1.1 Project/Component Working Name
GlassFish v3 HTTP Interface

1.2 Name(s) and e-mail address of Document Author(s)/Supplier:
Ludovic Champenois (Ludovic.Champenois@sun.com)
Jerome Dochez (Jerome.Dochez@sun.com)
Rajeshwar Patil (Rajeshwar.Patil@sun.com)

1.3 Date of This Document
05/07/2009

2. Project Summary
2.1 Project Description
Define and Implement HTTP interface for GlassFish. The HTTP interface proposed by this one-pager will enable GlassFish management by clients technologies such as browsers (JavaScript), JavaFX, PERL, Ruby etc., in addition to the Java clients it currently supports. This project will expose GlassFish Admin Commands via HTTP, and GlassFish management and monitoring API as HTTP REST urls.

2.2 Requirements
Provide REST API for GlassFish management (configuration and monitoring) to retrieve, delete, create and update backend config objects
Update/Create/Delete operations on admin config objects should be done on the backend whenever possible via admin commands so that changes in the backend configuration are validated and are done in a consistent way.
API should provide multiple representations such as XML, HTML, JSON etc. Should be able to plug-in more representations if needed.
API support should be pluggable/extensible. Module owners should be able to extend this API.

3. Problem Summary
3.1. Problem Area
Currently, GlassFish management API's cater to only Java technology clients. Today, numerous scripting technologies are popular and are adopted increasingly. By providing Rest like management interface we can attract the user community using these technologies. Management of GlassFish will no longer be limited to just Java or any particular Java API.

3.2 Justification
Exposing GlassFish in a technology-agnostic way enables and encourages the community to write management clients for GlassFish. Enable users to manage GlassFish using their language of expertise. REST interface exposes GlassFish management and monitoring through interoperable, scalable, and distributed API.

4. Technical Description
4.1 Details
This interface will provide HTTP API for GlassFish management and monitoring. This set of exposed urls constitutes new API for GlassFish administration in addition to existing API's. The API support will be provided through adapter interface. This HTTP interface can be divided into URLs which enable "configuration" of GlassFish, and those that enable "monitoring" of GlassFish. Configuration management API enables management of GlassFish through HTTP urls whereas monitoring HTTP API enables monitoring of GlassFish through HTTP urls.

4.1.1 Configuration Management

Configuration management API consists of Admin Commands HTTP API and RESTFul HTTP API.

4.1.1.1 Admin Commands HTTP Access

GlassFish v3 has an extensible admin commands backend framework that has the capability of exposing the commands via HTTP access (refer to https://glassfish.dev.java.net/nonav/v3/admin/planning/j109/admin-cli.html section 4.1.3.1 for details). Using the USER-AGENT request header, on can control the format of the output of the command. This control is implemented in a generic way so that the command developer does not need to worry about the formatting. Current formats are Text, text/html, XML, JSON and Jar Manifest.

4.1.1.1.1 URL Format

The Admin Command http urls has the following format-
`http://`***`host`***`:`***`port`***`/__asadmin/`***`command`***`?`***`parameters`***`.`***`extension`***
`host`: server host
`port`: administration port
`command`: name of the command to execute.
`parameters`: command parameters specified as http QUERY_STRING
`extension`: expected output representation (for example: json, xml, html)

In case of command line urls help for the command is supported using query param, `help`
`http://`***`host`***`:`***`port`***`/__asadmin/`***`command`***`?help=`true

4.1.1.1.2 Some examples of Command Line HTTP API

Example 1 Get the list of http listeners

```
http://localhost:4848/rest-resources/list-http-listeners
```

Output:

```
<html><body><h1>GlassFish list-http-listeners AdminCommand command report</h1><br>
Exit Code : SUCCESS
<hr>
<h2>list-http-listeners:</h2>
<h3>http-listener-1</h3>
<h3>http-listener-2</h3>
<h3>admin-listener</h3>
<hr>
</body></html>
```

Different output representation can be obtained by using the appropriate extension(see Example 2). Extensions for the supported representations are `.xml`, `.json` and `.html`. If no extension is specified the default representation returned, as in this example, is XML.

