

Infopark CMS Fiona

Portal Manager

While every precaution has been taken in the preparation of all our technical documents, we make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein. All trademarks and copyrights referred to in this document are the property of their respective owners. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without our prior consent.

Contents

1 Preface	7
1.1 System Requirements	7
2 The Concept of the Portal Manager	8
2.1 Tasks of the Portal Manager	8
2.1.1 Access Control for Content	8
2.1.2 Dynamic Content	8
2.1.3 Portlets	9
2.2 Modular Configuration with Spring Beans	9
2.3 Architecture of the Portal Manager	11
2.3.1 Technical Components	11
2.3.2 Portal Manager and HTTP Server	11
2.3.3 Response Process	12
2.4 Authentication and Access Control	13
2.4.1 Definition of the Terms Used	13
2.4.2 Authentication Methods	13
2.4.3 Preventing Delivery of Protected Content	14
2.4.4 Hiding Links Pointing to Protected Content	14
2.4.5 Excluding Protected Content from Search Results	15
2.5 Providing Portlets	16
2.5.1 Local Portlets	16
2.5.2 Portlets in a Remote Infopark Portal Manager	16
2.5.3 Portlets via Web Services	16
2.6 Requirements	17
2.7 Restrictions	17
3 Using the Portal Manager	18
3.1 Access Control and Personalization	18
3.1.1 includePortlet	19
3.1.2 includePage	19
3.1.3 showIfAccessible	20
3.1.4 showIfMember	21
3.1.5 showIfLoggedIn	21
3.1.6 showIfLanguage	22
3.1.7 Parameterizing npspm Elements	22

3.1.8 Error Handling with Wrong npspm Tags	23
3.2 Creating News and News Feeds in the Editorial System	23
3.2.1 Functions for Creating News Feeds	23
3.2.2 Using the npsobj newslst Command	23
3.2.3 Examples	24
4 Installing and Configuring the Portal Manager	27
4.1 Using the Portal Manager with another Application Server	27
4.1.1 Using the Portal Manager with Tomcat 5.x	27
4.1.2 Using the Portal Manager with WebSphere Application Server 5.1	29
4.2 Standard Configuration	29
4.2.1 Servlet and Filter Definition	29
4.2.2 Portal Manager Configuration	29
4.2.3 Configuring a Portlet Web Application	30
4.2.4 Configuring a Portlet	31
4.2.5 Portlet Modes and Window States	32
4.3 User Management in the Portal	32
4.3.1 Using the Portal Manager with LDAP or ADS	32
4.3.2 Providing User Properties in Portlets	32
4.3.3 Retrieving User Properties from the Directory Service	33
4.4 Using the Portal Manager with Named Virtual Hosts	34
4.4.1 Prerequisites and Mode of Operation	34
4.4.2 Sample Configuration	34
4.5 Providing Portlets via WSRP	36
4.6 Making WSDL files Available	37
4.6.1 Using a Proxy Server	37
4.6.2 Making Included Files Locally Available at the Producer	37
4.6.3 Making Included files Locally Available at the Consumer	37
5 Portlets	39
5.1 News Portlet	39
5.1.1 Operation	39
5.1.2 Configuration	40
5.1.3 Usage	41
5.2 Login Portlet	41
5.2.1 Configuration	42

5.2.2 Usage	42
5.3 Portlet for Editing the Current Page	42
5.3.1 Operation	42
5.3.2 Configuration	42
5.3.3 Usage	42
5.4 Form Portlet	43
5.4.1 Configuration	43
5.4.2 Usage	46
5.5 Search Portlet	47
5.5.1 Usage	47
5.5.2 Configuration	47
5.5.3 Example	51
5.6 Portlet for Displaying and Sorting a Table	55
5.6.1 Operation	55
5.6.2 Configuration	55
5.6.3 Usage	55
5.7 Storing User-Specific Portlet Preferences	55
5.8 Note on Developing Portlets	57



1 Preface

This document addresses administrators who wish to use Infopark Portal Manager, which is part of Infopark CMS Fiona, for implementing functionalities such as access control, channels, personalization, or JSR 168-compatible portlets.

