How-to: Consume Enterprise Services with Ruby on Rails – Part I

Applies to:

Ruby on Rails and Enterprise Services

Summary

This paper describes how to create a simple Ruby on Rails application which consumes an Enterprise Service hosted on SDN's ES Workplace. For this the standard Ruby library soap4r is used to call an example Enterprise Service on the ES-Workplace and Ruby on Rails is used to build a web application on top of it.

The paper also cares about security issues. Unfortunately the US-Workplace Enterprise Services are only secured with the HTTP Basic Auth. Due to that fact Part I it shows, how the standard authentication with HTTP Basic Auth works. But for the reason the standard security configuration is not the best way, part II shows how to configure more advanced security settings for a service and how to consume it.

Author: Tobias Braner

Company: SAP AG

Created on: 14 November 2007

Author Bio

Tobias Braner is a student from SAP for business information technologies. He began his studies in 2005 and is currently in the 5th semester. His interests for Ruby arose when he was searching for an alternative for PHP. Since then he has been an enthusiastic Ruby hacker.

Table of Contents

1 Introduction

1.1 Preface

I don't have to mention the impact of scripting languages on web application programming. I'm sure if you are reading this guide you are well aware of this fact. What we are going to do is to consume an Enterprise Service using a Ruby on Rails web application.

This article will cover all steps needed to get the application running. What I won't cover is the basic Ruby syntax. I assume that you are familiar to Ruby or other scripting languages.

Also this how-to cares about security issues. The standard authentication mechanism for Enterprise Services is the HTTP Basic Auth if nothing else is configured. The services on the ES-Workplace we are going to consume are also secured with this mechanism. To access them you need to authenticate with a username and a password you can get on the ES-Workplace. To get your user refer to section 1.4. In this part of the document I will describe how to authenticate with this type of authentication, but for the reason that this is not the best way I will show how to authenticate using the wsse:Username-Token which is specified by OASIS (http://www.oasis-open.org/). Note that the ES-Workplace doesn't use a HTTPS connection. In a real scenario I highly recommend to use a SSL (Secure Socket Layer) secured connection.

1.2 Installation

There is a pretty good guide on how to install Ruby on Rails (RoR) on http://www.rubyonrails.com/down. Follow this guide to install Ruby, Rubygems and Rails. I am using the following versions:

- Ruby 1.8.6
- Rails 1.2.3
- Rubygems 0.9.4
- soap4r 1.5.8

As we want to access Enterprise Services we need a SOAP library. "soap4r" is the one I chose for this how-to. It is a Ruby standard library, but the release cycle of soap4r is faster than the ruby core release cycle. So you should also install the newest version of soap4r. You can do this via command shell:

```
gem install soap4r
```

If you are using a proxy you should set the http_proxy environment variable before:

```
set http_proxy=http://proxy:8080
```

1.3 Creating a new Rails project

Using an IDE such as Netbeans 6.0 beta or Eclipse with the RadRails plugin we can create a new project using the "New Project" assistant. But to show that RoR does not necessarily need an IDE you can create a new project using the command shell. We go to the folder you want to add the project and execute the following command:

```
rails MaterialManager
```

The string *MaterialManager* is a parameter for the Rails command and stands for the name of the project. After execution a new directory with the name *MaterialManager* is created. In this directory we can have a look at the Rails project structure.

To use the newest version of *soap4r* in the Rails application and not the built in library, we have to add a new line to the *boot.rb,* which we can find in the following directory:

```
MaterialManager\config\boot.rb
```

After opening the file we should to add the following code lines right after the *require 'rubygems'* line:

```
require 'rubygems'   # should be there, search for it
gem 'soap4r'         # add this line
```

Now Rails knows that it should use the soap4r module we downloaded. If you are using a proxy we can also add the following line:

```
ENV['http_proxy'] = http://proxy:8080
```

Replace the *http://proxy:*8080 with your real proxy address. Now we can access web services through a proxy. By the way: you can also modify the *http_proxy* environment variable on runtime. Note that the line has the same effect as the command line proxy command:

```
set http_proxy=http://proxy:8080
```

After changes in the file *boot.rb* the server has always to be restarted, because the script file is always executed at server startup (as the file name “boot” says).