Example 2 Get the list of http listeners in JSON format

```
http://localhost:4848/rest-resources/list-http-listeners.json
```

Output:

```
{  "command":"list-http-listeners AdminCommand" , "exit_code":"SUCCESS" , "result" : [ {
"name":"http-listener-1" } , { "name":"http-listener-2" } , { "name":"admin-listener" } ] }
```

This example shows the structure of the output of a command:
"command" which contains the name of the command executed
"exit_code" which returns the status of the execution (SUCCESS, FAILURE)
"result" which is an collections of lines emitted by the command in a hierarchic way.

4.1.1.2 RESTFul HTTP API

More sophisticated clients, such as Admin UI, may need fine grained access to configuration. To satisfy the need of such clients, this API will provide direct access to the whole configuration tree. All the backend config beans will be exposed as JAX-RS resources enabling access to their attributes and parent-child relationships.

A RESTful web service is based on the following principles:

* Resources and representations. Instead of providing just one endpoint for a particular web service and having that endpoint perform various operations, you provide access to resources. A resource is a single piece of a web application that is made accessible to clients.

* Addressability and connectedness. Resources have their representations, but providing representations of resources would be useless if you could not address them. In REST, every resource must have at least one address, that is, one URI. To address the resource, you simply specify the URI. This concept is called "addressability". By publishing a web application, you introduce many different URIs that are connected to each other. Because of that connectedness, the only URI you need to give to your clients is one URI called the "bootstrap URI".

* Uniform interface. Even if you make resources available through URIs and representations to exchange between the client and server, it still does not allow you to establish communication. You need a communication protocol/interface to use. In a REST architecture, such an interface must be uniform. It means that whatever URI you access, the interface should be the same. For instance, on the World Wide Web no matter what URI (resource address) you enter, your web browser simply uses an HTTP GET method to retrieve a corresponding web page (resource representation) and displays it.

* Statelessness. Statelessness means that a web application is not responsible for keeping any information about the state of its clients. REST does not encompass the concept of HTTP sessions. The client is responsible for tracking its own actions (if it needs to). The service maintains its resources and provides a uniform interface to the clients.

The RESTFul architecture applies well to expose access to the generic @Configured ConfigBeans layer described in HK2 (http://wiki.glassfish.java.net/Wiki.jsp?page=GlassfishV3Final_hk2_overview#section-GlassfishV3Final_hk2_overview-4.1.4ConfigurationSupport).

HK2 Configured objects represent configuration that will be used by the server, it is usually store in an xml file (like the domain.xml in glassfish). Such configuration is described using interfaces definition that are binding interfaces (like JAXB interfaces) using annotations in the Config module. An interface is annotated with the @Configured annotation, and xml attributes can be described using the @Attribute annotation while xml sub elements are defined with a @Element annotation. @Element can return one single object of a List of objects.

Example :

```
@Configured
public interface SomeConfig extends ConfigBeanProxy {
   @Attribute
   String getAttributeName();
   @Element
   SubConfig getSub();
}
```

GlassFish v3 backend configuration is mostly described in the admin/config-api module (and for Grizzly configuration, in the Grizzly component). The root of all the ConfigBeans is the "Domain" object that holds a reference to all the configbeans available in the system.

What is proposed here is to provide access to RESTFul resources that are mapped to the configbeans so that the content of these configbeans can be accessed, modified, created or deleted using the RESTful concepts. The Jersey implementation (https://jsr311.dev.java.net/) which is available now in the Java EE 6 specification is used via a special Grizzly connector to avoid a dependency to a servlet container. The mapping is as follow:

- Each @Configured configuration will have a REST resource that exposes the list of @Attribute of this configuration (HTTP GET)

- Navigation rules using Jax-RS sub-path will be available to navigate from one configuration resource to a child configuration resource or to a List of children.
- HTTP methods (POST, PUT, DELETE) on these resources will create , update or delete the corresponding configbeans.
- An optional redirect to actual Admin Command is possible to implement these POST, PUT, DELETE methods to guarantee the behavior of the actions.