The document explains how to configure the Portal Manager, how to integrate it into a given IT landscape, and how to make use of existing portlets.

A license is required for using any of the supplied portlets except the login portlet.

1.1 System Requirements

The general [system requirements](#) also apply to the Portal Manager. Additionally, the following applies:

- If you wish to use authentication and authorization via LDAP, you require an LDAP Server.

2

2 The Concept of the Portal Manager

2.1 Tasks of the Portal Manager

The Portal Manager which is part of Infopark CMS Fiona offers companies and organizations the possibility to present to their customers and employees any kind of content in a common user interface and with a common look and feel.

- Access restrictions can be applied to individual files or document parts meaning that access to these files is given to authorized persons only after logging in.
- Furthermore, content can be made dynamic, i.e. adapted to individual users or user groups.
- Last but not least, the Infopark Portal Manager supports Java Portlet technology which makes it possible to enrich content by embedding interactive components – so-called portlets – into it.

These functions can easily be accessed by editors and designers working with the editorial system.

2.1.1 Access Control for Content

Access Control makes it possible to restrict the access to content to users who are a member of a particular user group. To determine a user's group memberships, she is asked to log in as soon as she requests access-restricted content. If the authorization fails, the request is rejected. This applies to access-restricted HTML pages as well as binary content such as PDF files.

Additionally, it might also be desired to suppress hyperlinks to access-restricted content if the user is not permitted to view the link target. Alternatively, a teaser might be displayed. The Portal Manager offers several functions with which the desired behavior can be achieved.

Access restrictions can be applied to web pages as a whole or to parts of them. If, for example, you wish to offer special services to the users logged into your site, you can have the links to the corresponding pages displayed only for logged-in users.

2.1.2 Dynamic Content

What a web page contains is often not determined in advance (statically) but in the moment the page is delivered. This allows you let the page contain different content, depending on the conditions that are met when the page is requested. You might, for example, want to include links to the last five documents the logged-in user has viewed at his previous visit to your site.

Furthermore, content that often changes and is displayed at several locations of the web site can be included dynamically when the pages are delivered. Thus, only one instance of the content needs to be

created and maintained. This avoids redundant work and reduces the time needed for exporting the pages into which the content is embedded.

2.1.3 Portlets

Portlets are independent and interactive components of HTML pages. Typical use cases for portlets are ranking and voting, logging-in, multi-page forms, or news tickers.

The Portal Manager includes a JSR 168 compatible portlet container. JSR 168 is a standard to which many application servers comply. Thus, if required, the portlets developed for use with the Portal Manager can be used with other application servers as well. Furthermore, JSR 168 portlets are modular, meaning that they are part of the content and not vice versa, making it easy to change their arrangement on web pages. The portal software of many other vendors requires a fixed page layout that can only be changed with great effort.

Compared to other mechanisms for generating content dynamically (PHP, for example), Java-based portlets have the following advantages:

- The code is clearly distinct from the content. Portlets are embedded into HTML pages only by declaration (using `np:sp:pm` tags);
- Object orientation ensures clear interfaces and data structures;
- A higher grade of modularity has the effect that code can be reused much better.

2.2 Modular Configuration with Spring Beans

Infopark's Java web applications consist of several components. By means of the [Spring Framework](#) these components can be easily exchanged and adapted. Infopark CMS Fiona includes several components, for example for the following tasks:

- Determining users
- Delivering content
- Access control
- Authentication

Each of these components is a so-called bean. The combination of beans to be used in the web application can be specified by means of one or several XML files. As a result, the available functions and the behavior of the application can be modified by adapting the configuration files accordingly. This also allows for integrating custom components.

Every bean is based on a Java class. If a bean is used, the Spring Framework creates an object of the corresponding class. A bean can be integrated as follows:

```
<bean id="sessionInvalidator" class="com.infopark.pm.DefaultSessionInvalidator"/>
```

An optional `id` makes it possible to refer to the same bean in different locations.

