How to…
Use the Application Integrator

ENTERPRISE PORTAL 6.0

PUBLIC

ASAP "How to…" Paper

Applicable Releases: EP 6.0 SP1/SP2

November 2003

1 Scenario

A customer wants to integrate (Web) applications very flexibly, for example passing parameters or providing credentials. These applications could be SAP applications or any application that comes with a Web (HTTP/HTTPS) frontend. Parameters could be static or retrieved dynamically.

2 Result

SAP Enterprise Portal 6.0 SP1/SP2 comes with a very flexible and powerful toolset (Application Integrator) that can be used to satisfy the above requirements.

The toolset includes the following portal applications:

- com.sap.portal.appintegrator
- com.sap.portal.appintegrator.sap

The Application Integrator provides an infrastructure for integrating remote (Web) applications into the portal. It allows you to create portal components that serve as a template for iViews representing the remote application. Several such portal components have already been implemented using the Application Integrator. In the below list the focus is on SAP applications:

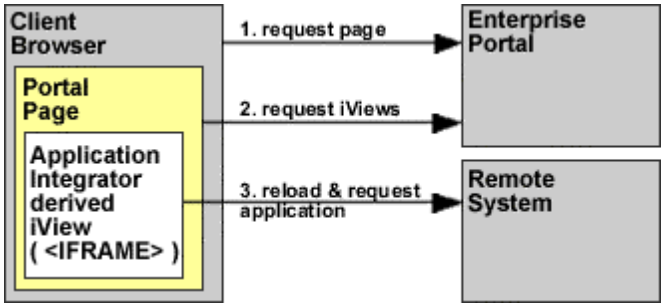
- Business Server Pages (BSP)
- BEx Web Applications (BW Report)
- Crystal Reports
- DNR Objects
- Internet Application Components (IAC)
- Workplace 2.x MiniApps
- SAP Transactions (R/3 Transactions)
- WebDynpro
- HTTP-Based Web Applications

This guide will focus on HTTP-based Web applications and on how you can pass parameters to these applications. It also describes how parameters can be retrieved dynamically.

For details about integrating the other options, see the EP 6.0 documentation (http://help.sap.com).

3 Functionality of an Application Integrator iView

An Application Integrator iView creates the required HTML content that ultimately loads the remote application it represents. This is done in three steps:



Remark: The loading procedure implies that it must be possible to reach all the content included in the portal using the Application Integrator from the client computer. The Application Integrator infrastructure is only used to create the HTML that loads the remote application. After the application has been loaded, the iView usually ceases to interact with the portal itself.

4 Integrating an HTTP Web Application

4.1 Component com.sap.portal.appintegrator.sap.Generic

The "Generic" component, which can be found in portal application "com.sap.portal.appintegrator.sap", is the preferred component for integrating any kind of HTTP-based Web applications in a very flexible manner. For example, parameters can be passed and SSO can be performed.

Remark: As already mentioned in the previous chapter, the Web application is called directly by the client (isolation level URL). Since the portal does not serve as content proxy, the Web application has to be accessible from the client computer, that is firewalls have to be opened for specific ports, ….

4.2 Iview/Component Parameters

- **ApplicationParameter**: Defines name-value pairs to be passed to the application, like `appParam1=value1&appParam2=value2`. Multiple name-value pairs have to be separated using the ampersand character (&).
- **DebugMode**: Setting this parameter to `true` returns a screen with debugging output before launching the application. `false` disables debugging mode.
- **Id**: Like parameter "ApplicationParameter", this parameter defines name-value pairs. It is used as a placeholder and can be useful for predefining some settings for delta links and iView templates.
- **LAF**: Defines the theme subset (e.g. `ur` (unified rendering), `controls`).

- **SSO2Template**: Defines a template that is used in SSO parameter <*Authentication*> if the system defines logon with SAP Logon Tickets (for details see below).
- **System**: Name of the system alias.
- **URLTemplate**: Defines a template that is processed by the template processor in order to generate the application url (for details see below).
- **UserMappingTemplate:** Defines a template that is used in SSO parameter <*Authentication*> if the system defines logon with user mapping (for details see below).
- **RequestMethod**: Overwrites the default behavior for retrieving the request method of the redirect (GET or POST). For SSO with user mapping, the default is "POST". "POST" is also used if the parameter list is longer than 1024 characters.
  Remark: **Currently (SP2 Patch1) this property has to be added manually to the deployment descriptor of the application. It is therefore necessary to modify the portal application and to upload the par file again!**

4.3 Template Processor

The template processor can parse a template (e.g. URLTemplate for the "Generic" component) and process its tag expressions. The template-processing step replaces the tag expressions with values stored in contexts, where the context names must match the tag expressions.