1.4 Introducing the Enterprise Service

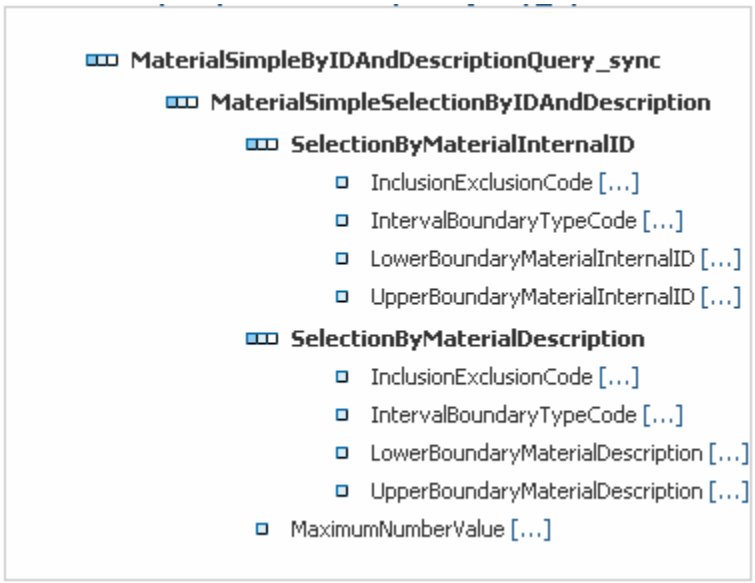
In this guide we will consume an Enterprise Service. You can get a user for the ES Workplace on this website (requires D-, I-, S-User):

https://www.sdn.sap.com/irj/sdn/explore-es

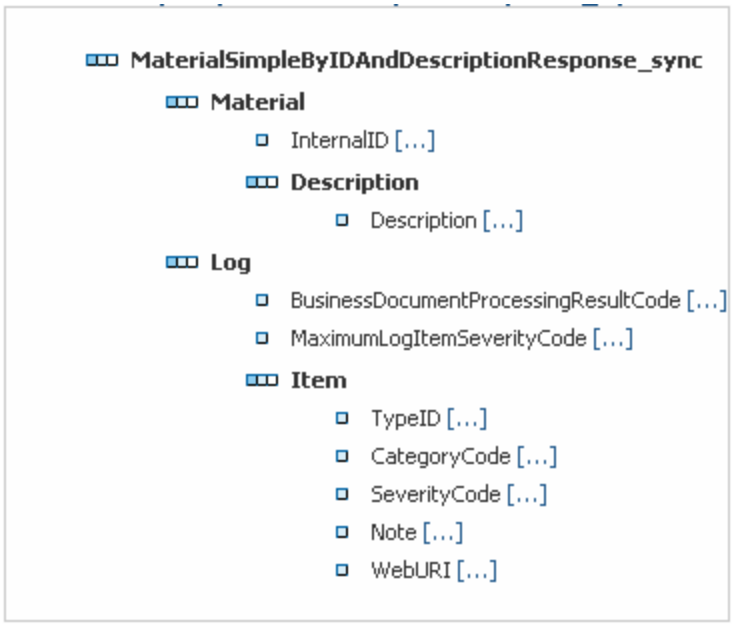
The ES Workplace provides a nice web frontend to browser the Enterprise Services and it also provides Enterprise Service descriptions for the services we want to use. After registration we can follow the hyperlink on top of the page to explore all available Enterprise Services. Right under the Enterprise Service index header we can use the link *Enterprise Services by Process Components*. On the following site we select the *Product Data Management* and right now we follow the link *Query Material In* and then click *Find Material by ID and Description*. At this point we can access the *WSDL* file for the Enterprise Service using the link *Back-end WSDL (XML) for ECC_MATERIALBYIDDESCR001QR*. Also interesting is the detailed field description you can find by clicking *Click here to get detailed field description*.

Here you can see screenshots from the ES-Workplace which show the input and output message structure.