If a bean has properties that need to be adapted to the installation environment, you can set their values in the following way:

```
<bean id="httpHelper" class="com.infopark.libs.http.ApacheHttpHelper">
  <property name="connectionTimeout" value="3"/>
  <property name="defaultSocketTimeout" value="60"/>
</bean>
```

```
<property name="maxConnections" value="10"/>
</bean>
```

A bean can access the functionality of another bean. The `permissionManager` bean, for example, requires access to content. For being flexible when choosing the bean that supplies the content, the `permissionManager` bean has the `documentSource` property. To this property the bean to be used can be assigned as a reference. In the following example, the `externalDocumentSource` bean is integrated for providing content. Then, the `permissionManager` bean is included and configured to access the content via the `externalDocumentSource` bean. To the properties `permissionReader` and `permissionResolver` beans are assigned as well. Since the respective bean is only used here, it is integrated directly in the property, i.e. without using an id.

```
<bean id="externalDocumentSource" class="com.infopark.pm.doc.FileDocumentSource">
  <property name="documentRoot"
    value="/opt/infopark/fiona/instance/default/export/online/docs"/>
  <property name="inputEncoding" value="UTF-8"/>
</bean>

<bean id="permissionManager" class="com.infopark.pm.doc.DocumentPermissionManager">
  <property name="documentSource" ref="externalDocumentSource"/>
  <property name="permissionReader">
    <bean class="com.infopark.pm.FionaPermissionReader"/>
  </property>
  <property name="permissionResolver">
    <bean class="com.infopark.pm.FionaPermissionResolver"/>
  </property>
</bean>
```

The properties of a bean are determined by its class. To be able to set the value of a property, the class needs to have a public method. The name of this method is composed of the string `set` followed by the name of the property. Exactly one argument is passed to this method, namely the value of the property. In the example above, the `setDocumentSource()` method of the `com.infopark.pm.doc.DocumentPermissionManager` class is called. For the list of the public methods of the Java classes included in Infopark CMS Fiona, please refer to the API documentation located in the `share/doc/javadoc/pm` directory in the installation directory of the CMS.

Debugging Hints

Configuration errors have the effect that the web application concerned can no longer be [deployed](#). Please refer to the log file of the web application for details about the error that occurred. Errors like this always raise a `BeanDefinitionStoreException`. Typical reasons for this error include:

- Line ... in XML document ... is invalid

The configuration file contains invalid XML. Check the file for syntactical correctness.

- Bean class [*ClassName*] not found

The Java class *ClassName* specified as the bean's `class` is unknown. The class name might have been misspelled, or the class can neither be found in the `WEB-INF/classes` directory nor as a jar archive in the web application's `WEB-INF/lib` directory.

- Error setting property values

The value of the property could not be set. If the name has not been misspelled, the method might not be (`public`) or the value does not suit the method's argument type.

- Could not instantiate class [*ClassName*]

The specified Java class, *ClassName* was found. However, an object could not be created. A subclass of this class may be needed instead.

2.3 Architecture of the Portal Manager

2.3.1 Technical Components

Infopark Portal Manager is a standard servlet application which is deployed into a servlet container (such as Trifork Application Server). It consists of filters, servlets, and beans plus a few supporting components.

Filters

Filters analyze and modify incoming requests. They are organized in a so-called filter chain. Before a request is answered, it is processed by all or a subset of the filters in a defined order. On its way back to the user, the filters are applied once more, in the reverse order. The filters may add information to the request and the response, modify both or even block them. Depending on the required functionality, filters can be exchanged, removed or added. The most important filters supplied with Infopark Portal Manager are:

- NTLMFilter
- AuthenticationFilter
- PortletFilter
- ContentFilter
- PermissionFilter

Servlets

The task of servlets is to deliver content. In the Portal Manager, a servlet for delivering static (binary) and dynamic data exists, the `ContentServlet`.