4.3.1 Definitions

- A **template** is a string containing plain text sections as well as tags,
  example: *This is a template with a simple <tag>.*
- A **tag** is a variable expression followed by an optional modifier sequence within angle brackets,
  example: *surround tags always with <angle.brackets>*
- A **tag expression** (or **variable expression**) is a non-empty set of variables separated by dots,
  example: <*tag.expression.with.multiple.variables*>
- A **context** is a data structure that stores name-value pairs like java.util.Hashtable. In contrast to hast tables, contexts can be nested.
- A **variable** is an identifier that has matching entries in the context. If a variable name is the same as a context entry name, the tag is replaced with the value of the context entry,
  example: <*context1.variable1*> is replaced with the value of variable1 in context1.
- A **modifier sequence** contains one or more modifiers seperated by blanks within square brackets. Modifiers are applied to the substitute of a processed tag expression in the order they appear,
  example: *This is a template with <modified.tag[URL_ENCODE UPPERCASE]>*
- A **substitute** of a tag expression is the result of a processed tag expression,
  example: <*User.Name*> is the tag expression and
  Blair Pascal is the substitute.

4.3.2 Use of Templates

Nested contexts are accessed by tag expressions, where the single variables in the tag expression describe the "path" in the context tree (e.g. <*x.y*>). If a variable does not match a context, the tag is replaced with the empty string.

General rules:

- Processed tag expressions can be automatically URLEncoded using the URL_ENCODE modifier,
  example: *<this.will.be.Urlencoded[URL_ENCODE]>*
- The slash (/) is the escape character in templates. This means that angle brackets prefixed by the slash are not treated as tags
  example: < has to be escaped by */<*
  > has to be escaped by */>*

Template grammar:

- template ::= (plain | tag)*
- tag ::= "<" tagexpr ">" | "<" tagexpr (modif_seq)">"
- tagexpr ::= simpletagexr ("." simpletagexr)*
- simpletagexpr ::= alpha alphanum*
- alpha ::= "A" | ... | "Z" | "a" | ... | "z" | "_"
- alphanum ::= alpha | "0" | ... | "9"
- reserved ::= "<" | ">"
- unreserved ::= ... all characters except the reserved ones
- plain ::= unreserved*
- modif_seq ::= (alphanum+ " ")+

Example:

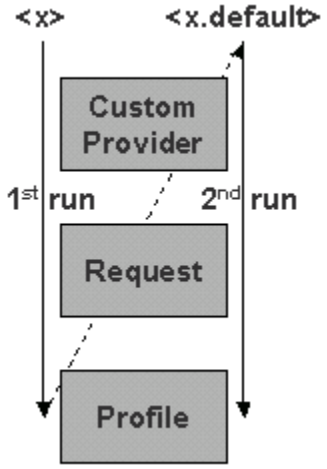
<Protocol>://<HostAddress>:<Port>/some/path?user=<User.Name[UPR_CASE]>&url=<HomeServer.url[URL_ENCODE UPPERCASE]>

Remark: The question mark (?) introduces the parameters in an HTTP/S url.

4.3.3 Resolution of Expressions

4.3.3.1 Expressions with Variables Only

Expressions with variables only look like <*x*>. They are resolved from a custom provider, the request and the iView profile (in this order). This is performed in two iterations. The first iteration checks for variables with the name "x", and the second iteration checks for "x.default".



4.3.3.2 Expressions Using a Context

Expressions using a context like <*x.y*> offer data retrieval from data providers such as the system landscape or user management. By using a context, it is possible to access a specific property. This could be necessary to overcome the resolution rules explained in 4.3.3.1. For example, the same property exists in the profile and in the request. In order to access this property from the profile, you have to explicitly access the property with <*Profile.property*> because accessing it with <*property*> would return the property from the request.

The resolution of expressions can be performed either indirectly or directly:

- **Indirectly**: An expression like <*System.url*> is resolved indirectly since the name for the system is not initially known. It has to be retrieved from the iView profile before the system landscape can be accessed.
- **Directly**: If the context is known an expression like <*User.UserID*> is resolved directly.

4.4 Available Variables

- All properties according to the rules given in 4.3.3.1: <*iViewParameterName*>
- User mapping user: <*MappedUser*>
- User mapping password: <*MappedPassword*>
- Authentication string: <*Authentication*>
- User ID used for accessing applications: <*LogonUserId*>. The <*LogonUserId*> is the portal user ID if no user mapping is maintained and is the result of <*MappedUser*> if a user mapping exists.

4.5 Available Contexts

4.5.1 Look and Feel Context (LAF)

This context provides the following information:

- Current theme: <*LAF.Theme*>
- Base URL of current theme: <*LAF.BaseUrl*>
- Complete stylesheet url: <*LAF.StylesheetUrl*>

4.5.2 Portal Component Profile Context (Profile)

This context provides information from the iView profile:

- Property name from the iView profile: <*Profile.[id]*>

4.5.3 Portal Context (Portal)

This context provides the following portal information:

- Portal Runtime version: <*Portal.Version*>
- Root url of portal: <*Portal.RootComponent*>

4.5.4 Request Context (Request)

This context provides the following information from the servlet request:

- Language: <*Request.Language*>
- Locale (as of SP2): <*Request.Locale*>
- Protocol: <*Request.Protocol*>
- Server name: <*Request.Server*>
- Port: <*Request.Port*>
- Server plus port: <*Request.ServerPort*>
- SSO2 ticket: <*Request.SSO2Ticket*>
- Any request parameter: <*Request.[ParameterName]*>

4.5.5 System Landscape Context (System)

This context provides all the properties of the system landscape. The possible variable names are defined by the system landscape itself:

- System property value: <*System.[SystemPropertyName]*>

4.5.6 User Context (User)

This context provides the following user information:

- Accessibility level: <*User.Accessibility*>
- Logon ID: <*User.LogonUid*>
- Logon ID: <*User.UserID*>
- User attribute: <*User.[AttributeName]*>

4.6 Available Modifiers

- *BASE64*: Performs base64 encoding (as of SP2)
- *HTML_ESCAPE*: Escapes the result so that it can be put in HTML documents
- *LOWERCASE*: Converts all characters to lower case
- *MD5*: Performs MD5 encoding (as of SP2)
- *SAP_BOOL*: Converts boolean values to SAP notation (true/1 -> 'X', false/0 -> '')
- *SAP_ITS_NAMESPACE*: Replaces '/' with '-' (used for IACs)
- *TRIM*: Removes all leading and trailing whitespaces
- *UPPERCASE*: Converts all characters to upper case
- *URL_DECODE*: Performs url decoding
- *URL_ENCODE*: Performs url encoding
- *PROCESS_RECURSIVE*: Forces recursive resolution of templates, i.e. if the result of a template is also a template, another recursive step of template processing is performed.

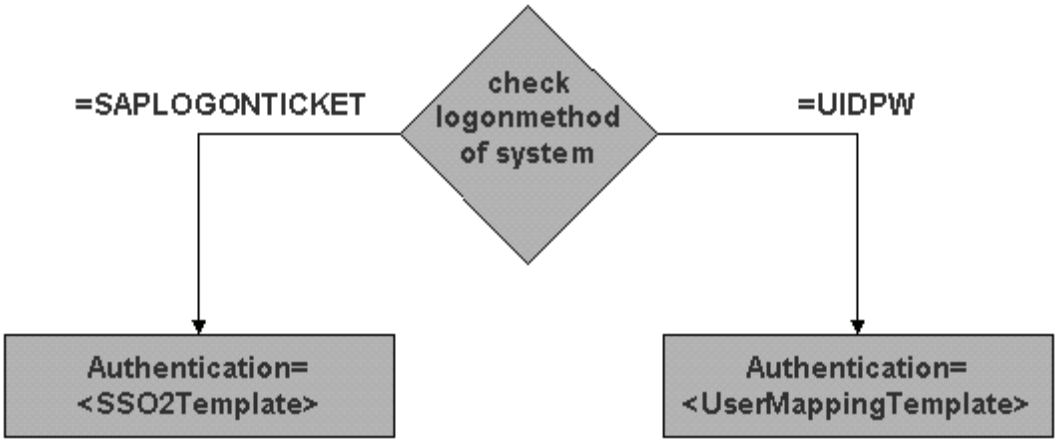
4.7 Single Sign-On

Using component com.sap.portal.appintegrator.sap.Generic permits you to perform a Single Sign-On to HTTP Web applications by including tag <Authentication> in the URL template.

Prerequisite is a system with a system definition that includes at least the following properties:

- user mapping type (if user mapping is used)
- logon method

The tag <*Authentication*> returns the result of either the <*UserMappingTemplate*> or the <*SSO2Template*> according to the following rule:



If you want to use user mapping, the *UserMappingTemplate* has to be set. This can be done in several ways:

- For basic authentication, it has to be set in the following way:
  <MappedUser>:<MappedPassword>@

  The *URLTemplate* would then look like this:
  http://<Authentication>protectedserver.com/xyz
- For HTTP form-based login, it could look like this:
  user=<*MappedUser*>&*password*=<*MappedPassword*>.
  Remark: Parameters *user* and *password* are application-specific. They need to be retrieved from e.g. the HTML source code of the logon form.

  The *URLTemplate* would then look like this:
  http://protectedserver.com/xyz?<Authentication>

If your system is set to log on with SAP logon tickets for authentication, you can specify an authentication template in parameter *SSO2Template*, e.g. *MYSAPSSO2*=<*Request.SSO2Ticket*>

4.8 Example (my.Yahoo.com)

The following example has four steps:

- Upload a new system template
- Create a system template for Web applications and a specific system for myYahoo.com
- Maintain the user mapping information
- Create an iView based on component "com.sap.portal.appintegrator.sap.Generic"

4.8.1 myYahoo.com Information

In order to log onto the Yahoo server (myYahoo.com), you need the following information, which can be retrieved from the HTML source of the Yahoo logon screen:

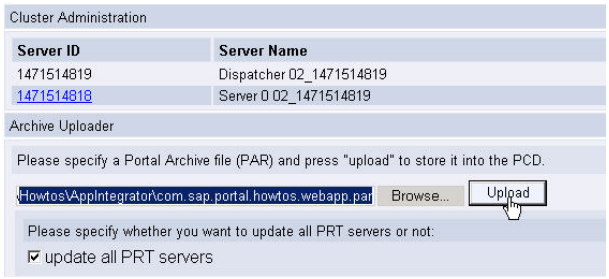
- (action) url: https://login.yahoo.com/config/login
- form method: *POST*
- parameter for user ID: *login*
- parameter for password: *passwd*

4.8.2 Creating a System Template for a Web Application and a System (myYahoo.com)

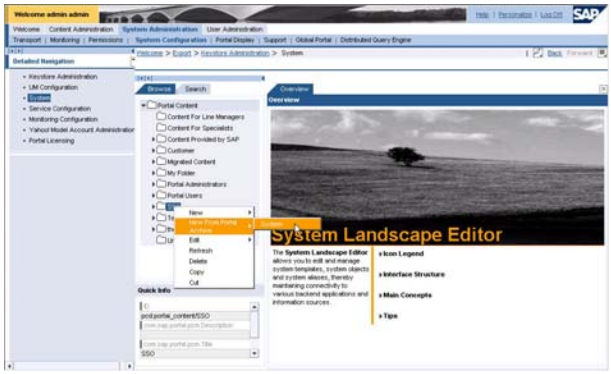
The system definition of a portalapp.xml needs to be packaged into a par-file.
Remark: You do not need to create a template, but it gives you the benefit of reusability.

The step-by-step procedure below uses the example file provided with the HowTo guide (com.sap.portal.howtos.webapp.par)

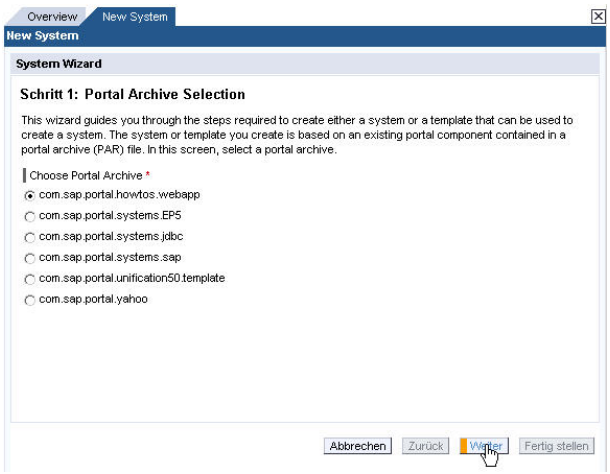
1. Upload file com.sap.portal.howtos.webapp.par to the portal using component "PortalAnywhere.default".



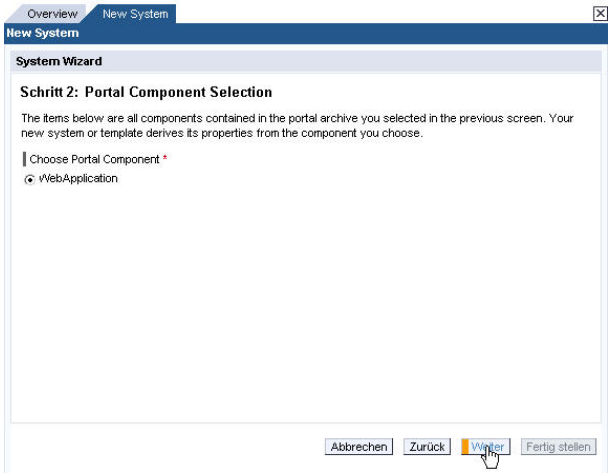
2. Create a system template by selecting "New from portal archive – System".



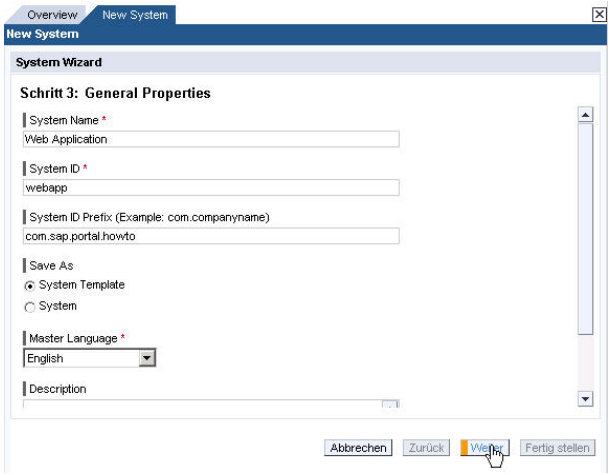
3. Select the portal archive "com.sap.portal.howtos.webapp". Click "Next".