Input message:



Output message:



1.5 Generating the web service client

In this step we are generating a web service client. Did I say “generating”? That’s right, the Ruby library with its tool *wsdl2ruby* generates the web service stubs for us. Rails uses the common MVC-Concept. As I figured out our web service client is the data source and therefore a model. That’s the reason why we use the command shell and change the directory to:

```
MaterialManager\app\models\
```

If the *WSDL* you are trying to access is not protected through HTTP Basic Auth or other authentication mechanisms you can execute the following line in the command shell to generate the client:

```
wsdl2ruby --wsdl url_of_the_wsdl --type client --force
```

Another possibility is to view the *WSDL* in your browser and save it to the model folder. I did it this way and saved the *WSDL* to

```
MaterialManager\app\models\material.wsdl
```

After that I used wsdl2ruby:

```
wsdl2ruby --wsdl material.wsdl --classdef material --type client --force
```

At this point I’d like to explain what *wsdl2ruby* does. It parses the *WSDL* file and generates some classes for the input/output parameters, the mapping for the data types and a driver we finally use to consume the Enterprise Service. That’s a bunch of code you also could also manually hack in using the good old keyboard. But this way it’s much faster. For more advanced options use:

```
wsdl2ruby --help
```

The generator created following files for us:

1. material.rb
2. MaterialSimpleByIDAndDescriptionQueryResponse_InServiceClient.rb
3. materialDriver.rb
4. materialMappingRegistry.rb

I advice to rename the files to the following:

1. material_classes.rb
2. material.rb
3. material_driver.rb
4. material_mapping_registry.rb

The file number one contains all the classes, so I called it *material_classes.rb*. File number two contains the final call to the Enterprise Service and therefore it's our model class. So I just called it *material.rb*. Further, I removed the CamelCase spelling and added underscores as common in Ruby on Rails. After the renaming procedure we also have to open all the files and update the *require* statements.

The require-statements should look like that:

material_classes.rb

```
Nothing to update
```

material.rb

```
require 'material_driver.rb'
```

material_driver.rb

```
require 'material_classes.rb'
require 'material_mapping_registry.rb'
require 'soap/rpc/driver'
```

material_mapping_registry.rb

```
require 'material_classes.rb'
require 'soap/mapping'
```

1.6 Altering the generated stub

The basis for the two following parts is this skeletal structure:

material.rb

```
require 'material_driver.rb'

class Material
  # define a method to get all materials
  def self.get_material_list(username, password, req)
  # create the driver
  obj =  MaterialSimpleByIDAndDescriptionQueryResponse_In.new()
  # authentication (we will add this code later)
  […]
  retVal = obj.materialSimpleByIDAndDescriptionQueryResponse_In(req)
  end
end
```

2 Accessing a HTTP Basic Auth secured Enterprise Service

The Enterprise Services we want to consume on the ES-Workplace are secured through the HTTP Basic Auth and in order to get a result from a request we have to authenticate ourselves. We will do that in the following steps:

2.1 Adding HTTP Basic Auth to the model

To alter the existing model open the *material.rb* and alter its code:

material.rb

```
require 'material_driver.rb'
class Material
  # define a method to get all materials
  def self.get_material_list(username, password, req)
  # create the driver
  obj =  MaterialSimpleByIDAndDescriptionQueryResponse_In.new()
  # add basic authentication credentials
  obj.options["protocol.http.basic_auth"] << [obj.endpoint_url, username,
  password]
  retVal = obj.materialSimpleByIDAndDescriptionQueryResponse_In(req)
  end
end
```

Here we add an array containing the endpoint URL, the username and the password to the options of the driver object. The *soap4r* library now automatically adds the user credentials to the request.

2.2 Creating the material controller

To create the material controller we can use the Rails generator. Change in the project's root directory:

```
MaterialManager\
```

To generate the controller you can type in and execute the following line:

```
ruby script/generate Controller Material show_material_list
```

