# Developing Custom Portlets

***Overview***

This page is a developers guide to creating GroundWork portlets.

CONTENTS

RELATED RESOURCES

- GroundWork Portlets
- Deploying Custom Portlets (VIDEO)

WAS THIS PAGE HELPFUL?

Leave Feedback

# 1.0 GroundWork Portlets — A Quick Developer Guide

GroundWork Monitor 7, is an industry leading open platform for unified monitoring. The GroundWork user interface is a modern web portal built upon a foundation of standard, open source tools and frameworks.

This document is intended to set forth best practices for developing custom extensions to the GroundWork Portal user interface. We'll guide through developing your own dashboards extensions to your GroundWork Portal. These extensions are known as portlets which are standardized Web UI components. This document provides a tutorial for building two portlets to help you get up to speed in developing your own extensions to the GroundWork Portal. We describe all the necessary steps for creating custom portlets with Java and JavaScript. Additionally, we'll demonstrate retrieving the data for the portlet via the Collage REST API with JavaScript. For a complete guide to developing portlets, there are several books available including this online reference, Portlets and Apache Portals.

To summarize, you will learn:

- About the Collage REST API and how to use this API to retrieve and display monitoring data
- How to develop a Java Portlet with server-side Java code, as well JSP and JSTL
- How to develop a Java Portlet with client-side JavaScript and REST APIs
- How to deploy your portlets to the GroundWork Monitor server

Both the Java and JavaScript examples will make use of the GroundWork Collage REST API.

Target Audience:

- We expect the reader to have a general programming background. Obviously having Java and JavaScript experience is a big plus. The target audience is integrators and developers who want to develop custom UI extensions to the GroundWork Portal.

System Requirements:

- Requirements for building, deploying and running the examples:
  - Java 7 JDK
  - Maven 3 (preferable 3.2.1 or higher)
  - GroundWork Monitor Portal 7.0.2

# 2.0 Developing Custom Portlets with Java

Our first example is a pure Java solution. This solution will use the Java REST API Client to retrieve monitoring data.

## 2.1 Introduction to the Java Client for the Collage REST API

REST (**RE**presentational **S**tate **T**ransfer) is a distributed computing style based on the architectural successes of the web. The Collage REST API for GroundWork Monitor is a set of distributed web service APIs adhering to the design principles of REST. A Java Client is provided with the tutorial project. The REST API from Java is very easy to use.

This Java Client will return results as Data Transfer Objects (DTOs). These DTOs are simple, serializable Java beans. All entities in the GroundWork Monitor model are accessible via the Java Client. Here are some examples of using the HostClient:

**List Hosts**

```
        HostClient hostClient = new HostClient("http://localhost/api");
List<DtoHost> hosts = hostClient.list();
```

**Lookup Hosts**

```
        HostClient hostClient = new HostClient("http://localhost/api");
DtoHost host = hostClient.lookup("myHost");
```

**Query Hosts**

```
        HostClient hostClient = new HostClient("http://localhost/api");
List<DtoHost> hosts = client.query("day(lastCheckTime) = 22 and month(lastCheckTime) = 5 and
minute(lastCheckTime) > 43 order by lastCheckTime");
```

**Query Hosts with Paging**

```
        HostClient hostClient = new HostClient("http://localhost/api");
List<DtoHost> hosts = client.query("property.LastPluginOutput like 'OK%'", firstRecord,
numberOfRecords);
```

**Hosts DTO Example**

```
        List<DtoHost> hosts = client.query("(property.ExecutionTime between 10 and 3500 and
(monitorStatus <> 'UP')) order by property.ExecutionTime");
        for (DtoHost host : hosts) {
          // example Getters
String hostName = host.getHostName();
          String status = host.getMonitorStatus();
          Date lastCheck = host.getLastCheckTime();
          host.getProperty("LastPluginOutput");
          executionTime = host.getProperty("ExecutionTime");

          // associations (depth dependent)
List<DtoService> services = host.getServices();
          List<DtoHostGroup> hostGroups = host.getHostGroups();
       }
```

Additionally, there are both Business/Bulk APIs and CRUD operations for inserting, updating and deleting GroundWork entities not covered in this tutorial. A brief list of APIs include:

- Authorization
- Hosts
- Services
- Host Groups, Service Groups, Custom Groups
- Events
- Devices
- Statistics
- Graphs
- MetaData
- Notifications
- Performance Data
- Downtime
- Device Templates

- BlackList

To learn more about the GroundWork REST API, see the online documentation: RESTful API Documentation

## 2.2 Introduction to the Portlet API

The Portlet interface is used by a Java portal to invoke portlets. Every portlet has to implement this interface, either by directly implementing it, or by using an existing class implementing the Portlet interface.

A portlet is a Java technology-based web component. It is managed by the portal and processes requests and generates dynamic content as response. Portlets are used by portals as pluggable user interface components.

