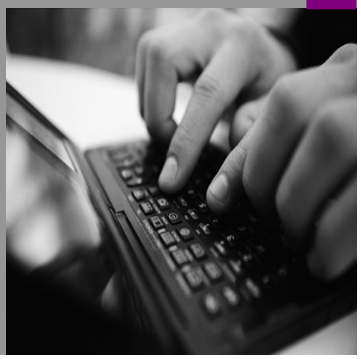

How-to Guide
SAP NetWeaver ‘04

How to Start Transaction iViews with a Dynamically Computed Language

Version 1.00 – October 2004

Applicable Releases:
SAP NetWeaver ’04
(SAP EP 6.0 SP2)

Copyright

Table of Contents

Table of Contents **3**
Preconditions **4**
Introduction **5**
Implementing the Customer Parameter Provider **6**
Testing the Solution **10**

Preconditions

You should be familiar with the fundamentals of Portal programming, especially Portal services within SAP NetWeaver™ Developer Studio. Furthermore, you should know how to access SAP systems and how to start transactions in an SAP system.

You need an SAP system with a user that has permission to start an arbitrary transaction. This SAP system has to be maintained in the Portal system landscape (see Note 761917). SSO is not required, but useful – of course you can use user mapping for testing purposes. You must also be a super administrator on the Portal to be used to test the solution.

The solution described in this guide is based on SAP NetWeaver ’04; however, it also works with some limitations on EP 6.0 SP2 on Web AS 6.20.

Introduction

This guide describes how to start transaction iViews with a dynamically computed language. SAP transactions always require a language for the logon to the SAP system; the Portal uses the user's portal language by default. However, the logon language can be adjusted individually for each iView by maintaining the iView property *Logon Language* – see Fig. 1.