The top level URL for accessing the entire resources is http://machinename:adminport/rest-resources. For example, to get the list of Attributes and child Elements of the Domain configbeans, the following URL will be used: http://localhost:4848/rest-resources/domain. This is the main entry point to access to all the sub configurations.

When the child Element is a List of other ConfigBeans, a corresponding resource will be exposed for this collection. To select one element among N inside this Collection, one must specifies the value of a Key attribute (@Attribute annotation supports the definition of a Key). The value of the Key is passed as a sub-path

Open Issue: what if the key contains the "/" char. (Question posted to Paul Sandoz and JFA)

The following picture shows all the resources and sub resources available. It is using JavaScript test client, driven from the application.wadl wadl file generated by the Jersey backend to describe the REST web application. Tree nodes like {Name} represents a variable (the node element is in fact a List of configs, each one uniquely identified by a key, in this example, the name of the config):

image

Multiple output formats are possible:

4.1.1.2.1 Output Formats

RESTful configuration API's will leverage existing support for `XML`, `HTML` and `JSON` representations. Additional representations can be plugged-in, if there is a requirement.
For the supported representations the output format will be as defined below.

As you can see in the following examples, the Domain configbean has log-root, application-root and version as attributes. We can see the values. Default values will be also be shown. The Domain also has children resources: applications, resources, servers, configs and property. Navigating between parent-children resources is very natural using URL sub-paths.

Open Issue: do we want to expose as well the fact that some values are Default? (Maybe needed by the Admin GUI team...)
Open Issue: do we want to expose as well the list of possible values (and the type) if they are defined at the config bean level (Maybe needed by the Admin GUI team...)
Ludo: I think we should...

JSON format

```
{"type":{attributes}, "resources":[values]}
```

type: the name of the resource
attributes: one or more Attribute name- Attribute value pairs separated by comma(,)
values: one or more child resource urls separated by comma(,)

Example

http://localhost:4848/rest-resources/domain

Output:

{"domain":{"log-root":"/installs/v3/domains/domain1/logs", "application-root":"/installs/v3/domains/domain1/applications", "version":"10.0"},
"resources": ["http://localhost:4848/rest-resources/domain/applications", "http://localhost:4848/rest-resources/domain/resources", "http://localhost:4848/rest-resources/servers", "http://localhost:4848/rest-resources/domain/configs", "http://localhost:4848/rest-resources/domain/property"]
}

XML format

```
<type attributes>
  <resource>resource url</resource>
  ...
</type>
```

type: type of the resource
attributes: name-value pairs separated by space(blank)
resource url: url of the sub-resource

Example

http://localhost:4848/rest-resources/domain

Output:

<domain log-root="/installs/v3/domains/domain1/logs" application-root="/installs/v3/domains/domain1/applications">
  <resource>http://localhost:4848/rest-resources/domain/applications</resource>
  <resource>http://localhost:4848/rest-resources/domain/resources</resource>

```
    <resource>http://localhost:4848/rest-resources/servers</resource>
    <resource>http://localhost:4848/rest-resources/domain/configs</resource>
    <resource>http://localhost:4848/rest-resources/domain/property</resource>
 </domain>
```

HTML format

```
<html><body>
<h1>type</h1><hr>
  <h2>attribute: value</h2>
  ...
<h1>resources</h1><hr>
  <h2>resource url</h2>
  ...
</body></html>
```

type: type of the resource
attribute: name of the attribute
value: value of the attribute
resource url: url of the sub-resource

Example

http://localhost:4848/rest-resources/domain

Output:

```
<html><body>
<h1>domain</h1><hr>
    <h2>log-root: /installs/v3/domains/domain1/logs</h2>
    <h2>application-root: /installs/v3/domains/domain1/applications</h2>
<h1>resources</h1><hr>
    <h2>http://localhost:4848/rest-resources/domain/applications</h2>
    <h2>http://localhost:4848/rest-resources/domain/resources</h2>
    <h2>http://localhost:4848/rest-resources/servers</h2>
    <h2>http://localhost:4848/rest-resources/domain/configs</h2>
    <h2>http://localhost:4848/rest-resources/domain/property</h2>
</body></html>
```

4.1.1.2.2 URL Format

The configuration management URLs can have the following format-
`http://`***`host`***`:`***`port`***`/rest-resources/`***`path`***