The portal instantiates portlets, manages their life-cycle and invokes them to process requests. The life-cycle consists of:

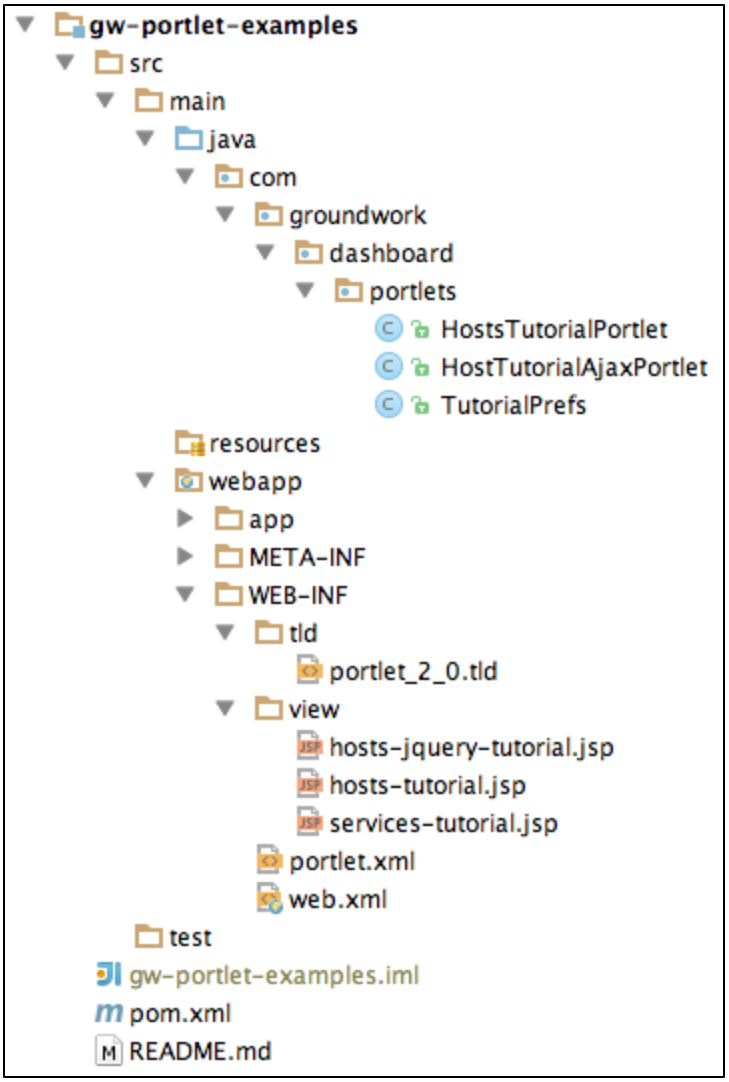
- Initializing the portlet using the init method
- Request processing
- Processing actions
- Taking the portlet out of service using the destroy method

We will covering two of these phase in our portlet examples: Request processing and Processing actions

## 2.3 Overview of the Tutorial Project

The Tutorial is zipped up and can be downloaded from here: gw-portlet-examples.zip

The project structure is a standard Maven directory structure. You can view the directory structure in the image below. The Java files are found under src/main/java. The sample source code is found under the package com.groundwork.dashboard.portlets. All web resources, such as CSS, Javascript, and JSP (Java Server Pages) are all found under src/main/webapp. The files you will be looking at are in src/main/webapp/WEB-INF/view/.jsp{*}. Also, there are two Java deployment descriptors under src/main/webapp/WEB-INF which are web.xml and portlet.xml.




### 2.3.1 Building the project

This project is built with the industry standard Maven open source build tool. To build the application, type:

```
cd gw-portlet-examples
mvn clean install
```

This command should complete with the message "BUILD SUCCESSFUL". The build creates a Java web application (WAR) file. WAR files are packaged in a standard way so that they can be deployed to any Java application server. Look in the target directory, the war file can be found there:

```
gw-portlet-examples.war
```

First, lets look at the pom.xml file found in the root of your project. This file contains the declarative Maven build instructions. All of the project dependencies on third-party Java libraries are listed here, for example:

```
 <dependencies>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>${javax.servlet.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>javax.servlet.jsp-api</artifactId>
            <version>${javax.servlet.jsp.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>${javax.servlet.jstl.version}</version>
        </dependency>
...
```

All of these dependencies are downloaded over the internet. Besides the standard internet locations, we've added several other remote repositories including GroundWork repositories:

```
    <repositories>
        <repository>
            <id>groundwork-public-repository</id>
            <name>Groundwork Repository</name>
            <url>http://archive.groundworkopensource.com/nexus/content/repositories/releases/</url>
<releases>
                <updatePolicy>never</updatePolicy>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
...
```

You will deploy this file to the GroundWork server in the Deployment section at the end of this document.

## 2.4 The Hosts Portlet Tutorial

Open up the tutorial project in your favorite IDE such as Eclipse or IntelliJ. In this tutorial, we are going to work with a portlet which shows a list of all Hosts in the system in an HTML table. Each row in the table will have a hyperlink to the services for the given host. When the user clicks on that link, a second page is displayed with all services listed for the select host.