As is common with Java web applications, the filters and servlets are configured by means of the `web.xml` file.

Beans

Beans are exchangable modules that provide particular functions to filters and servlets. They are defined and configured by means of the `pm.xml` file. Important beans are, for example, the `userManager`, `portletContainer`, `documentManager`, and `permissionManager` beans.

2.3.2 Portal Manager and HTTP Server

In principle, every servlet container also is a web server. Nevertheless, it is currently recommendable to let an HTTP server work as a web server in front of the Trifork server. Because of its popularity and stability we recommend to use Apache HTTP Server. The reasons to use an HTTP server are as follows:

- The option exists to use privileged ports such as 80 or 443 without having to start the webserver as a superuser. Currently, this is only supported by an HTTP server.
- To be able to continue to use existing log file analyzers, log files in the commons-logging format as produced by Apache HTTP Server are required. At present, Trifork Application Server cannot

generate log files in this format. From version 6.5, Infopark CMS Fiona provides a corresponding logging filter.

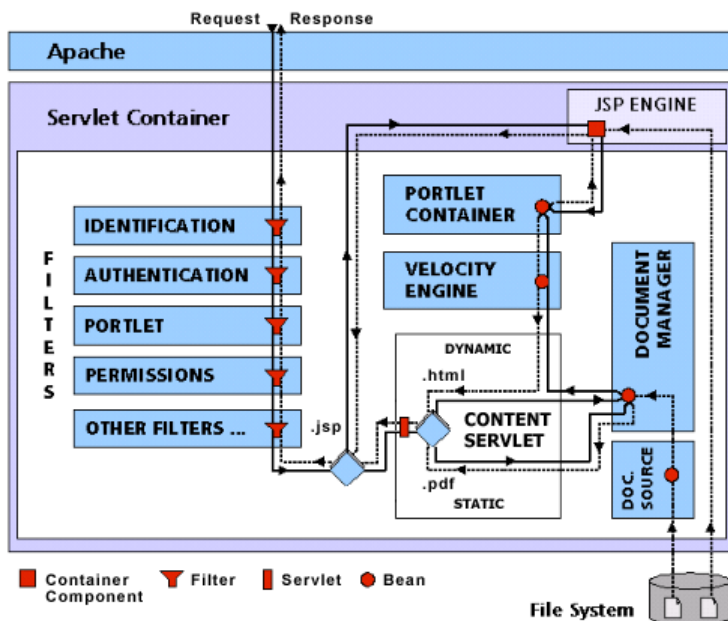
- Apache HTTP Server encrypts HTTPS pages very fast.

Sometimes, other reasons for placing an HTTP server in front of the Trifork server exist:

- Via the HTTP server, additional authentication mechanisms can be used to set the `RemoteUser` which can then be evaluated by the Portal Manager.
- Other modules such as URL rewriting, response compression, bandwidth restriction and the like can be used in conjunction with the Portal Manager.

2.3.3 Response Process

In the following, assuming the configuration specified above (Apache, Trifork, Portal Manager), the process of responding to a typical request is explained. The file requested is an HTML file containing a portlet.



The diagram above illustrates the process for JSPs as well as for binary content.

- Via a browser, a website visitor sends an HTTP request to port 80 of the webserver.
- An Apache webserver accepts the request and redirects it via `mod_jk` to the Trifork server.
- The Trifork server calls the following filters one after the other for the request:
 - Identification via `NTLMFilter`, for example. This filter determines the identity of the visitor and sets the `RemoteUser`.
 - The `AuthenticationFilter` queries the ADS for the `RemoteUser` and creates a user object.
 - The `PortletFilter` checks whether the incoming request is a portlet request. If so, the filter redirects the request to the portlet container.
 - The `PermissionFilter` checks whether access restrictions exist for the requested file. If so, it checks whether the file may be given to the user and ends the processing of the request if this is not the case.