`host``:` server host
`port``:` administration port
`path``:` identifies the resource

The ***`path`*** for the given object can be formed by traversing the tree from root to that node. Below is the example of the path to refer to `Address` attribute of `listener1` from `server-config`.
`/domain/configs/config/`*`server-config`*`/http-service/http-listener/`*`listener1`*

The HTTP url for `Address` attribute referred by above path will be-
`http://{host}:{port}/rest-resources/domain/configs/config/server-config/http-service/http-listener/listener1`

It is possible to discover the paths either by querying the application.wadl XML file generated by Jersey at the top of the web application (http://localhost:4848/rest-resources/applications.xml of by navigating from the root url (/rest-

resources/domain) and obtaining the name of the sub resources which are described on the current resource.

4.1.1.2.2.1 Methods

Generic configuration API will support `get`, `put`, `post`, `delete` and `options` methods.

`POST /rest-resources/{path} HTTP/1.1`	Create the given object
`GET /rest-resources/{path} HTTP/1.1`	Get the given object
`PUT /rest-resources/{path} HTTP/1.1`	Update the given object
`DELETE /rest-resources/{path} HTTP/1.1`	Delete the given object
`OPTIONS /rest-resources/{path} HTTP/1.1`	Get meta-data of the resource

Sometimes, it might not be enough to create, update or delete configbeans data without further validations than the validation rules defined at the @Component HK2 level. For example, deleting an entry in the List of Applications should be in fact mapped to a real "un-deploy" Admin command that takes care of all the business logic related to the undeploy process, that includes deleting the entry of the application in the configuration. To accommodate these cases, a new annotation @RestRedirects is introduced:
For example:

```
@Configured
@RestRedirects(
      {
        @RestRedirect(opType= RestRedirect.OpType.DELETE, commandName="undeploy"),
        @RestRedirect(opType= RestRedirect.OpType.POST, commandName = "redeploy")
      }
)
public interface Application extends ConfigBeanProxy, Injectable, Named, PropertyBag {
....
```

is a way to redirect the DELETE operation on an application to the undeploy command, and the POST operation to the redeploy command. In such cases, the payload of the HTTP operations will be the list of Key/Value pair of all the command parameters. For the DELETE operation, the operand will be determined by the Attribute Key of the resource. (for example, in the case of an Application, the "name" of this application is the key to be used to undeploy this application.

Below is a simple curl example on how to modify the "locale" Attribute of the Domain object, using the PUT method:

```
curl -i --data "{\"locale\":\"mylocale\"}" -H Content-type:application/json -X PUT http://localhost:4848/rest-resources/domain/
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked
Date: Fri, 08 May 2009 03:47:35 GMT
```

Below is another example for a GET that shows the modified "locale" attribute:

```
curl -HAccept:application/xml http://localhost:4848/rest-resources/domain/
<Domain log-root="${com.sun.aas.instanceRoot}/logs" application-root="${com.sun.aas.instanceRoot}/applications" locale="mylocale" version="ludo-private">
   <resource>http://localhost:4848/rest-resources/domain/configs</resource>
   <resource>http://localhost:4848/rest-resources/domain/resources</resource>
   <resource>http://localhost:4848/rest-resources/domain/servers</resource>
   <resource>http://localhost:4848/rest-resources/domain/property</resource>
```

<resource>http:///localhost:4848/rest-resources/domain/applications</resource>
<resource>http://localhost:4848/rest-resources/domain/system-applications</resource>

HK2 Config backend as some validation in place, so that sending the wrong set of data will trigger an error like:

curl -i --data "{\"BUGGYlocale\":\"mylocale\"}" -H Content-type:application/json -X PUT
http://localhost:4848/rest-resources/domain/
HTTP/1.1 500 Internal Server Error
Content-Type: text/plain
Transfer-Encoding: chunked
Date: Fri, 08 May 2009 03:49:31 GMT
Connection: close

javax.ws.rs.WebApplicationException: org.jvnet.hk2.config.TransactionFailure: Unknown property name BUGGYlocale on interface com.sun.enterprise.config.serverbeans.Domain
at org.glassfish.admin.rest.TemplateResource.updateConfig(TemplateResource.java:229)
at org.glassfish.admin.rest.TemplateResource.updateEntity(TemplateResource.java:143)

Such errors will return a code 500: Internal Server Error status.