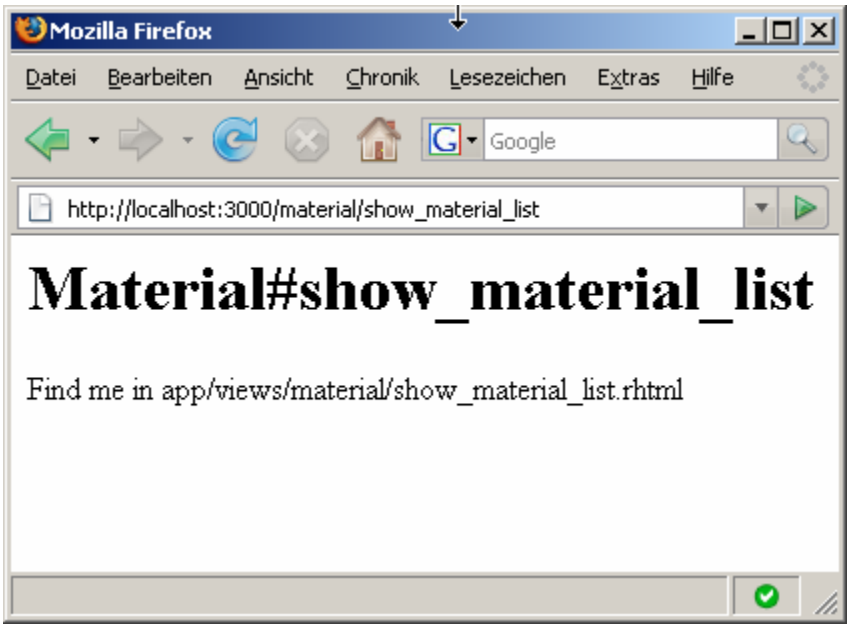
This command generates the *MaterialController* and a *show_material_list* action in it. As for the reason an action is representing a view, we have also created a view with the name *show_material_list.* Let's view our results, change to the project root and type in:

```
ruby script/server
```

This starts the built in WEBrick server. You can now open your favorite browser and view the following URL:

http://localhost:3000/material/show_material_list

The following screenshot shows how the view should look like:



Now that we have a view, a controller and a model, we should display the model's data. To do so we will edit the *material_controller.rb*:

material_controller.rb

```
class MaterialController < ApplicationController
  def show_material_list
        @materials =
        Material.get_material_list(”yourUser”,”yourPassword”,req).materia
        l
  end
end
```

In the *show_material_list* action we now make a call to the *Material* model and call its method *get_material_list*. We save the return value in the *@material* instance variable. For now you should replace *yourUser* and *yourPassword* with your credentials for the ES Workplace (later we will remove the hardcoded credentials). You might have noticed that we passed the *req* variable to the model call, but it is not defined yet. This parameter should contain the complex input parameters for the service call. We are going to initialize these parameters in a separate method in the material controller (note that this action should be private and that you must require the file *material_classes.rb*):

material_controller.rb

```
require 'material_classes.rb'


private


def create_query
  # create a new request object
  req = MaterialSimpleByIDAndDescriptionQueryMessage_sync.new
  # create the id and description object
  req.materialSimpleSelectionByIDAndDescription = id_and_desc =
  MaterialSimpleByIDAndDescriptionQueryMessage_sync::MaterialSimpleSelect
  ionByIDAndDescription.new
  # create a new object for the material internal id
```

```
    id_and_desc.selectionByMaterialInternalID << id =
    MaterialSimpleByIDAndDescriptionQueryMessage_sync::MaterialSimpleSelect
    ionByIDAndDescription::SelectionByMaterialInternalID.new

    # create a new object for the material description

    MaterialSimpleByIDAndDescriptionQueryMessage_sync::MaterialSimpleSelect
    ionByIDAndDescription::SelectionByMaterialDescription.new

    # fill this objects with values

    id.inclusionExclusionCode = 'I'

    id.intervalBoundaryTypeCode = '3'

    id.lowerBoundaryMaterialInternalID = id1 =

    ProductInternalID.new('MDE-001')

    id.upperBoundaryMaterialInternalID = id2 =

    ProductInternalID.new('MDE-999')

    id1.xmlattr_schemeID = 'schemeID1'

    id2.xmlattr_schemeID = 'schemeID2'

    id_and_desc.maximumNumberValue = 0

    return req

  end
```

And now that we can build an example request object we can finally call the model:

material_controller.rb

```
require 'material_classes.rb'

class MaterialController < ApplicationController


  def show_material_list

          req = create_query

          @materials =
Material.get_material_list(”yourUser””,”yourPassword”,

                req).material

  end


  private

  def create_query

    # create a new request object

    req = MaterialSimpleByIDAndDescriptionQueryMessage_sync.new

    # create the id and description object

    req.materialSimpleSelectionByIDAndDescription = id_and_desc =
    MaterialSimpleByIDAndDescriptionQueryMessage_sync::MaterialSimpleSelect
    ionByIDAndDescription.new


    # create a new object for the material internal id

    id_and_desc.selectionByMaterialInternalID << id =
    MaterialSimpleByIDAndDescriptionQueryMessage_sync::MaterialSimpleSelect
    ionByIDAndDescription::SelectionByMaterialInternalID.new

    # create a new object for the material description
```

```
      MaterialSimpleByIDAndDescriptionQueryMessage_sync::MaterialSimpleSelect
      ionByIDAndDescription::SelectionByMaterialDescription.new
      # fill this objects with values
      id.inclusionExclusionCode = 'I'
      id.intervalBoundaryTypeCode = '3'
      id.lowerBoundaryMaterialInternalID = id1 =
      ProductInternalID.new('MDE-001')
      id.upperBoundaryMaterialInternalID = id2 =
      ProductInternalID.new('MDE-999')
      id1.xmlattr_schemeID = 'schemeID1'
      id2.xmlattr_schemeID = 'schemeID2'
      id_and_desc.maximumNumberValue = 0
      return req
    end
  end
```