In this lesson, we will work with one Java class and JSP files:

- com.groundwork.dashboard.portlets.HostsTutorialPortlet.java
- hosts-tutorial.jsp
- services-tutorial.jsp

Additionally, we will be using:

- The Collage REST API Client Library for retrieving monitoring data from the GroundWork server
- Data Transfer Object classes, which are simple serializable bean classes, will be returned by the Collage REST API Client Library

### 2.4.1 doView

In HostsTutorialPortlet.java, we are using GroundWork REST API to get a list of all hosts in the system. The list is retrieved in the portlet's doView method and prepared for rendering on a browser. The doView method is called by the portal whenever the portlet is rendered on the page. Here is the doView method:

```
    @Override
    public void doView(RenderRequest request, RenderResponse response) throws PortletException,
IOException {
        response.setContentType("text/html");
        if (authenticate(request, response)) {
            String hostName = (String)PortletMessaging.consume(request, "hostName");
            String view = VIEW_HOSTS;
            if (hostName == null) {
                request.setAttribute("hosts", retrieveHosts(request));
            }
            else {
                request.setAttribute("services", PortletMessaging.consume(request, "services"));
                request.setAttribute("hostName", hostName);
                view = VIEW_SERVICES;
            }
            PortletRequestDispatcher dispatcher = getPortletContext().getRequestDispatcher(view);
            dispatcher.include(request, response);
        }
    }
```

### 2.4.2 Step by Step Review of doView

**doView method**
The doView method is the standard entry point to rendering a portlet. This interface is defined by the Portlet API. A RenderRequest and RenderResponse object is passed into the portlet. The request holds all HTTP information about the requesting being made, such as request parameters and HTTP headers. The response is where you write your web content to, usually by including a JSP template or similar technology.

```
public void doView(RenderRequest request, RenderResponse response) throws PortletException,
IOException {
```

**authenticate**

All REST API clients must authenticate to GroundWork. We provide the plumbing for authenticating in the provided base class (DashboardPortlet.java). You just have to make sure that your /usr/local/groundwork/config/ws_client.properties has the correct username, password, and end point:

```
if (authenticate(request, response))
```

**ws_client.properties**

```
webservices_user=RESTAPIACCESS
webservices_password=7UZZVvnLbuRNk12Yk5H33zeYdWQpnA7j9shir7QfJgwh
credentials.encryption.enabled=true
```

If you fail to authenticate, the base class will display a message to the portlet output. You should not render your portlet if authentication fails.

**Passing Attributes to View**
Here is how you make Java objects visible in your JSP:

```
  request.setAttribute("hosts", retrieveHosts(request));
```

The setAttribute method on the Portlet request makes Java objects easily accessible in your JSP template. In this case, we have taken the result of the retrieveHosts() method, which is a List of DtoHost plain old Java objects. We'll see how easy it is to access these objects in a JSP template in the next section.

**Dispatching to a JSP**
The Portlet API provides standard methods for including the merged content of a JSP into your portlet. Commonly, you will you use request dispatchers. You will want to use the include method of the dispatcher:

```
PortletRequestDispatcher dispatcher = getPortletContext().getRequestDispatcher(view);
dispatcher.include(request, response);
```

### 2.4.3 The View: JSP and JSTL

Portlets should generate their HTML using a template language. In our example, we use the JSP template language for creating our dynamic HTML content. Open up the hosts-tutorial.jsp file in your editor. We skip the boiler-plate code and get down to how we generate our table of hosts:

**hosts-tutorial.jsp**

```
<table id="hostsTable">
    <tr>
        <th>Host Name</th>
        <th>Monitor Status</th>
        <th>App Type</th>
        <th>Last Plugin Output</th>
    </tr>
    <c:forEach items="${hosts}" var="host"  varStatus="loopStatus">
        <tr class="${loopStatus.index % 2 == 0 ? '' : 'alt'}">
            <td>
                <a href='<portlet:actionURL><portlet:param name='hostName'
value='${host.hostName}'/></portlet:actionURL>'>${host.hostName}</a>
            </td>
            <td>
                    ${host.monitorStatus}
            </td>
            <td>
                    ${host.appType}
            </td>
            <td>
                    ${host.getProperty("LastPluginOutput")}
            </td>
        </tr>
    </c:forEach>
</table>
```

Here is what the Hosts List looks like in your portlet:

| Host Name | Monitor Status | App Type | Last Plugin Output |
|---|---|---|---|
| Amalthea | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| Eucamon | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| Fileshare | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| GDMA_Collectd_GWMEE | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| Kore | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| LYSITHEA | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| OracleCentos32VM | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| SQL-2008 | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| TRAINING-2008 | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| Tarawa | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| Tinian | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| UCSPE | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| ad.demo.com | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| bdc.demo.com | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| bern | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| bernina.groundwork.groundworkopensource.com | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| cent-6-64-template | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| centos7-docker-template | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| cfengine | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| child-test.groundwork.groundworkopensource.com | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| d-exchange-2010 | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| demo-centos-6-64 | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| demo-production | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |

### 2.4.4 Step by Step Review of the JSP

**forEach loop**
The forEach tag is a Java JSTL tag for iterating of Java collections such as the List of DtoHosts we added in the doView, by setting an attribute named "hosts". You can see this variable accessed here in our JSP template:

**forEach and hosts variable**

```
    <c:forEach items="${hosts}" var="host"  varStatus="loopStatus">
```

For each host in the collection, we iterate over the list, effecting creating a table row for each host in the list:

**Generating dynamic html from hosts**

```
 <td>
                    ${host.monitorStatus}
            </td>
            <td>
                    ${host.appType}
            </td>
            <td>
                    ${host.getProperty("LastPluginOutput")}
            </td>
```

Note that you simply use dot notation to retrieve values from the DtoHost object. GroundWork dynamic properties can also be retrieved with the getProperty() method. Notice that the host name has some special processing:

**Creating an action link**

```
<a href='<portlet:actionURL>
    <portlet:param name='hostName'
value='${host.hostName}'/></portlet:actionURL>'>${host.hostName}</a>
```

Here we use a Portlet API tag to create an Action URL linking back to our portlet. Additionally, the hostName is passed in as a standard portlet request parameter. Next, lets look at what happens when this link is clicked on...

### 2.4.5 Actions

Portlet Actions are how you handle HTTP actions such as form submits. In this example, we show how to process the action generated by the actionURL tag in the previous section. Actions are executed in the Portlet API processAction method:

```
 @Override
    public void processAction(ActionRequest request, ActionResponse response) throws PortletException,
IOException {
        String hostName = request.getParameter("hostName");
        if (hostName != null) {
            PortletMessaging.publish(request, "services", retrieveServices(request, hostName));
            PortletMessaging.publish(request, "hostName", hostName);
        }
    }
```

First thing we do here is retrieve the hostName that was set on the actionURL tag in the JSP above.

```
 String hostName = request.getParameter("hostName");
```

With this hostName, we retrieve all the services for the given host and pass the list of host names to the doView method by publishing a portlet message:

```
   PortletMessaging.publish(request, "services", retrieveServices(request, hostName));
```

The doView method then consumes the message passed to it in the doView method:

```
request.setAttribute("services", PortletMessaging.consume(request, "services"));
```

Lets take a closer look at how we retrieve hosts and services...

### 2.4.6 Retrieving Hosts and Services with REST API

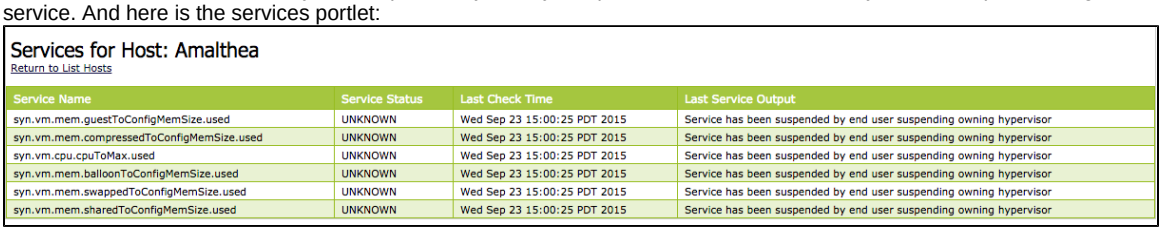
Hosts and Services are retrieved using the GroundWork REST API. Java clients are provided to greatly simplify the usage. The Java clients have a robust interface including CRUD and Query operations. Here we retrieve all hosts:

```
protected List<DtoHost> retrieveHosts(PortletRequest request) {
    HostClient hostClient = new HostClient(getRestEndPoint(request));
    return hostClient.list();
}
```

And for services, we retrieve all services for a given host:

```
protected List<DtoService> retrieveServices(ClientDataRequest request, String hostName) {
    ServiceClient serviceClient = new ServiceClient(getRestEndPoint(request));
    return serviceClient.list(hostName);
}
```

The DtoHost and DtoService objects are plain old java objects (serializable and safe for the java session) containing all attributes of a host or service. And here is the services portlet:



Services for Host: Amalthea

Return to List Hosts

| Service Name | Service Status | Last Check Time | Last Service Output |
|---|---|---|---|
| syn.vm.mem.guestToConfigMemSize.used | UNKNOWN | Wed Sep 23 15:00:25 PDT 2015 | Service has been suspended by end user suspending owning hypervisor |
| syn.vm.mem.compressedToConfigMemSize.used | UNKNOWN | Wed Sep 23 15:00:25 PDT 2015 | Service has been suspended by end user suspending owning hypervisor |
| syn.vm.cpu.cpuToMax.used | UNKNOWN | Wed Sep 23 15:00:25 PDT 2015 | Service has been suspended by end user suspending owning hypervisor |
| syn.vm.mem.balloonToConfigMemSize.used | UNKNOWN | Wed Sep 23 15:00:25 PDT 2015 | Service has been suspended by end user suspending owning hypervisor |
| syn.vm.mem.swappedToConfigMemSize.used | UNKNOWN | Wed Sep 23 15:00:25 PDT 2015 | Service has been suspended by end user suspending owning hypervisor |
| syn.vm.mem.sharedToConfigMemSize.used | UNKNOWN | Wed Sep 23 15:00:25 PDT 2015 | Service has been suspended by end user suspending owning hypervisor |

### 2.4.7 Preferences

The Portlet API provides a standard way of storing portlet data, known as preferences. These preferences are persistent and can be stored on a per user basis. Going back to doView method, you can see where we retrieve a single preference named "rows". This preference holds the number of rows to display on a page. Although this simple example does not make use of the rows preference, this gives you an idea of how you can customize the view of a portlet on a per user basis:

```
String rows = request.getPreferences().getValue(ROWS_PREFERENCE, "20");
request.setAttribute(ROWS_PREFERENCE, rows);
```

Allowed preferences for a portlet are defined in the portlet.xml descriptor. In fact, all portlets much be defined in the portlet.xml deployment descriptor.