- With the help of the configuration in the `web.xml` file, the Trifork determines that the `ContentServlet` is responsible for handling the request.
- The `ContentServlet` requests the file from the `DocumentManager`.
- In live mode (as opposed to preview mode), the `DocumentManager` fetches the file via the `FileDocumentSource` from the online directory tree, which was created by the Template Engine.
- The `ContentServlet` determines the MIME type of the document and delivers the content either as static or dynamic content, depending on the MIME type.
- The HTML file will be delivered dynamic by processing the document as a Velocity template.
- The Velocity engine recognizes a command embedding a portlet and passes the request to the `PortletContainer`.
- The `PortletContainer` sends a render request to the specified portlet and returns the HTML code it receives as a response to its request.
- The `ContentServlet` delivers the assembled page.
- All filters, in reverse order, are now given the opportunity to modify the outgoing response. The `AuthenticationFilter` uses it and adds a cookie for faster authentication at the next request.

2.4 Authentication and Access Control

2.4.1 Definition of the Terms Used

Authentication is a two-step process. In step one, a user identifies himself, for example by means of a user name (login) and a password. In step two, the user's identity is checked.

In a computer program identities (i.e. identified persons) are usually associated with permissions. Authorization denotes the process a computer carries out to check whether a particular identity possesses a particular permission, for example the permission to read a particular file.

2.4.2 Authentication Methods

The authentication methods available in the Portal Manager can be extended by providing the filters that implement the desired authentication functionality. In the standard scope of supply the following mechanisms are included.

AuthenticationFilter

In Infopark CMS Fiona, the `AuthenticationFilter` supports the following mechanisms:

- Basic Authentication by means of a user name and a password.
- Single-sign-on tickets, for example for SSO against an SAP Enterprise Portal.
- Adopting a `RemoteUser` that has already been set. This makes it possible to use any method that has been placed in front of the Portal Manager.

Single Sign On via Windows' NTLM

The `NTLMFilter` determines a `RemoteUser` via NTLM. Therefore, when using Windows and the Internet Explorer (restrictions apply to Firefox), the user needs to log-in only once to Windows. Subsequently, his log-in information is also available in the Portal Manager.

The NTLMFilter enforces authentication. However, falling back to Basic Authentication can be enabled for authenticating users even if NTLM authentication fails.

Particular URLs and IP ranges can be exempted from authentication.

Handling anonymous users

If no explicit log-in request is issued and the `RemoteUser` has not been set, the request is anonymous. If an anonymous user accesses a protected file, the `PermissionFilter` blocks the request. Instead of the file, the HTTP response code "401 Unauthorized" is delivered to the user. Typically, the servlet container redirects this response to an error page that has been defined in the `web.xml` configuration file.

In the delivery state of the Portal Manager, anonymous users cannot access content because the `NTLMFilter` enforces authentication. However, by specifying an appropriate URL pattern in the `web.xml` configuration file of the filter, particular sections of the website can be exempted from authentication, making them available to anonymous users.

2.4.3 Preventing Delivery of Protected Content

The delivery of textual as well as binary content can be restricted to particular user groups.

Access permissions are checked on the file level. Typically, a file either has no access restrictions (its content may be accessed by everyone), or user groups permitted to read the file have been defined (protected content). In the first case, the file is simply delivered. In the second case the Portal Manager checks whether reasons for delivering the file to the user exist. One such reason is that the user is a member of at least one user group permitted to read the file.

Alternatively, any number of additional checks can be configured. The scope of delivery includes the option to check access permissions on the basis of the IP address of the computer requesting a file.

The user groups permitted to read an exported file can be assigned to the corresponding file in the editorial system in two ways: by means of the Content Navigator or via the Tcl interface of the Content Management Server (CM). The permission concerned is labelled *Live server read permission*, its Tcl keyword is `permissionLiveServerRead`.

2.4.4 Hiding Links Pointing to Protected Content

The Portal Manager supports the following two mechanisms for suppressing hyperlinks that point to protected content.