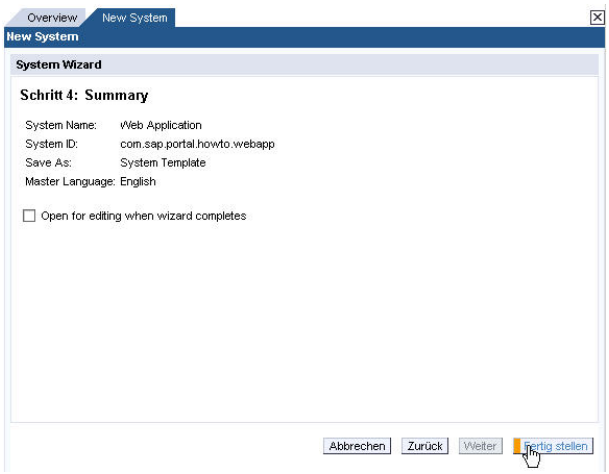
4. Select component "WebApplication". Click "Next".



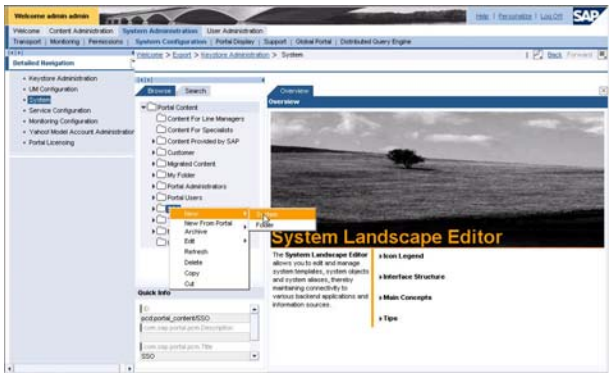
5. Define a name (e.g. "Web Application") and ID for the system template. Make sure that "Save As System Template" is selected. Click "Next".



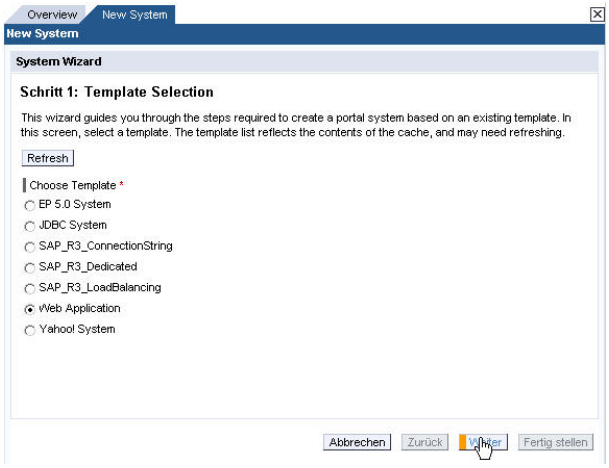
6. Finish system template creation by clicking "Finish".



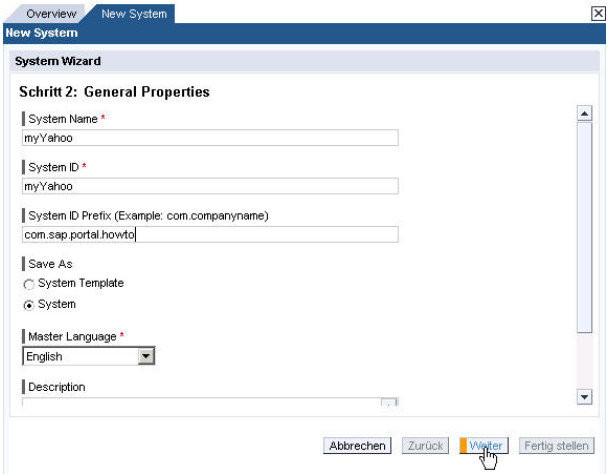
7. Create a new system by selecting "New – System".



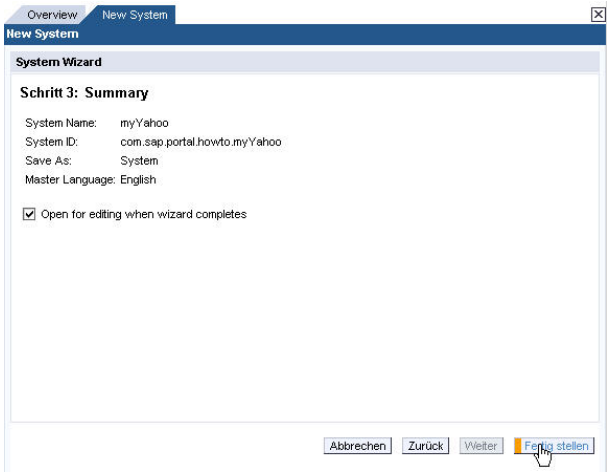
8. Select the template you created, e.g. "Web Application".
Click "Next".