### 2.4.8 Portlet Deployment Descriptor

The portlet.xml file, found in src/main/webapp/WEB-INF/portlet.xml, holds definitions of all portlets in this application. It tells the portal about the portlets being deployed in this application. Notice that there is a section for preferences, where we define the rows preference default value. This default value is can be overridden per user.

```
   <portlet>
        <description>Groundwork Monitor Hosts Portlet Tutorial</description>
        <portlet-name>HostsTutorialPortlet</portlet-name>
        <display-name>Groundwork Hosts Tutorial</display-name>
        <portlet-class>com.groundwork.dashboard.portlets.HostsTutorialPortlet</portlet-class>

        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>VIEW</portlet-mode>
        </supports>
        <supported-locale>en</supported-locale>
        <portlet-info>
            <title>Hosts Tutorial</title>
            <short-title>Hosts</short-title>
            <keywords>dashboard</keywords>
        </portlet-info>
        <portlet-preferences>
            <preference>
                <name>rows</name>
                <value>10</value>
            </preference>
        </portlet-preferences>
    </portlet>
```

# 3.0 Developing Custom Portlets with JavaScript

In this section, we will develop a second portlet with JavaScript. This portlet makes Rest API calls directly from JavaScript in your browser to the Groundwork backend. Like the example in chapter 4, we will make use of APIs that retrieve hosts and services and display them in a JQuery table widget. Each host row in the table will have a link to the services for the given host. When the user clicks on that link, all services listed for the select host.

In this lesson, we will work with one Java class and JSP files:

- com.groundwork.dashboard.portlets.HostsTutorialAjaxPortlet.java
- hosts-jquery-tutorial.jsp

Additionally, we will be using:

- The JQuery JavaScript library to facilitate AJAX calls to the Collage REST API and retrieve monitoring data from the GroundWork server
- The JQuery DataTables plugin for displaying columnar tables of hosts and services retrieved over AJAX
- The JQuery Cookies plugin to help with storing authentication token cookies in the browser

## 3.1 doView

In the first portlet example, we are calling the GroundWork Java REST API to get a list of all hosts in the system. In this example, we will receive the list of hosts directly from JavaScript. However, the doView method does still handle authentication and dispatching to the JSP:

```
    @Override
    @Override
    public void doView(RenderRequest request, RenderResponse response) throws PortletException,
IOException {
        response.setContentType("text/html");
        if (authenticate(request, response)) {
            PortletRequestDispatcher dispatcher =
getPortletContext().getRequestDispatcher(VIEW_HOSTS);
            dispatcher.include(request, response);
        }
    }
```

## 3.2 Authentication

When working with a JavaScript portlet, the server side still performs the authentication in a provided base class (DashboardPortlet.java) during

the doView request. The credentials are retrieved from /usr/local/groundwork/config/ws_client.properties:

```
if (authenticate(request, response))
```

**ws_client.properties**

```
webservices_user=RESTAPIACCESS
webservices_password=7UZZVvnLbuRNk12Yk5H33zeYdWQpnA7j9shir7QfJgwh
credentials.encryption.enabled=true
```

If you fail to authenticate, the base class will display a message to the portlet output. You should not render your portlet if authentication fails.

If authentication is successful, the base class DashboardPortlet sets a cookie in the browser holding a valid Groundwork token.
This token is good for 8 hours. This cookie is named FoundationToken. When making requests to the Groundwork server with JavaScript, you will need to pass in the token by retrieving it from a cookie, for example:

```
     var foundationToken = $JQ.cookie("FoundationToken");
```

And then when making your Ajax request to retrieve hosts:

```
   $JQ.ajax({
      url: hostUrl,
      dataType: "json",
      headers: {
       'GWOS-API-TOKEN': foundationToken,
       'GWOS-APP-NAME' : 'monitor-dashboard'
      },
...
```

## 3.3 Provide Headers

The Java Portlet API defines an interface for providing header elements during the 'render phase' of an HTTP request. This is a very useful concept,
allowing portlets to contribute and then merge scripts and CSS into the head area of the HTML content.
In our portlet, we need to contribute one CSS file for the Data Tables, as well as the JQuery main script, and 3 JQuery plugins:

- JQuery Data Tables plugin
- JQuery Cookies plugin
- JQuery No Conflict plugin

```
    protected void provideHeaders(RenderRequest request, RenderResponse response) {
        addStyleLink(response, "https://cdn.datatables.net/1.10.6/css/jquery.dataTables.css",
"gwforge-datatable");
addJavaScript(response, request.getContextPath() + "/app/scripts/jquery.min.js", "gwforge-jquery");
        addJavaScript(response, request.getContextPath() + "/app/scripts/jquery-noconflict.js",
"gwforge-jquery-nc");
        addJavaScript(response, request.getContextPath() + "/app/scripts/jquery.dataTables.js",
"gwforge-jquery-dt");
        addJavaScript(response, request.getContextPath() + "/app/scripts/jquery.cookie.min.js",
"gwforge-jquery-cookie");
    }
```