If incorrect content types are used, the JAX-RS JerSey backend will correctly report the error as well (415 Unsupported Media Type):

curl -i --data "{\"locale\":\"mylocale\"}" -H Content-type:application/buggytype -X PUT
http://localhost:4848/rest-resources/domain/
HTTP/1.1 415 Unsupported Media Type
Content-Type: text/plain; charset=iso-8859-1
Content-Length: 0
Date: Fri, 08 May 2009 03:54:00 GMT

4.1.1.2.2.2 Hooking Admin Commands to ConfigBeans resources

In the previous section, we mentioned a way to redirect POST, PUT, DELETE methods to admin commands. Another possibility which is under investigation is to declare in an annotation admin commands in the confibean interface. This declaration could be used by our generator and dynamic command introspection and command invocation to add new sub resources under config bean resources. For example, for the Domain configBean, we could add the "restart", "stop-domain", "log", "clone" etc, commands. (Note that log or clone would have to be new commands, they do not exist yet). Some of these commands of course contain parameters (mandatory and optional, many times, the mandatory operand is the config bean key). These parameters would be described as HTTP parameters for the command resource.
Command resources would be hosted under the "commands" path of a config bean resource (or another unique path we decide).

To stop the domain, a GET would be issued at:
curl http:///localhost:4848/rest-resources/domain/commands/stop-domain
Operand parameters would be added as:
curl http:///localhost:4848/rest-resources/domain/commands/stop-domain?echo=true

The output of the command execution would return the status of the command, and when it makes sense the result of the command (i.e the modified REST resource) in the format specified (xml, html, json).

One interesting command would be the /domain/commands/log that could tail the server log. A JavaScript client could use some pooling (like it is currently done in the the Hudson Web Application to tail process execution output in the Hudson Console) too continuously display the log of the server.

Since the command parameters are described as @WebParams in the command resource, the metadata file application.wadl maintained by the Jersey backend will described them so that they can be discovered by a client, and some client side validation can be performed as well.

To deploy an application the first time, such URL would be used:
*curl http://localhost:4848/rest-resources/domain/*applications/application/deploy?path=/Users/ludo/WebApplication3/build/web&name=WebApplication3&force=true&property=keepSessions=true

To redeploy this WebApplication3 application, one could use:
*curl http://localhost:4848/rest-resources/domain/*applications/application/WebApplication3/deploy&property=keepSessions=true

In this last example, the path already points to the WebApplication3 resource, so it is not necessary to pass the application name to the deploy command.

4.1.1.2.3 Some examples of RestFul HTTP API

Example 1 Get http listener named `http-listener-2`

GET `/rest-resources/domain/configs/config/server-config/http-service/http-listener/http-listener-2 HTTP/1.1`
`Accept: application/json`

Output (JSON):

```
{"http-listener":{"http-listener-port":8181, "id":"http-listener-2", "address"=0.0.0.0,
"security-enabled":true, "default-virtual-server":"server", "server-name":"" },
 "resources":[http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/http-listener/http-listener-2//ssl]
}
```

GET `/rest-resources/domain/configs/config/server-config/http-service/http-listener/http-listener-2 HTTP/1.1`
`Accept: application/xml`

Output (XML):

```
<http-listener http-listener-port="8181" id="http-listener-2" address="0.0.0.0" security-enabled="true" default-virtual-server="server" server-name:"">
  <resource>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/http-listener/http-listener-2/ssl</resource>
</http-listener>
```

`http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/http-listener/http-listener-2`

GET `/rest-resources/domain/configs/config/server-config/http-service/http-listener/http-listener-2 HTTP/1.1`
`Accept: text/html`

Output (HTML):

```
<html><body>
<h1>http-listener</h1><hr>
<h2>http-listener-port: 8181</h2>
<h2>id: http-listener-2</h2>
```

```
<h2>address: 0.0.0.0</h2>
<h2>security-enabled: true</h2>
<h2>default-virtual-server: server</h2>
<h2>server-name: </h2>
<h1>resources</h1><hr>
<h2>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/http-
listener/http-listener-2/ssl</h2>
</body></html>
```

Example 2 Get `http-service` object using RESTful HTTP API. Addressed resource does not have any attributes.