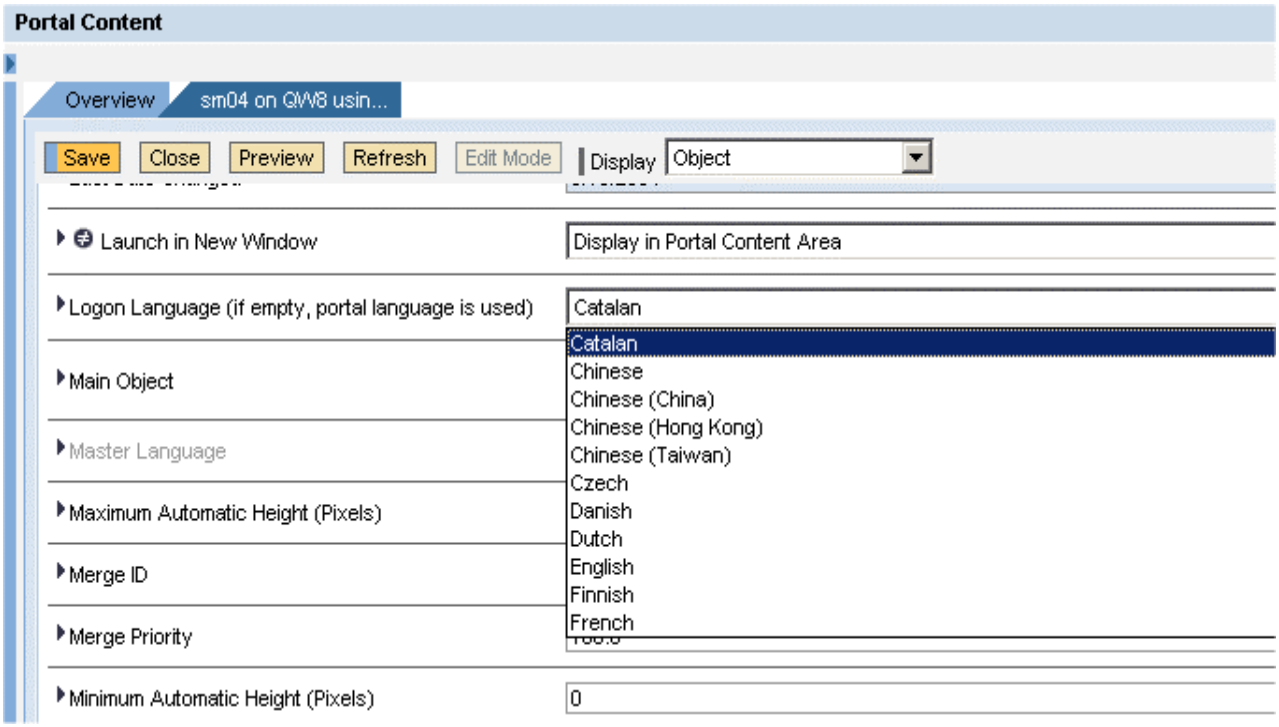

Fig 1: Setting the Logon Language for a Transaction iView

If you want more flexibility, use the *customer parameter provider* of the Portal's *Application Integrator* (for short, "App-Integrator"). The App-Integrator is a Portal application that provides a special component for each SAP application technology, such as GUI transaction, BSP, BI Report, IAC, or Web Dynpro application. The App-Integrator gets the required parameters to compute the target URL or connection string from the iView profile; however, the customer parameter provider allows you to implement your own parameter provision. Please refer to the document "*Using the Application Integrator"* on http://service.sap.com/ep60howtoguides for details regarding the customer parameter provider.

The objective is to write your own customer parameter provider that dynamically defines the logon language for transactions. Of course this solution is not only valid for SAP

transactions; it can also be used for any kind of SAP application covered by the App-Integrator.

Implementing the Customer Parameter Provider

A customer parameter provider is implemented as a Portal service. Therefore we have to create a new Portal application called *DynamicLanguageParamProvider*, which contains the Portal service called *TheService*. The service itself is implemented by a Java class called *DynamicLanguageProviderService*. The portalapp.xml contains a registry section where the customer parameter provider is registered as customer exit for the App-Integrator – the complete portalapp.xml file is shown in Listing 1.

```
<?xml version="1.0" encoding="utf-8"?>
<application>
  <registry>
    <entry path="/com.sap.portal.appintegrator/customer_exits/parameter_provider/
DynamicLanguage" name="TheService" type="service"/>
  </registry>
  <application-config>
    <property name="PrivateSharingReference" value="com.sap.portal.appintegrator"/>
  </application-config>
  <components>
  </components>
  <services>
    <service name="TheService">
      <service-config>
        <property name="className" value="com.customer.appintegrator.paramprovider.
dynamiclanguage.DynamicLanguageProviderService"/>
      </service-config>
    </service>
  </services>
</application>
```

Listing 1: portalapp.xml File of the Customer Parameter Provider Service

The Java class must implement the interface `com.sapportals.portal.appintegrator.parameter.ICustomerParameterProvider` provided by the App-Integrator – see Listing 2 (make sure that your project contains a reference to the App-Integrator public-jar).

Note that file portalapp.xml also contains a sharing reference to the App-Integrator application.

```
package com.customer.appintegrator.paramprovider.dynamiclanguage;

import com.sapportals.portal.appintegrator.parameter.*;

import […]

public class DynamicLanguageProviderService implements
   IDynamicLanguageProviderService, ICustomerParameterProvider {
[…]
```

Listing 2: Customer Parameter Provider Service Implements `ICustomerParameterProvider`

The App-Integrator calls `ICustomerParameterProvider.getParameter()` for all properties that are used to compute the target URL for the current application type. Since the language is not an iView profile property but is provided by a PRT-API call, there is no call to the customer parameter provider for the language itself. However, the language can be overwritten at the POM node for an iView.

When retrieving the locale using the PRT-API method `IPortalComponentRequest.getLocale()`, the PRT checks the adjacent POM node for an attribute called *"com.sapportals.portal.prt.component.IPortalComponentRequest.getLocale"*. If this attribute exists and contains a valid `java.util.Locale` object, the method returns this object as result. It is important that `ICustomerParameterProvider.getParameter()`immediately returns if the customer parameter provider is not interested in the given property. Otherwise execution of the App-Integrator and therefore the computation of URLs will slow down significantly. The implementation of `getParameter()`is shown in Listing 3.