## 3.4 JQuery No Conflict

The Groundwork portal uses an older version of JQuery. In order for us to use the latest JQuery version, we need to namespace our version of JQuery to another version. This is accomplished with the JQuery No Conflict plugin. You will see throughout the JavaScript examples that we can't use the typical $ for JQuery functions, instead we use $JQ, for example:

```
$JQ("#servicesBlock").show();
```

### 3.5 Building the Hosts Data Table

Notice that this portlet comes with several cool features built into the JQuery Data Table plugin including:

1. Search
2. Sortable Columns
3. Pagination

Lets see how this all works. Open up hosts-jquery-tutorial.jsp and look at the ready function:

```
    $JQ(document).ready(function() {
        refreshHosts();
    });
```

JQuery's ready() function is the entry point into our JavaScript portlet. This JQuery callback is called when your page is loaded and ready, and is a good place to retrieve all the hosts from the Groundwork server. First, we leverage the Portal's Preference API to retrieve per user-customized settings. The
Portlet API provides some JSP tags to encode a request to the portal to retrieve the preferences for a current user and current portlet:

```
<portlet:resourceURL var="readPrefs" id="readPrefs" escapeXml="false" />
```

The tag above creates a Portlet API Resource URL. This URL can be used to make Ajax calls directed at a portlet. Portlets can serve resources with the
standard serveResource API. In this case we are simply retrieving all the preferences and returning them in JSON format using a Jackson object mapper:

```
  @Override
    public void serveResource(ResourceRequest request, ResourceResponse response) throws
PortletException, IOException {
        String resourceID = request.getResourceID();
        if (resourceID == null) {
            response.addProperty(ResourceResponse.HTTP_STATUS_CODE, "400");
            response.getWriter().println(createError(400, "invalid resource id"));
            return;
        }
        if (resourceID.equals("writePrefs")) {
            StringWriter writer = new StringWriter();
            drain(request.getReader(), writer);
            String json = writer.toString();
            ObjectMapper writeMapper = new ObjectMapper();
            TutorialPrefs update = writeMapper.readValue(json, TutorialPrefs.class);
            request.getPreferences().setValue(PREFS_REFRESH_SECONDS , Integer
.toString(update.getRows()));
            request.getPreferences().store();
        }
        TutorialPrefs prefs = new TutorialPrefs();
        prefs.setRows(Integer.parseInt(request.getPreferences().getValue(PREFS_ROWS, "10")));
        ObjectMapper objectMapper = new ObjectMapper();
        objectMapper.configure(SerializationFeature.INDENT_OUTPUT, true);
        StringWriter writer = new StringWriter();
        objectMapper.writeValue(writer, prefs);
        response.getWriter().println(writer);
    }
```

Here is how that call is made in the JavaScript refreshHosts() function. Note that we have to encode the URL being passed:

```
    function refreshHosts() {
        var prefsEndPoint = '<%=renderResponse.encodeURL(readPrefs.toString())%>';
        $JQ.ajax({
            url: prefsEndPoint,
            dataType: "json",
            success: function (prefs) {
                buildHostTable(prefs);
            },
            error: function (e) {
                console.log("Error Retrieving Prefs " + e.status + ", " + e.statusText);
            }
        });
    };
```

On success, we call another JavaScript function to build the Host table using the 'number of rows' preference retrieved in the first request:

```
function buildHostTable(prefs) {
        $JQ('#hostsTable').dataTable( {
            "pageLength": prefs.rows,
            "columns": [
                { "data": "hostName" },
                { "data": "monitorStatus" },
                { "data": "appType" },
                { "data": "properties.LastPluginOutput"}
            ],
            "ajax": function (data, callback, settings) {
                var foundationToken = $JQ.cookie("FoundationToken");
                var foundationRestService = $JQ.cookie("FoundationRestService");
                var hostUrl = window.location.origin + '/api/hosts';
                $JQ.ajax({
                    url: hostUrl,
                    dataType: "json",
                    headers: {
                        'GWOS-API-TOKEN': foundationToken,
                        'GWOS-APP-NAME' : 'monitor-dashboard'
                    },
                    type: "GET",
                    success: function( data ) {
                        console.log("success " + data);
                        for ( var ix=0, ixlen=data.hosts.length; ix<ixlen ; ix++ ) {
                            data.hosts[ix].hostName = "<a href='javascript:showService(\"" +
data.hosts[ix].hostName + "\")'>"
                            + data.hosts[ix].hostName + "</a>";
                        }
                        var wrapped = {};
                        wrapped.data = data.hosts;
                        callback(wrapped);
                    },
                    error: function(e) {
                        console.log("Error retrieving Hosts " + e.status + ", " + e.statusText);
                    }
                });
            }
        } );
    }
```

Without spending too much time on the details of the JQuery Data Table Plugin, lets cover a few of the important areas of using Data Table.