Automatic Link Removal

The Template Engine can be configured to suppress links that point to protected content. A link is suppressed by removing the `href` attribute from the `a` tag. To achieve this, every `href` attribute is embedded into Velocity code at export time. At request time, this code checks the user's permission to access the link target and removes the `href` attribute if the check fails.

The advantage of this method is that it is safe and reliable. No links to protected files will ever be published, even if links to such files are placed into the content by, for example, the editorial staff.

However, the method has the following disadvantages:

- Only the link itself is removed. The linked text, for example the title of the protected document, as well as HTML code possibly used for formatting purposes (like a table cell in a menu) are kept.
- It can be enabled and disabled globally only.
- It slows down the delivery of the pages because additional Velocity code needs to be processed for every link.

Automatic link removal can be switched on and off in the `export.xml` file by means of the `export.convertNpspm.hideForbiddenLinks` configuration entry. It is enabled by default.

npspm Elements

By means of `npspm` elements, layouters are given a means to directly modify the HTML code with which, for example, link lists are displayed, causing the relevant parts to be only visible if the user meets particular criteria:

- `npspm_showIfAccessible`
- `npspm_showIfLoggedIn`
- `npspm_showIfMember`

2.4.5 Excluding Protected Content from Search Results

Files a user is not permitted to access should not show up in his search results. For this purpose, the user groups permitted to read a file are indexed together with the contents of the file. This makes it possible to add the required group-membership condition to the search query. Thus, documents containing the search words will be added to the search result only if no access restrictions apply, or if the user is a member of one of the given user groups.

The following sample configuration for the search portlet (`SearchPortlet`) implements such a query:

```
#macro (getPermissionQuery)
  #set($groups = "")
  #foreach ( $perm in $user.permissions)
    #if ("${groups}" != "")
      #set ($groups = "$groups , "$perm.trim()" )
    #else
      #set ($groups = "$perm.trim()")
    #end
  #end
  #if ("${groups}" != "")
    <#ANY>((($groups) <#IN> permissionLiveServerRead), (
      "free" <#IN> noPermissionLiveServerRead))
  #else
    <#YESNO>("free" <#IN>noPermissionLiveServerRead)
  #end
#end
...
<condition>
  <and>
    [Place your search query here]
    <vql-statement>#getPermissionQuery</vql-statement>
  </and>
</condition>
...
```

From Infopark CMS Fiona, version 6.5, the Portal Manager also supports base queries in addition to the actual search queries. The base query is made prior to the actual search query. Its purpose is to reduce the number of documents to be searched. It simplifies the code considerably:

To the and section of the base query the following is added:

```
<or>
  <vql-statement>
    "free" <#IN>noPermissionLiveServerRead
  </vql-statement>
  #foreach ($group in $user.permissions)
    <vql-statement>
      "$group" <#IN>permissionLiveServerRead
    </vql-statement>
  #end
</or>
```

2.5 Providing Portlets

For integrating more portal functions and services, the Infopark Portal Manager offers three ways to make portlets available.

2.5.1 Local Portlets

Local portlets are executed in a web application located in the same Application Server as the Portal Manager itself. Compared to remote portlets this kind of making portlets available offers the best performance since no communication via the network is required.

2.5.2 Portlets in a Remote Infopark Portal Manager

Under particular circumstances it makes sense to operate several Infopark Portal Managers:

- Security issues: Portlets processing data critical to safety should be executed on a server protected against attacks.
- Reusability: If several servers are operated in different locations, it is sufficient to make common portlets available only on one server. This makes it easier to maintain these portlets.
- Load issues: Computationally intensive portlets can be executed on a separate server to improve the performance of the complete system.

2.5.3 Portlets via Web Services

Infopark Portal Manager makes it possible to use portlets provided by web services. Infopark Portal Manager accesses these remote portlets by means of HTTP requests to the server that runs them.

The communication between the servers is based on the WSRP specification (Web Services for Remote Portlets). Commonly, the remote server is a JSR168 portal server. The party that uses portlets offered by a web service is called consumer, the party that provides them is the producer.