Important note for EP 6.0 SP2: The customer parameter provider is available with EP 6.0 SP2, but with some limitations. All registered customer parameter providers are called for all iViews. It is not possible to restrict certain customer parameter providers to certain iViews; this feature is available with SAP NetWeaver '04. Therefore you need to make sure that a specific customer parameter provider handles only the properties for the relevant iViews.

```
// the PRT uses this key to check for a language at the POM node
private static final String LOCALE_NODE_KEY =
  "com.sapportals.portal.prt.component.IPortalComponentRequest.getLocale";

/* returns the customer value for relevant parameters, otherwise null */
public String getParameter(IPortalComponentRequest request, String id) {
  // this method should return immediately for uninteresting ids
  if (! isLocaleDefined(request)) {
    setLocale(request);
  }
  return null;
}

/* checks whether the language is already defined at the adjacent node */
private boolean isLocaleDefined(IPortalComponentRequest request) {
  return request.getNode().getValue(LOCALE_NODE_KEY) != null;
}
```

Listing 3: Implementation of getParameter()

The method `setLocale()`is only called when no locale is yet defined at the adjacent POM node (see method `isLocaleDefined()`). This ensures that the custosmer parameter provider does not create too much overhead. Furthermore, `setLocale(request)` uses the servlet session to store the computed locale, so that the computation is performed only once per user session. Of course the session key must be different for different scenarios. If you want different languages for different systems, the key has to contain for example the system ID.

```
// the computed language is stored in the servlet session using this key
private static final String LOCALE_SESSION_KEY =
  "urn:com.customer.appintegrator.paramprovider.dynamiclanguage:TheLocale.";

/* sets the locale at the POM node adjacent to the given request */
private void setLocale(IPortalComponentRequest request) {

  String sessionKey = getLocaleSessionKey(request);

  // get the locale from servlet session
  HttpServletRequest servletRequest = request.getServletRequest();
  HttpSession servletSession = servletRequest.getSession();
  Locale locale = (Locale) servletSession.getAttribute(sessionKey);

  if (locale == null) {
    // compute the language and store it in the servlet session
    locale = getDynamicLanguage(request);
    servletSession.setAttribute(sessionKey, locale);
  }

  // store the language at the node –
  // the PRT will consider this node in request.getLocale()
  request.getNode().putValue(LOCALE_NODE_KEY, locale);
}

/* returns the language using any algorithm or strategy you need */
private Locale getDynamicLanguage(IPortalComponentRequest request) {
  // TODO: add your custom language resolution strategy here

  return Locale.ENGLISH; // return "en" for demonstration purposes
}

 /* returns the key for the servlet session */
private String getLocaleSessionKey(IPortalComponentRequest request) {
  // TODO: implement your custom key computation here

  IPortalComponentProfile profile = request.getComponentContext().getProfile();
  String system = profile.getProperty("System");
  return LOCALE_SESSION_KEY + system;
}
```

Listing 4: Implementation of setLocale()

Listing 4 shows how this is done. At first `setLocale()` checks the servlet session. It calls `getLocaleSessionKey()` in order to compute a proper key for the given scenario. In this case the key is made up of a constant and the system alias ID, so that there is a separate entry in the servlet session for each system. If the session does not contain such an entry, the method `getDynamicLanguage()` is called; this method implements the custom scenario and dynamically maps the locale to the iView. In this case the method simply returns an English locale for demonstration purposes.

This method could for example look up a database table, user management, and/or the PCD in order to check the role of the current iView. The given `IPortalComponentRequest` provides all the information you need. The iView ID, for example, can be retrieved with the statement:

```
String iViewID = equest.getComponentContext().getComponentName();
```

The role of the iView can be determined with the statement

```
IUnit unit = initialPcdCtx.getUnit("pcd:" + iViewID);
```

Refer to the PCD documentation for details.