The table is a simple HTML table:

```
<table id="hostsTable" width="100%" >
    <thead>
    <tr>
        <th>Host Name</th>
        <th>Monitor Status</th>
        <th>App Type</th>
        <th>Last Plugin Output</th>
    </tr>
    </thead>
</table>
```

We use a simple JQuery selector to select it, and then inject a Data Table into that table:

```
$JQ('#hostsTable').dataTable( {
```

We then define the number of rows (from the portlet preferences) and data columns. Its important that data columns match the JSON attribute names returned
by the Groundwork API:

```
            "pageLength": prefs.rows,
            "columns": [
                { "data": "hostName" },
                { "data": "monitorStatus" },
                { "data": "appType" },
                { "data": "properties.LastPluginOutput"}
            ],
```

And then we make the AJAX call by providing the Data Table with our URL and authentication token:

```
            "ajax": function (data, callback, settings) {
                var foundationToken = $JQ.cookie("FoundationToken");
                var hostUrl = window.location.origin + '/api/hosts';
                $JQ.ajax({
                    url: hostUrl,
                    dataType: "json",
                    headers: {
                        'GWOS-API-TOKEN': foundationToken,
                        'GWOS-APP-NAME' : 'monitor-dashboard'
                    },
                    type: "GET",
```

Note that the API call is made using JQuery's ajax function. To build the URL to make the service call, we use the browser's window location to point back to the Groundwork server origin. We append the api name: /api/hosts to that URL. Note that we also pass in the headers including the GWOS-API-TOKEN, which was retrieved via the token cookie set by the portlet.

This particular service retrieves all monitored hosts. There are many more apis available including querying and CRUD operations. The full documentation for the Groundwork Hosts API can be found here:

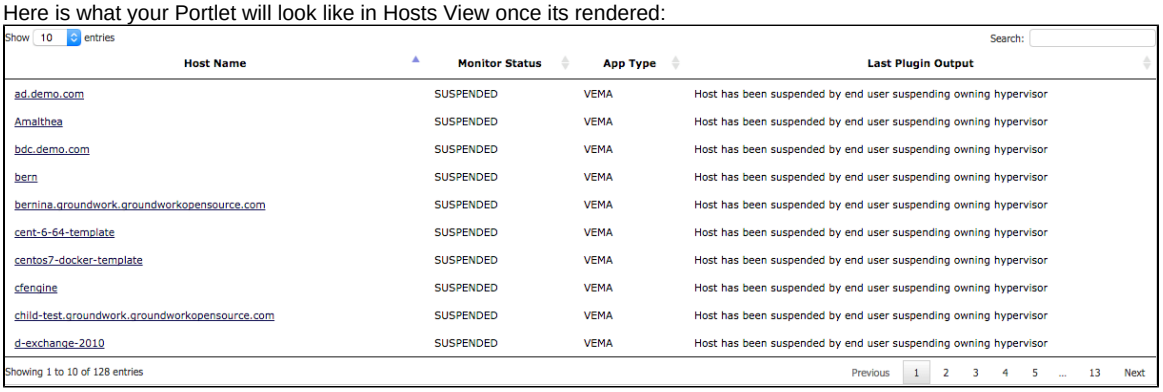
Groundwork Hosts API Documentation

Finally, we handle the AJAX request on a success function:

```
                    success: function( data ) {
                        console.log("success " + data);
                        for ( var ix=0, ixlen=data.hosts.length; ix<ixlen ; ix++ ) {
                            data.hosts[ix].hostName = "<a href='javascript:showService(\\"" +
data.hosts[ix].hostName + "\\")'>"
                            + data.hosts[ix].hostName + "</a>";
                        }
                        var wrapped = {};
                        wrapped.data = data.hosts;
                        callback(wrapped);
                    },
```

Note that we are building an HREF link on the hostName that will link you to a second table showing all Groundwork services for the given host.

Here is what your Portlet will look like in Hosts View once its rendered:



Show 10 entries — Search:

| Host Name | Monitor Status | App Type | Last Plugin Output |
|---|---|---|---|
| ad.demo.com | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| Amalthea | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| bdc.demo.com | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| bern | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| bernina.groundwork.groundworkopensource.com | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| cent-6-64-template | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| centos7-docker-template | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| cfengine | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| child-test.groundwork.groundworkopensource.com | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |
| d-exchange-2010 | SUSPENDED | VEMA | Host has been suspended by end user suspending owning hypervisor |

Showing 1 to 10 of 128 entries — Previous 1 2 3 4 5 ... 13 Next

## 3.6 Building the Services Data Table

Each host row in the table is hyperlinked to another table containing all the services for a that host. Lets take a look at the HTML for the service page:

```
<div id='servicesBlock' style="display: none">
    <br/>
    <div id="servicesInfo" class='hostNameFormat'> Services for Host: <span class='hostNameFormat'
id='hostNameField'/></div>
    <div id="returnToList"><a href="javascript:showHosts()"> Return to List Hosts</a></div>
    <br/>

    <table id="servicesTable" width="100%" >
        <thead>
        <tr>
            <th>Service Name</th>
            <th>Service Status</th>
            <th>Last Check Time</th>
            <th>Last Plugin Output</th>
        </tr>
        </thead>
    </table>
</div>
```