2.3 Altering the view

You can find the view in the following location:

```
MaterialManager/app/views/material/show_material_list.rhtml
```

Open this file and enter the following code:

show_material_list.rhtml

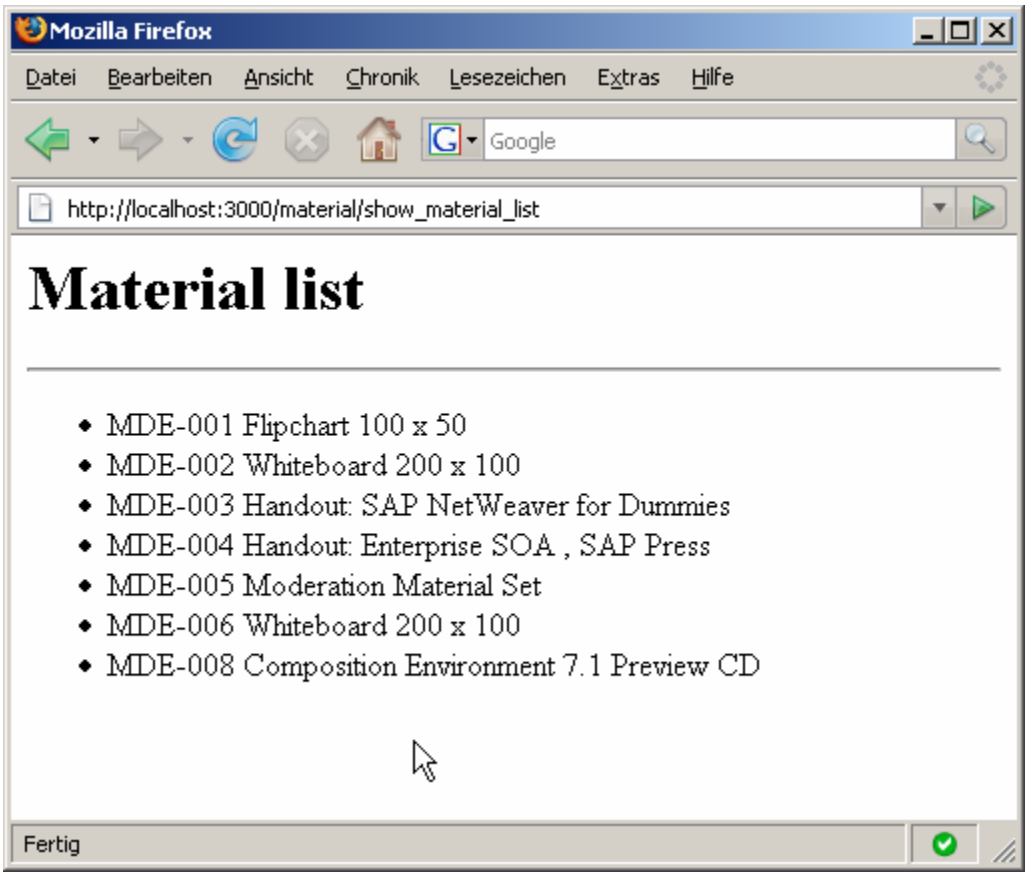
```
<h1>Material list</h1>
<hr/>
<ul>
<% for material in @materials do %>
<%= "<li>#{material.internalID}
#{material.description.description}</li>" %>
<% end %>
</ul>
```

The @materials variable is a list of all materials. We set this variable in the *MaterialController's show_material_list* action. Using the *for* iterator we can easily iterate trough this list of material objects and create a HTML list item for any material and display the ID and the description.