Thus, WSRP creates a common platform for remote portal functionality. By using Infopark Portal Manager as a WSRP consumer, portlets running on other systems such as IBM Websphere Portal Server can be utilized.

2.6 Requirements

From version 6.5.1, Infopark Portal Manager, which is included in Infopark CMS Fiona, can be used as a WSRP 1.0 consumer.

The following requirements must be met to be able to use remote portlets in conjunction with the Infopark Portal Manager:

- A portal server must exist that offers remote portlets in accordance with the WSRP 1.0 specification and that serves as a producer.
- The portlets offered via web services must be present on the portal server.
- Infopark Portal Manager and the producer must be able to communicate with each other via HTTP/HTTPS.
- The WSDL file describing the service as well as all the schemes and name spaces included in it must be accessible to Infopark Portal Manager even if they are located on remote servers.
- The URL of the WSDL file for this service must be known.

Example:

`http://portalstandards.oracle.com/portletapp/portlets?WSDL`

2.7 Restrictions

Although a WSRP specification exists, some vendors might have included proprietary features in their products. Currently, Infopark Portal Manager does not support proprietary features.

Infopark Portal Manager offers an implementation of the WSRP 1.0 specification, except for the following features:

- Producer Mediated Sharing (`CookieProtocol:perGroup`) is not supported.
- Anchors (`wsrp-fragmentID`) are not supported.

3

3 Using the Portal Manager

3.1 Access Control and Personalization

Access control is one of several methods to present personalized information to the visitors of your website, or to suppress the delivery of content. This concept is often understood solely as a means to restrict the visitor's access to information. However, displaying different kinds of content selectively to different groups of users is an equally important goal of access control.

Access control presupposes the possibility to assign read permission to live documents. This possibility exists in the editorial system where any number of user groups can be added to the live server read permission field of each file.

When the Portal Manager delivers content it takes account of the permission assignments that have been made. If access to a document has been restricted via the CMS to particular user groups on the live server, the Portal Manager will give access to the document only after the user has been successfully authenticated. Links to this document are deactivated if required.

However, it is not only possible to grant or deny permission on a per-file basis but also partially. For this, a special language element, `<npspm>`, is available that can be used in the content itself. The `npspm` element can be inserted into the content of a file like a normal HTML element is inserted.

Analogously to `npsobj` elements, which are translated into HTML text during the export, `npspm` elements are translated into a language evaluated on the server side. One of the supported languages is Velocity because the Portal Manager includes a Velocity engine. The Portal Manager itself is a web application that runs in a servlet container which includes a JSP engine. Therefore, JSP is another language into which `npspm` elements can be translated during the export (see also [Architecture of the Portal Manager](#)). A PHP page cannot contain Portal Manager functionality such as access control or portlets because the server can evaluate only one language per file.

The language into which `npspm` elements are translated during the export depends on the name extension of the file being exported. You can assign languages to file name extensions by means of the `export.convertNpspm.mapping` parameter in the [export.xml](#) configuration file. For the reason mentioned above, no other languages than `velocity` and `jsp` can be specified as server-side languages.

Please always use the `npspm` elements supplied for access control and personalization purposes, not their Velocity or JSP equivalents into which they have been translated. This ensures that always properly translated code is used, independently of the Fiona version you are using.

The different access control functions of the `npspm` element are accessible by means of tag attributes.

3.1.1 includePortlet

Task

This element includes a portlet into an HTML page. If required, you can use an instance identifier to generate different instances of the portlet. If, for example, you would like to include a voting portlet, one instance is sufficient and there is no need to specify the instance name even if the portlet is embedded into several or all the pages of your site. However, if the portlet is used to let users rate individual pages, an individual instance name must be specified for each page. In this case, the path of the page might be used as the instance name because it is unique. The path can be determined by means of an @ reference (see the example below).