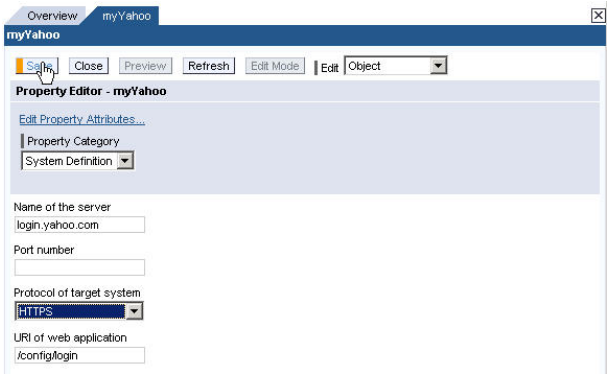
9. Define a name (e.g. "myYahoo") and ID for the system.
Click "Next".



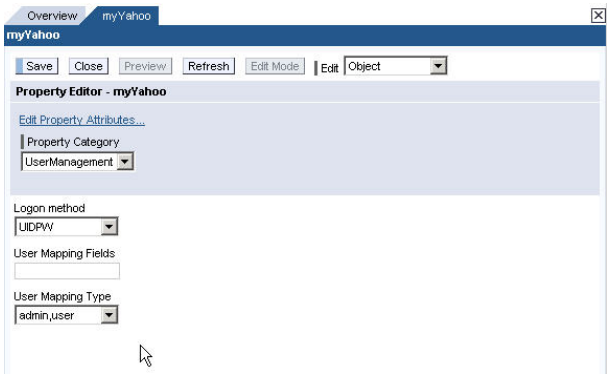
10. Finish system creation by clicking "Finish".



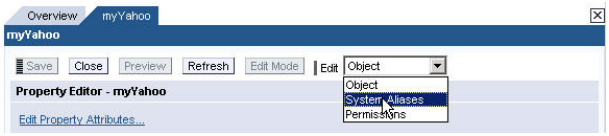
11. Configure the system definition properties of the myYahoo system (for the values, see above):
Name of server:
login.yahoo.com
Port number: *empty*
Protocol of the system : *HTTPS*
URI: */config/login*



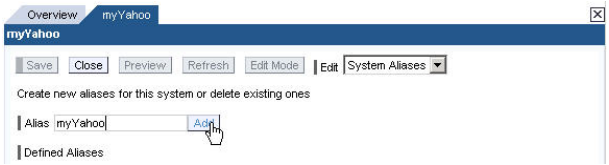
12. Configure the user management properties of the myYahoo system:
    Logon method: *UIDPW*
    User mapping fields: *empty*
    User mapping type : *<according to your needs>*



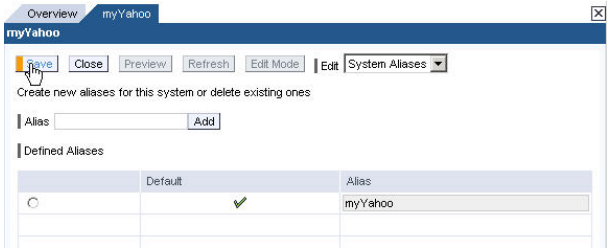
13. Open the system alias editor.



14. Add an alias "myYahoo".



15. Save your settings.



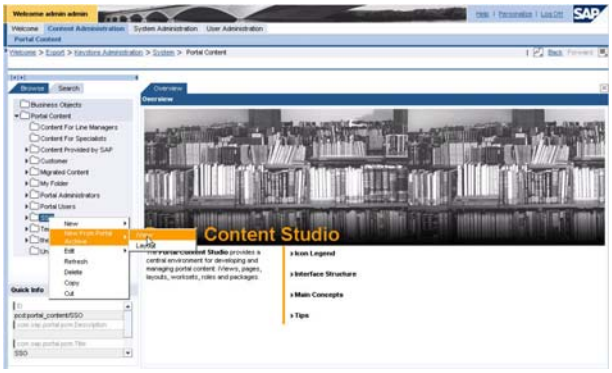
4.8.3 Maintain User Mapping

1. The user goes to Personalize -> Portal and maintains the user mapping for the "myYahoo" system.

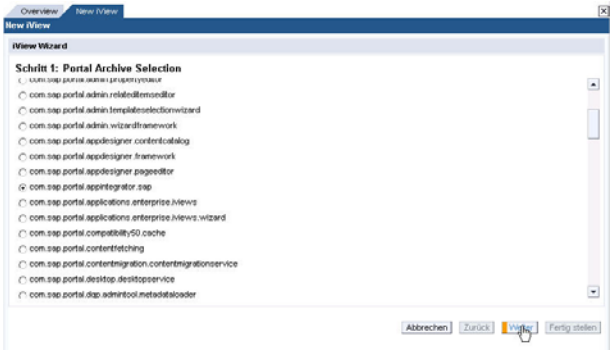
   Alternatively an administrator can do this in the user mapping administration.

4.8.4 Create an iView

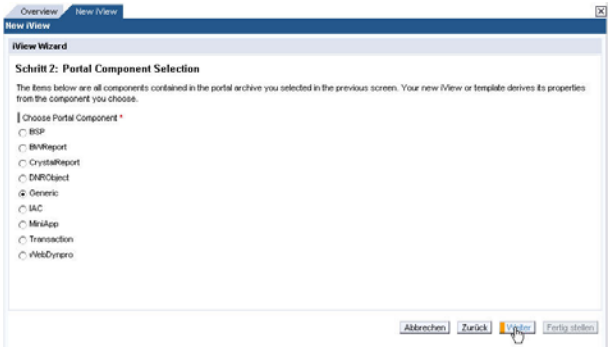
1. Create a new iView by selecting "New From Portal Archive – iView".