Open your browser again and refresh the *show_material_view* or just load the following url:

http://localhost:3000/material/show_material_list

Here is a screenshot of how it should look like:



2.4 Creating the authentication mechanism

2.4.1 Creating the user controller

So far the username and password to access the web service are hard coded in the *material_controller.rb*. That's not a nice solution and what we are going to do is to create a form view where the user has to login with his username and password.

To do so we will create a user controller with the following command in the root directory of the project:

```
ruby script/generate Controller User show_logon_screen
```

This command creates a user controller and a view with the name show_logon_screen. To call the view use the following URL in your browser:

http://localhost:3000/user/show_logon_screen

There is nothing exciting to see so far. But now we will edit the *show_logon_screen.rb* in the *app/views/user/* folder. We replace the view's code with the code below:

show_logon_screen.rb

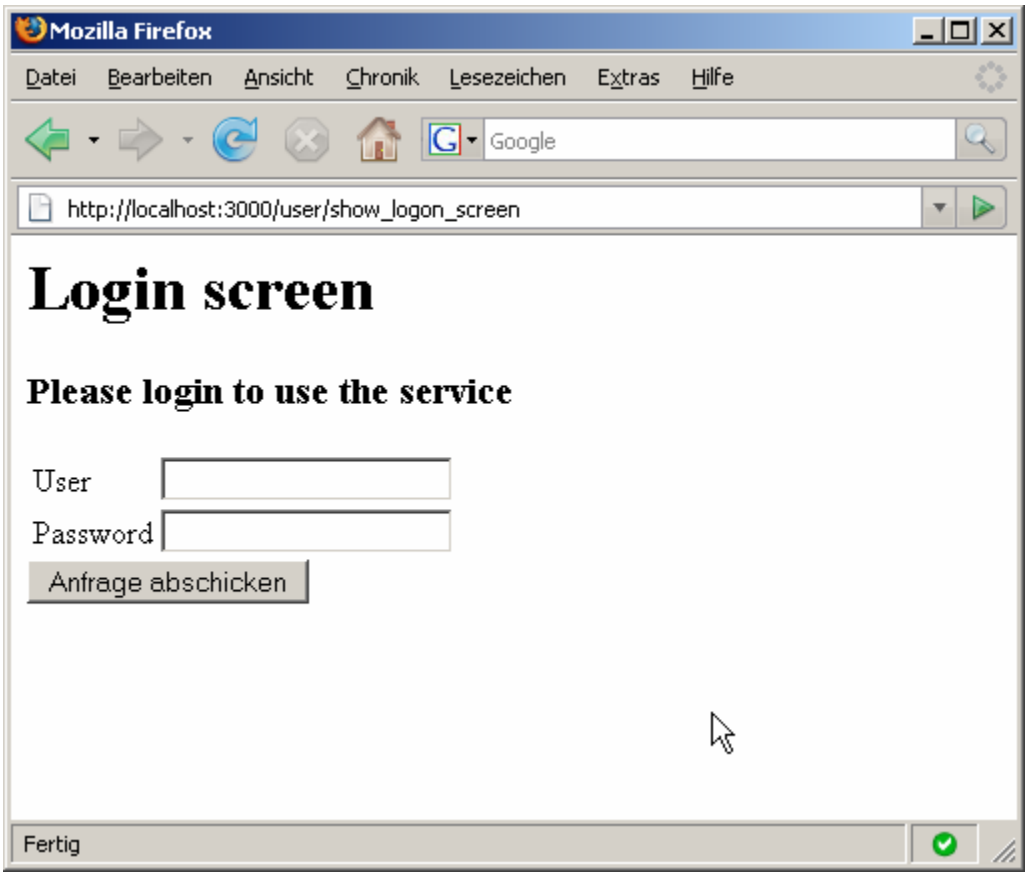
```
<h1>Login screen</h1>
<h3>Please login to use the service</h3>

<%= flash[:notice] %>

<form action="/user/do_login" method="post">
  <table>
    <tr>
      <td>User</td>
```

```
        <td><input type="text" name="uname"/></td>
      </tr>
      <tr>
        <td>Password</td>
          <td><input type="password" name="pw"/></td>
      </tr>
      </table>
      <input type="submit">
  </form>
```

If we refresh the browser we should be able to see the following window:



We basically create a form with two text fields, one for the username and one for the password. The first thing to mention about this code is the *<%= flash[:notice] %>* command. The flash variable is a hash which contains messages we want to show the user. In our case we want to display the message “Sorry, your username and password are wrong” if the login fails. Further you might have noticed the action attribute in the form tag. Our target action is “do_login”. This action does not exist. So we will create the action now.