Note that the first div, 'servicesBlock' has a style 'display: none'. This means that the service table is not displayed by default.
When a hostname is clicked on in the hosts table, a Javascript function is called which hides the hosts table and shows the service table:

```
    function showService(hostName) {
        $JQ("#hostsBlock").hide();
        $JQ("#servicesBlock").show();
        ...
```

The showService function then injects a data table into the table "servicesTable". Note that the columns are matched to the JSON payload returned by the Groundwork Service API. The columns data names description, monitorStatus, lastCheckTime, and properties.LastPluginOutput are all fields in the returned AJAX payload:

```
$JQ('#servicesTable').dataTable( {
        "columns": [
            { "data": "description" },
            { "data": "monitorStatus" },
            { "data": "lastCheckTime" },
            { "data": "properties.LastPluginOutput"}
        ],
```

Here we make the Service API call to retrieve all services for the selected host:

```
        "ajax": function (data, callback, settings) {
            var foundationToken = $JQ.cookie("FoundationToken");
            var foundationRestService = $JQ.cookie("FoundationRestService");
            //var serviceUrl = foundationRestService + "/services?hostName=" + hostName;
            var serviceUrl = window.location.origin + '/api/services?hostName=' + hostName;
            $JQ.ajax({
                url: serviceUrl,
                dataType: "json",
                headers: {
                    'GWOS-API-TOKEN': foundationToken,
                    'GWOS-APP-NAME' : 'monitor-dashboard'
                },
```

Note how the hostName is used to build out the API URL:

```
var serviceUrl = window.location.origin + '/api/services?hostName=' + hostName;
```

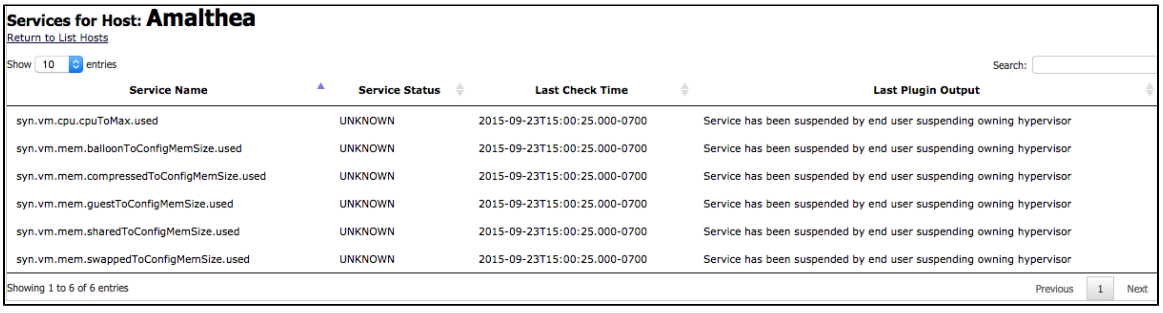
On success, the AJAX success callback is called where we take the JSON data and prepare it for JQuery Data Table consumption. This involves taking the response, which is a JSON array, wrapping it in a JSON object named 'data', and calling the Data Table callback, which handles the details of matching rows and columns in the JSON payload into the Data table widget.

```
    success: function( data ) {
      var wrapped = {};
      wrapped.data = data.services;
      callback(wrapped);
      },
```

For more information on using the Groundwork Services API for queries and CRUD operations, reference the documentation here:

Groundwork Service API Documentation

Here is what the services page will look like. Note that you can return to the Hosts Lists by clicking on a hyperlink.



# 4.0 Deployment

Next, we will deploy our war file that we built in the Build section to the GroundWork Portal.

## 4.1 Deploy the war file

Deploying the war file requires the commands:

```
cp target/gw-portlet-examples.war /usr/local/groundwork/jpp/standalone/deployments/
touch /usr/local/groundwork/jpp/standalone/deployments/gw-portlet-examples.war.dodeploy
```

To ensure your war file was deployed, there should be a marker file denoted that the deployment went safely:

```
ls /usr/local/groundwork/jpp/standalone/deployments/gw-portlet-examples.war.deployed
```

If it failed to deploy, you will see a failed marker file

```
ls /usr/local/groundwork/jpp/standalone/deployments/gw-portlet-examples.war.failed
```

## 4.2 Un-deploy the war file

To un-deploy the war file you will need to remove any instances from the portal. Using the root login run the following commands:

This will log out all portal users.

```
rm /usr/local/groundwork/jpp/standalone/deployments/gw-portlet-examples.war
```

```
/etc/init.d/groundwork restart gwservices
```

## 4.3 Adding the Portlet to the Portal Site Menu

Watch the video below to learn how to deploy a portlet to the GroundWork Portal and add it to the site menu. In this video, we will:

1. Update the Portal Registry
2. Create a new menu option on our site under Dashboards menu called "Tutorial"
3. Create a new page and attach it to the menu
4. Place the HostsTutorialPortlet on the page

DeployGroundworkPortlet.mp4