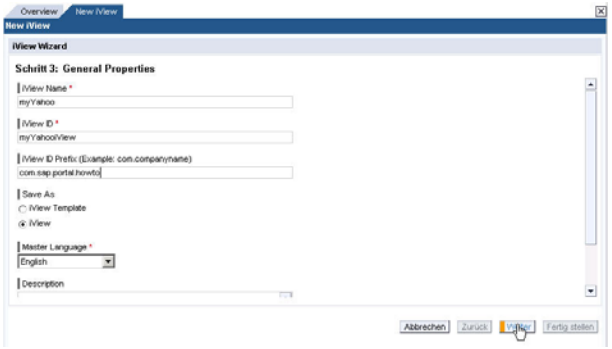
2. Select portal application "com.sap.portal.appintegrator.sap". Click "Next".



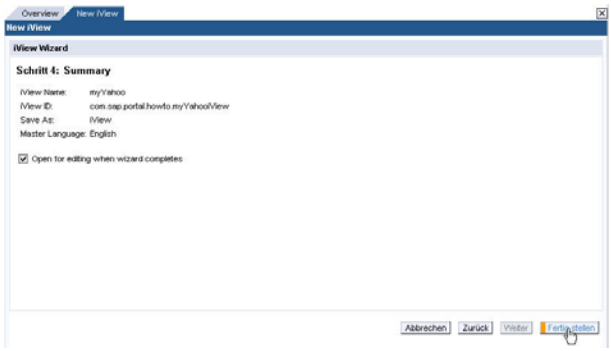
3. Select component "Generic". Click "Next".



4. Set the iView properties according to your needs. Click "Next".



5. Mark the checkbox "Open for editing when wizard completes" and finish creation of the iView by clicking "Finish".



6. Set the iView properties:

   System: *myYahoo* (system alias)

   URL template:
   <System.protocol>://<System.server><System.uri>?<Authentication>

   User mapping template:
   login=<MappedUser>&passwd=<MappedPassword>

4.8.5 Example portalapp.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application>
  <application-config>
    <property name="ServicesReference" value=""/>
    <property name="releasable" value="true"/>
  </application-config>
  <components>
    <component name="WebApplication">
      <component-config>
        <property name="ClassName" value=""/>
      </component-config>
      <component-profile>
        <property name="com.sap.portal.pcm.Title" value="WebApplicaton"/>
        <property name="ComponentType"
value="com.sapportals.portal.system"/>
        <property name="protocol" value="HTTP">
          <property name="validvalues" value="4/HTTP5/HTTPS"/>
          <property name="category" value="System Definition"/>
          <property name="personalization" value="none"/>
          <property name="type" value="String"/>
          <property name="plainDescription" value="Protocol of target
system"/>
          <property name="longDescription" value="Protocol of target
system"/>
        </property>
        <property name="server" value="">
          <property name="category" value="System Definition"/>
          <property name="personalization" value="none"/>
          <property name="type" value="String"/>
          <property name="plainDescription" value="Name of the server"/>
          <property name="longDescription" value="Name of the server"/>
        </property>
        <property name="port" value="">
          <property name="category" value="System Definition"/>
          <property name="personalization" value="none"/>
          <property name="type" value="String"/>
          <property name="plainDescription" value="Port number"/>
          <property name="longDescription" value="Port number"/>
        </property>
        <property name="uri" value="">
          <property name="category" value="System Definition"/>
          <property name="personalization" value="none"/>
          <property name="type" value="String"/>
          <property name="plainDescription" value="URI of web
application"/>
          <property name="longDescription" value="URI is the part after the
port, e.g. /irj/"/>
        </property>
        <property name="usermappingtype" value="admin,user">
          <property name="validvalues" value="4/user5/admin10/admin,user"/>
          <property name="category" value="UserManagement"/>
          <property name="personalization" value="none"/>
          <property name="type" value="String"/>
          <property name="plainDescription" value="User Mapping Type"/>
          <property name="longDescription" value="type of user mapping for
connection"/>
          <property name="defaultvalue" value="admin,user"/>
```