We open the *user_controller.rb* in the *controllers* folder. The current code looks pretty simple:

user_controller.rb

```
  class UserController < ApplicationController
```

```
  def show_logon_screen
  end
end
```

We are going to add the *do_login* action here:

user_controller.rb

```
def  do_login
  # some code to test if the user is valid
end
```

We will leave the action's body empty for now.

2.4.2 Creating the User Model

You can create the user model using the following command in the command line:

```
ruby script/generate Model User
```

Open the generated model in

```
MaterialManager/app/models/user.rb
```

and alter its code to:

user.rb

```
class User

  def self.check_logon(username, password)

    begin
      # 1. call the url
      client = HTTPClient.new()
      url = "http://erp.esworkplace.sap.com:80/sap/bc/bsp/sap/system"
      result = client.get(url, [["sap-user",username],["sap-
            password",password]]).header
    end

    if result.response_status_code == 401
      return false
    else
      return true
    end

    rescue Exception => err
      return false

  end
end
```

The static method *check_logon* tries to create a new HTTP client and call the URL http://erp.esworkplace.sap.com:80/sap/bc/bsp/sap/system. This URL is an official logon page for BSP (Business Server Pages) Applications. It exists an all Systems under the URL *http(s)://<host>:<port>/sap/bc/bsp/sap/system*. We pass the username and password as parameters to the URL and if the credentials are not valid, we get a *401* as status code in the response. We will go on modifying the controller and the *do_login* action:

user_controller.rb

```
def do_login
    logon = User.check_logon(params[:uname], params[:pw])
    if logon
              # create session and redirect
              session[:username] = params[:uname]
              session[:password] = params[:pw]
              flash[:notice] = "Successfully logged on"
              redirect_to :controller => 'material', :action =>
'show_material_list'
    else
              # flash and redirect
              flash[:notice] = "Logon failed, wrong username or password"
              redirect_to :controller => 'user', :action =>
'show_logon_screen'

    end

  end
```

We call the *check_logon* method from the model and check the return value. If the credentials are valid we set the two session variables *username* and *password* and assign the credentials to them. Further we can access this session variables in all controllers using *session[:username]* and *session[:password]*. In the next step we will remove the hard coded username and password in the *material* controller.

2.4.3 Altering the MaterialController

We do not want anybody who has not logged on to our application to show the *show_material_list* view. So we add a filter:

material_controller.rb

```
    class MaterialController < ApplicationController

      before_filter :verify_credentials

      def show_material_list
            req = create_query
            @materials = Material.get_material_list(session[:username],
                  session[:password], req).material
      end
```

```
    private
    […]
    def verify_credentials
          if (session[:username].nil?) or (session[:password].nil?)
                # if a value is nil redirect
                redirect_to :controller => 'user',
                      :action => 'show_logon_screen'
          end
    end

end
```

As you can see we replaced the hard coded username and password with the session variables. This code helps us to prevent anybody who has not yet logged on to view the material list. The *before_filter* method takes a parameter which represents the name of the method called before the view is displayed. In this case the *verify_credentials* method is called. Note that this method should be in the private part of the class. The method checks, if the session variables we set above are *not nil*. If they are *nil*, the user will be redirected to the logon screen, else nothing happens and the view will be displayed.

2.5 Extensions

At this time you are able to call an Enterprise Service. But it would be much better to have a web interface to get the values of the input parameters for the service. I won't cover this step in this document because they are out of scope, but I hope you are going to try it out.

3 Summary

As you hopefully have experienced, it is not too hard to consume an Enterprise Service with a scripting language like Ruby. And using the framework Ruby on Rails it is easy to build a web application on top.

A critical problem is the security. Generally authentication for web services can be done by the with transport- or document security. In this how-to we used the HTTP Basic Auth for authentication which belongs to the transport security. But it would be better to use the document security for authentication, because the web service specification says nothing about the transport protocol. So HTTP is not necessarily needed for transport of the SOAP document and because of that fact document security is the way to go. In my next how-to I will describe how to configure an Enterprise Service for document security and how to access this service extending our current Rails application.

Copyright