directive, supports the GET, PUT, POST, and DELETE methods. For existing applications that use the `invoke` directive, you can update a service to call the `pub.flow:getTransportInfo` service and then perform a branch on `/transport/http/method` to execute the appropriate portions of your existing code, as in the following example:

```
➔ pub.flow:getTransportInfo
🏠 BRANCH (on '/transport/http/method')
  ⚡ GET: SEQUENCE
  ⚡ PUT: SEQUENCE
  ⚡ POST: SEQUENCE
  ⚡ DELETE: SEQUENCE
```

Note, however, that if you use the `invoke` directive instead of the `rest` directive, you cannot take advantage of these REST-specific features:

- `$resourceID` pipeline variable
- `$path` pipeline variable
- `_default` service

4 Documenting Your Rest Application

- Providing Information About Your Application 22

Providing Information About Your Application

It is important to document your REST application so that your customers and partners will be able to build clients that interact with it correctly. Your documentation should cover how to:

- Send requests to your application
- Handle responses from your application

The following sections describe the different areas your documentation should cover.

General Information

Include the following general information about your application:

- A list of resource types
In the sample Discussion application described above, resource types would be discussion and topic.
- The HTTP methods your application supports for each resource
In the sample Discussion application, the discussion resource supports GET, but the topics resource supports DELETE, GET, POST, and PUT.

Information About Each Request

Include the following information about each request:

- The format of the request URL
For example, documentation for the Discussion application could provide a list of possible client requests:
 - Return general information about the Discussion application:
`GET /rest/discussion HTTP/1.1`
 - Return a list of all topics contained in the database:
`GET /rest/discussion/topic HTTP/1.1`
 - Display entries made by participant Robertson in 2009 to topic 3419:
`GET /rest/discussion/topic/3419?year=2009&name=Robertson HTTP/1.1`
- Which request header fields are required or optional and how your application responds to them. For example, the Discussion application might specify the following information to explain which header fields it accepts and how it responds to them:

- **Authorization.** The Discussion application accepts BASIC and DIGEST authorization. All requests must include an Authorization header.
- **Content-Type.** Clients should include a Content-Type header with all requests. Acceptable Content-Type values for requests that contain a body are application/json, text/xml, text/html, and text/plain. For more information about Content-Types, see *webMethods Integration Server Administrator's Guide*.
- **Accept.** Clients can optionally supply an Accept header to indicate the Content-Type they want the response to use. When you specify the Content-Type for the Accept header, Integration Server uses the content handler registered to that Content-Type to respond to the request. For example, if the content handler is application/json, Integration Server responds to the request with JSON content. Acceptable values are application/json, text/xml, and text/html. If no Accept header is specified in the request, the response will use text/xml. For more information about the Accept header, see *webMethods Integration Server Administrator's Guide*.
- Whether a body is required and what structure the body should have.

Documentation for the Discussion example might provide the following examples to illustrate body structure:

Example 1: Creating a new topic

Request:

```
POST /discussion/topic HTTP/1.1
Host: IS_server:5555
Authorization: BASIC <your-credentials>
Content-Length: <request-body-length>
Content-Type: text/xml; charset=utf-8
```

Response: If the request was valid, the Discussion application will respond with the following:

```
HTTP/1.1 201 Created
Content-Length: 0
ETag: 32619
Location: http://host/discussion/topic/32619
```

Example 2: Adding an entry to an existing topic

Request:

```
PUT /discussion/topic=36219 HTTP/1.1
Host: IS_server:5555
Authorization: BASIC <your-credentials>
Content-Length: 17
Content-Type: text/xml; charset=utf-8
comment=I+agree
```

Response: If the request was valid, the Discussion application will respond with the following:

```
HTTP/1.1 200 OK
Content-Length: 0
Location: http://host/discussion/topic/36219?comment=2
```

Information About Responses

Your documentation should include the following to describe the response that corresponds to each request:

- A list of HTTP Status-Codes and Reason-Phrases the application returns and the circumstances under which it returns them. For a list of possible responses that you can code your application to return, refer to <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.2>.
- A list of the response header fields you return and what they mean in the context of your application.
- A description of what will appear in the body of the response.

Index

A

application services 18

C

configuration 19

converting an existing REST application 20

D

documentation

 using effectively 5

documenting your REST application 22

H

header fields of response to REST client 16

HTTP request methods

 supported 8

I

invoke directive 20

M

message body of response to REST client 16

P

processing directives

 invoke 20

 rest 12, 20

R

request messages

 format 12

response to REST client

 header fields 16

 message body 16

 status line 16

REST application

 directory structure 18

 setting up on Integration Server 18

REST application services 18

rest directive 12

 alternative name for 19

REST processing

 input parameters 12

 passing input to application services 12

 request format from REST perspective 12

 request parsing 12

 supported HTTP request methods 8

REST request messages 12

REST server

 setting up on Integration Server 18, 18

S

status line of response to REST client 16

Symbols

\$httpMethod input variable 18

\$path input parameter 14

\$resourceID input parameter 13

_default service 18

_delete service 18

_get service 18

_post service 18

_put service 18