Testing the Solution

The implementation part is now finished and the dynamic language service can be tested in the Portal. After creating a PAR file for the *DynamicLanguageParamProvider* Portal application and deploying it to the test portal, the new customer parameter provider should be visible within the *Portal Registry Browser*[1] (see Fig. 2).

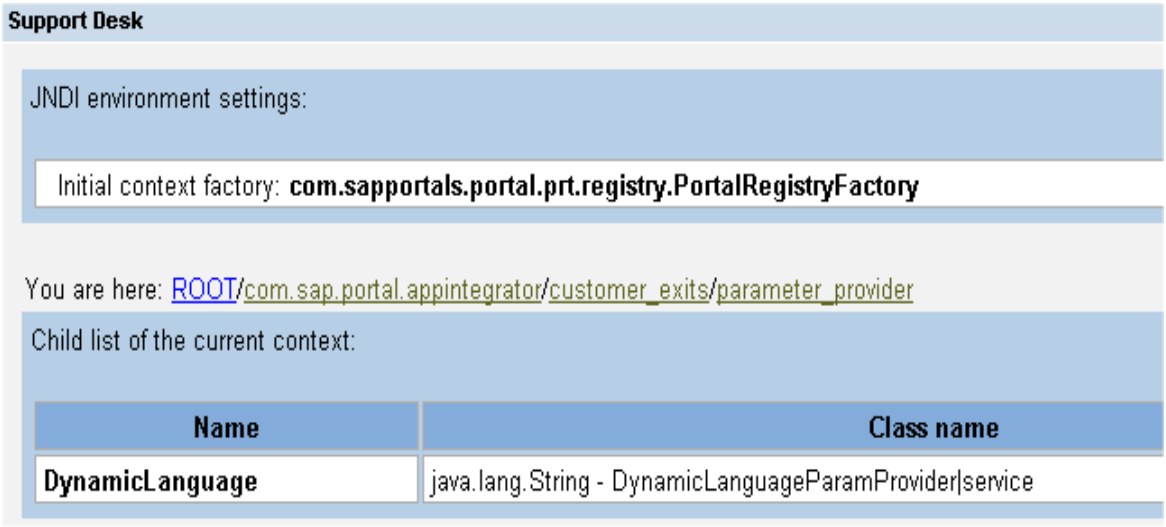

Fig. 2: Checking the Customer Parameter Provider in the Portal Registry Browser

In a second step you have to activate the customer parameter provider globally (this is not necessary in EP 6 SP 2). The customer parameter provider is switched off in NW04 by default in order to save the overhead occurring when looking up the Portal registry. Start the *Service Configuration* within the *System Administrator* role. Navigate through the tree to *Applications -> com.sap.portal.appintegrator -> Services -> Common_Configuration* and open the Configuration Editor using the context menu. Find the property *Use_CustomerExit_ParameterProvider* and set it to *true* – see Fig. 3. You only have to

[1] To start the Portal Registry Browser go to the *Support Desk -> Portal Runtime* and choose *Portal Registry Browser* in the *JNDI Browser* section.

perform this step once. If you want to disable all customer parameter providers globally, reset this property to *false*.