```
        </property>
        <property name="usermappingfields"        value="">
          <property name="category" value="UserManagement"/>
          <property name="personalization" value="none"/>
          <property name="type" value="String"/>
          <property name="plainDescription" value="User Mapping Fields"/>
          <property name="longDescription" value="additional comma
seperated user mapping fields"/>
        </property>
        <property name="logonmethod" value="UIDPW">
          <property name="validvalues"
value="14/SAPLOGONTICKET5/UIDPW8/X509CERT"/>
          <property name="category" value="UserManagement"/>
          <property name="personalization" value="none"/>
          <property name="type" value="String"/>
          <property name="plainDescription" value="Logon method"/>
          <property name="longDescription" value="Authentication method to
log to the SAP system"/>
        </property>
      </component-profile>
    </component>
  </components>
  <services/>
</application>
```

5 Developing your Own Application Integrator Component

Please refer to the documentation in the SAP Portal Development Kit, which can be found at http://www.iviewstudio.com

6 Making Parameters Dynamic (as of EP 6.0 SP2)

As of SP2, a user exit is provided to make retrieval of parameters more dynamic. This user exit permits you to plug in a custom provider for parameter retrieval. In this way you can code any algorithm for dynamic parameter retrieval and plug it into the Application Integrator.

You can do this with the following steps:

1.) Develop a new portal service (the custom provider)

2.) Upload the new portal service

6.1.1 Deployment Descriptor for New Portal Service (portalapp.xml)

The deployment descriptor has to look like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<application>

  <registry>
    <entry
path="/com.sap.portal.appintegrator/customer_exits/parameter_provider/
map_id1" name="IDProvider1" type="service"/>
  </registry>

  <application-config>
    <property name="SharingReference"
value="com.sap.portal.appintegrator"/>
  </application-config>

  <services>
    <service name="IDProvider1">
      <service-config>
        <property name="className"
value="com.customer.customerparameterprovider.IDProvider"/>
        <property name="startup" value="true"/>
      </service-config>
      <service-profile/>
    </service>
  </services>

</application>
```

Remarks:

- All parameters in **bold letters** have to be written as specified above (entry path, entry type, SharingReference, startup type)
- All underlined parameters can be custom defined (user exit ID = "map_id1", service name and className)

6.1.2 Implementing the Functionality

You must implement the interface

com.sapportals.portal.appintegrator.parameter.ICustomerParameterProvider

6.1.2.1 getParameter(…)

The method

```
public String getParameter(IPortalComponentRequest request, String id)
```

usually first tests if parameter "id" is relevant, e.g.

```
if ("System".equals(id)) {...
```

and returns for example the requested system identifier. For all other parameters, it simply returns "null"

```
} else {
  return null;
}
```

6.1.2.2 getParameterDefault(…)

The method

```
public String getParameterDefault(…)
```

can always "`return null`".

6.1.2.3 getAllParameterNames()

The method

```
public Enumeration getAllParameterNames()
```

has to return an enumeration of all parameter names (as String) that are handled by the custom provider in method getParameter().

6.1.2.4 getProviderName()

The method

```
public String getProviderName()
```

has to return the name of the custom provider for parameter handling. This is used for log file entries in case an error occurs.

6.1.3 Interface

```
/*
 * Created on 04.08.2003
 *
 * To change the template for this generated file go to
 * Window>Preferences>Java>Code Generation>Code and Comments
 */
package com.sapportals.portal.appintegrator.parameter;


import java.util.Enumeration;


import com.sapportals.portal.prt.component.IPortalComponentRequest;


/**
 * @copyright
 *
 * @version $Revision: #2 $.
 * @author SAP, Last modified by $Author: SAP $, Change list $Change:
102755 $.
 *
 */
public interface ICustomerParameterProvider {


  /**
   * Returns the value for the parameter defined by its <code>id</code>.
   * If the custom parameter provider doesn't define the parameter it
should return null.
   * All <code>java.lang.Throwable</code> that this method throws are
catched and logged;
   * further the custom parameter provider is ignored for this parameter-id
and treated as it
   * had returned <code>null</code>.
   * @param request the Portal component request
   * @param id the id of the parameter
   * @return the value for the parameter with id=<code>id</code> or null
   * @see #getParameterDefault(IPortalComponentRequest, String)
   * @see #getProviderName()
   * @see java.lang.Throwable
   */
```

```
  public String getParameter(IPortalComponentRequest request, String id)
throws Throwable;


  /**
   * Returns the default value for the parameter defined by its
<code>id</code>.
   * For details see {@link #getParameter(IPortalComponentRequest, String)}
   * @param request the Portal component request
   * @param id the id of the parameter
   * @return the default value for the parameter with id=<code>id</code> or
null
   * @see #getParameter(IPortalComponentRequest, String)
   */
  public String getParameterDefault(IPortalComponentRequest request, String
id) throws Throwable;


  /**
   * Returns an Enumeration of all parameters that are known by this custom
parameter provider
   * @return Enumeration of Strings of all known parameter ids
   */
  public Enumeration getAllParameterNames();


  /**
   * Returns the custom parameter provider's name. This method is called to
produce more accurate
   * log entries when an exception inside {@link
#getParameter(IPortalComponentRequest, String)} or
   * {@link #getParameterDefault(IPortalComponentRequest, String)} occured.
   * @return the name of this custom parameter provider
   * @see #getParameter(IPortalComponentRequest, String)
   * @see #getParameterDefault(IPortalComponentRequest, String)
   */
  public String getProviderName();
}
```