`http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service`

```
GET /rest-resources/domain/configs/config/server-config/http-service HTTP/1.1
Accept: application/json
```

Output (JSON):

```
{ "http-service":{},
"resources":["http://localhost:4848/rest-resources/domain/configs/config/server-config/http-
service/access-log", "http://localhost:4848/rest-resources/domain/configs/config/server-
config/http-service/http-listener/http-listener-1",
"http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/http-
listener/http-listener-2",
"http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/http-
listener[admin-listener]",
"http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/keep-
alive",
"http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/http-file-
cache",
"http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/virtual-
server/server",
"http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/virtual-
server/__asadmin"]
}
```

`http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service`

```
GET /rest-resources/domain/configs/config/server-config/http-service HTTP/1.1
Accept: application/xml
```

Output (XML):

```
<http-service>
  <resource>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-
service/access-log</resource>
  <resource>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-
service/http-listener/http-listener-1</resource>
  <resource>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-
service/http-listener/http-listener-2</resource>
  <resource>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-
service/http-listener/admin-listener</resource>
  <resource>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-
service/keep-alive</resource>
  <resource>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-
service/http-file-cache</resource>
  <resource>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-
service/virtual-server/server</resource>
  <resource>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-
service/virtual-server/__asadmin</resource>
</http-service>
```

`http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service`

```
GET /rest-resources/domain/configs/config/server-config/http-service HTTP/1.1
Accept: text/html
```

Output (HTML):

Output (HTML):

```
<html><body>
<h1>http-service</h1><hr>
<h1>resources</h1><hr>
<h2>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/access-log</h2>
<h2>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/http-listener/http-listener-1</h2>
<h2>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/http-listener/http-listener-2</h2>
<h2>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/http-listener[admin-listener]</h2>
<h2>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/keep-alive</h2>
<h2>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/http-file-cache</h2>
<h2>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/virtual-server/server</h2>
<h2>http://localhost:4848/rest-resources/domain/configs/config/server-config/http-service/virtual-server/__asadmin</h2>
</body></html>
```

4.1.2 Monitoring

Monitoring HTTP API enables monitoring of GlassFish through HTTP urls. This support enable GlassFish monitoring through new tools and technologies. For example, using these API's cloud computing provider can charge customers based on number of sessions, number of requests etc. This API essentially exposes telemetry objects as HTTP urls. This RESTFul API will provide direct access to the whole monitoring tree.

4.1.2.1 Output Formats

Monitoring API supports following output formats `XML`, `HTML` and `JSON`. Additional representations can be plugged-in/provided, if there is a requirement.
For the supported representations the output format will be same as defined for config objects(see section **4.1.1.2.1** above) Using this, the output format for specific data types will be-

Example 1 Output for Statistic object

in JSON format:

```
{"{statistic}":{"name"="xyz", "unit"="xyz", "description"="xyz", "start-time"=xyz, "last-sample-time"=xyz}}
```

in XML format:

```
<statistic name="xyz" unit="xyz" description="xyz" start-time="xyz" last-sample-time="xyz">
```

in HTML format.

```
<html><body>
<h1>statistic</h1>
<hr>
<h2>name: xyz</h2>
<h2>unit: xyz</h2>
<h2>description: xyz</h2>
<h2>start-time: xyz</h2>
<h2>last-sample-time: xyz</h2>
</body></html>
```

Example 2 Output for Range Statistic object

JSON

```
{"range-statistic":{"high-water-mark"=xyz, "low-water-mark"=xyz, "current"=xyz, "name"="xyz",
"unit"="xyz", "description"="xyz", "start-time"=xyz, "last-sample-time"=xyz}}
```

XML

```
<range-statistic high-water-mark="xyz" low-water-mark="xyz" current="xyz"
name="xyz" unit="xyz" description="xyz" start-time="xyz" last-sample-time="xyz">
```