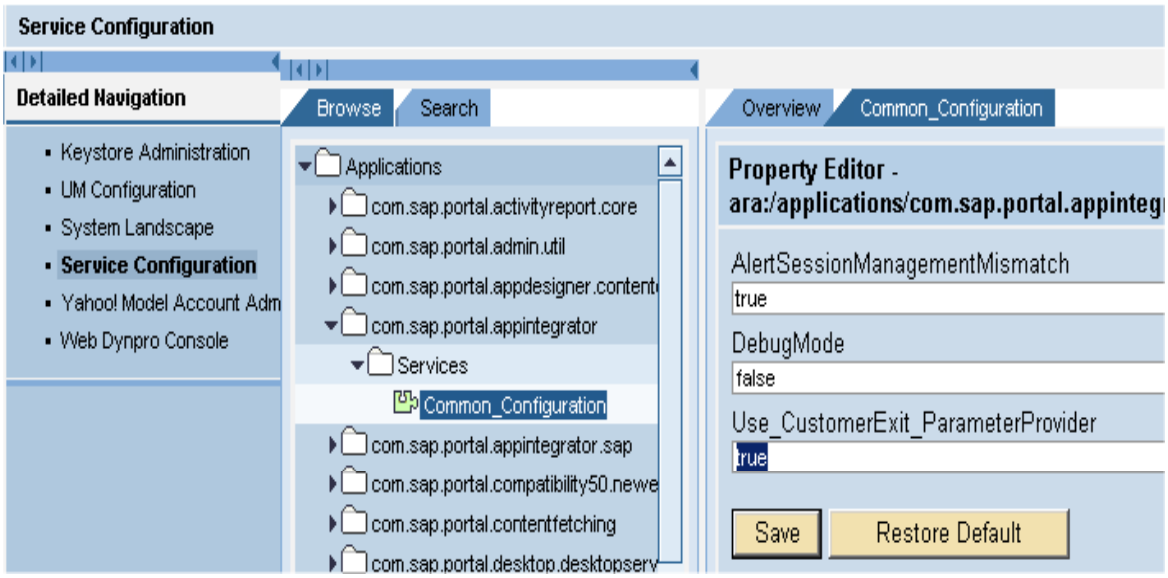

Fig 3: Activating the Customer Parameter Provider

You now have to assign the DynamicLanguage custosmer parameter provider to the relevant iViews (this step is not necessary for EP 6 SP2). Open the iView in the iView Editor, set the property *Customer Exits for 'ParameterProvider'* to *DynamicLanguage* and save the iView (see Fig. 4). Note that you might not need to set this property for all iViews, but only for a common *master iView*[2].

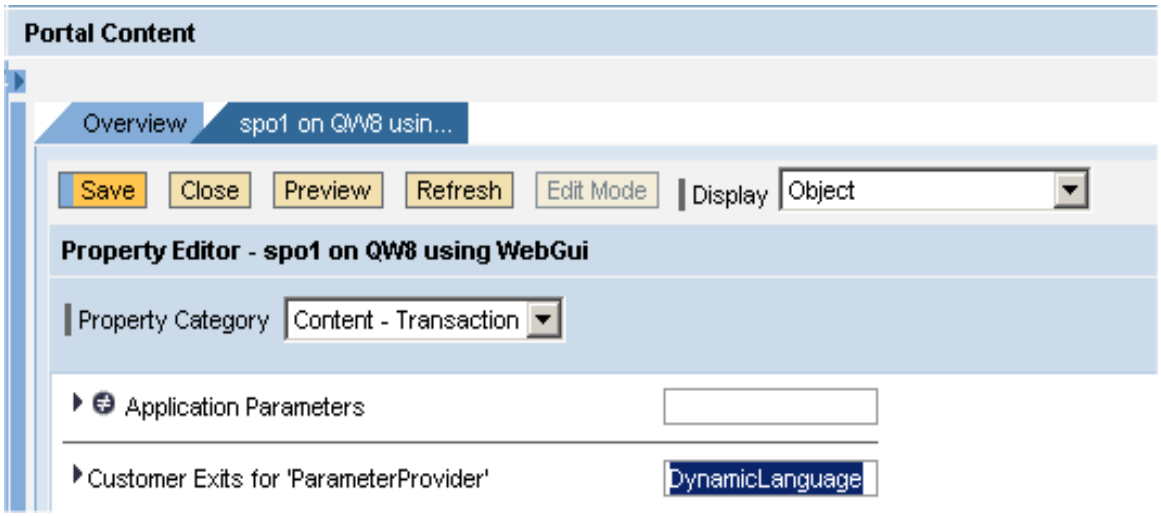

Fig. 4: Assigning the Customer Parameter Provider to an iView

[2] The template iView *pcd:portal_content/templates/iviews/sap_transaction_iview* located in the Content Catalogue at *Portal Content -> Templates -> iView Templates ->SAP Transaction iView* is a good candidate, because it's the root iView for all transaction iViews in the PCD.

If you now launch this iView, the corresponding transaction appears in English, regardless of the language defined in the Portal.