HTML

```
<html><body>
<h1>range-statistic</h1>
<hr>
<h2>high-water-mark: xyz</h2>
<h2>low-water-mark: xyz</h2>
<h2>current: xyz</h2>
<h2>name: xyz</h2>
<h2>unit: xyz</h2>
<h2>description: xyz</h2>
<h2>start-time: xyz</h2>
<h2>last-sample-time: xyz</h2>
</body></html>
```

4.1.2.2 URL Format

The monitoring URLs can have the following format-
`http://`***`host`***`:`***`port`***`/rest-resources/`***`monitoring/path`***

`host``:` server host
`port``:` administration port
`path``:` identifies the resource

4.1.2.3 Methods

Monitoring API will support `get` and `options` methods.

`GET /__monitoring/{path} HTTP/1.1`	Get the given object
`OPTIONS /__monitoring/{path} HTTP/1.1`	Get meta-data of the resource

4.1.3 Extensibility

This API support leverages the existing extensibility support provided by configuration CLI, monitoring and the configuration.
The way the current resources are created is by running a java code generator that introspects the ConfigBeans interfaces (getting the list of attributes, elements, annotations, validation rules, REST redirects rules). In the very near future, this generator will be replace by an in process ASM generations of REST resources that will map the current configbeans for the server. This way, when a GlassFish module developer adds new ConfigBeans (for example to manage Grizzly or to manage the jRuby container), the corresponding REST resources will be correctly added at the right location under the resource tree. The only source of information is all located in the configbean interface definition. Other metadata that could be use at generation time should always be added in these central interfaces that represent the model for all the management backend.

4.2 Bug/RFE Numbers(s)
None. These are new APIs.

4.3 In Scope

4.4 Out of Scope
This API will not provide any support for notification.
Role based administration will not be supported for JavaOne release.
This API may not provide higher level APIs where by you can fetch multiple configuration (or telemetry) objects in single request.

4.5 Interfaces
4.5.1 Exported Interfaces
All of the command line HTTP urls and the RESTFul HTTP urls for all of the config beans and telemetry objects. The resources structure and payload (for update or create) are driven from the corresponding ConfigBean data and the possible Admin Commands used for these operations.
Jersey will always maintain the application.wadl file that describes all the resources and operations and types available for each resource. For this wadl file, it is possible to generate Java Clients (https://wadl.dev.java.net/) or use the RestFul Web Services tester tools from IDEs like NetBeans or Intellij).

4.5.2 Imported Interfaces
GlassFish API
GlassFish Config API and Admin Command APIs
HK2 API (ConfigSupport, @Configured, @Attribute, @Element)

4.6 Doc Impact
All the interfaces need to be documented.

4.7 Admin/Config Impact
Significant changes to Admin GUI code to use this new API.

4.8 HA Impact
None.

4.9 I18N/L10N
I18N/L10N support can be provided by central facility provided by config and telemetry beans. This support can be targeted for next release.

4.10 Packaging & Delivery
Generic configuration and monitoring API support will be part of kernel module (`com.sun.enterprise.v3.admin` and `com.sun.enterprise.v3.common` package).
Command line HTTP API implementations will continue to get bundled in *-management IPS packages as defined by the packaging and layout specification.

4.11 Security
Security support for RESTFul APIs will be same as provided by existing security support for command line API.
Authorization through admin user name and password.

4.12 Compatibility Impact
None. These are new APIs.

4.13 Dependencies

Asadmin commands (https://glassfish.dev.java.net/nonav/v3/admin/planning/j109/list-of-commands.html)

5 Reference Documents

Path Specification	https://glassfish.dev.java.net/nonav/v3/admin/planning/V3Changes/V3_AdminCLI.html
Project HK2	https://hk2.dev.java.net/
HTTP Specification	http://www.ietf.org/rfc/rfc2616.txt
JSON ORG	http://json.org/
REST Design	http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
Project Jersey	https://jersey.dev.java.net/

6 Schedule

6.1 Project Availability

API for config objects will be delivered by Milestone 2 (4/10/09). Monitoring API's delivered by Milestone 3 (5/22/09).