Syntax

```
<npspm includePortlet="urlPath" instance="instanceId" language="lang"
  withBorder="borderFlag" />
```

If the portlet to be included is part of the same web application as the Portal Manager, `urlPath` is the name of the portlet or the alias path that has been specified as `portletPathMapping` in the `pm.xml` file. This name must not be `/`. However, if the portlet is part of a different web application, `urlPath` is the URL path of the portlet relative to the directory containing the web applications (normally `webapps`).

By means of the `instanceId` attribute an instance identifier can be specified. This makes it possible to use the portlet several times on the same website or even on the same page. The identifiers may consist of any characters.

The language in which a portlet displays itself normally corresponds to the language the user has specified in his browser settings. The `language` parameter serves to override this setting.

The value of `language`, `lang`, is a code such as `en` or `fr` standing for one of the languages supported by the portlet. These languages are defined in the `portlet.xml` file which can be found in the `WEB-INF` directory of the portlet web application.

As `borderFlag`, the value of `withBorder`, `true` or `false` can be specified to determine whether the portlet is to be displayed including a frame and a title bar with buttons.

Example

```
<npspm includePortlet="/myportlets/ranking" instance="@visiblePath" withBorder="false" />
```

3.1.2 includePage

Task

This element is replaced with the parsed contents of the referenced page if the current user is permitted to access the page. Otherwise, the element is ignored. To avoid endless recursions resulting from cyclic insertions, the maximum nesting depth is limited to 100.

Syntax

```
<npspm includePage="path" />
```

As *path* an internal path needs to be specified. This path is required to point to a file of the document or publication (folder) type. The element has no content.

If the page with the specified path contains internal links, these links might no longer point to the desired target after the page has been included. The reason for this is that paths in links are relative. If, for example, page C is included in A and B, and B is located in a different folder than A, C would have to be relative to two locations in the folder hierarchy to point to the same location.

This effect can be avoided by using absolute links in included files. However, links with different prefixes are required for the preview and the live server.

This can be solved by means of velocity code, which is evaluated at runtime. The `$document` tool provides the `getUrl` method which adds the required prefix to an absolute path and converts the result into an URL. The following example illustrates this by means of a list of links that are created using a `toclist`. The code that generates the URL is first stored in an export variable whose value is then retrieved by means of an `@` reference as the `href` attribute value:

```
<npsobj list="toclist">
  <npsobj modifyvar="set" name="robustPath">$document.getUrl(
    <npsobj insertvalue="var" name="visiblePath"/>
  )</npsobj>
  <a href="@robustPath"><npsobj insertvalue="var" name="title"/></a>
</npsobj>
```

This code causes the absolute Path of the link target (which refers to the folder hierarchy) to be retrieved at runtime and complemented with code which converts the path to the correct URL.

Example

```
<npspm includePage="/intranet/docs/public/apps" />
```

3.1.3 showIfAccessible

Aufgabe

The contents of this element is displayed if and only if the user logged into the portal (or the default user if nobody is logged in) has been granted the live server read permission (`permissionLiveServerRead`) for the page referenced by *path*. By means of `negate="true"` this condition can be reversed, meaning that the page is only displayed if the user has not been granted this permission.

Syntax

```
<npspm showIfAccessible="path" [negate="negateFlag"]>content</npspm>
```

As *path* please specify the path to a CMS file, i.e. an internal path. For *negateFlag* the values `true`, `yes`, `on`, `1` (for `true`), and `false`, `no`, `off`, `0` (for `false`) can be specified. The `negate` attribute is optional.

Example

```
<npspm showIfAccessible="/intranet/docs">
```

```
<a href="/intranet/docs">Our documents</a>
</npspm>
<npspm showIfAccessible="/intranet/docs" negate="true">
  You have no permission to access our documents
</npspm>
```

3.1.4 showIfMember

Task

The contents of this element is displayed if and only if the user logged into the portal (or the default user, if nobody is logged in) is a member of at least one of the specified groups. The contents of the element is also displayed if no group has been specified. By means of `negate="true"` this condition can be reversed, meaning that the page is only displayed if groups have been specified and the user is not a member of any of these groups.

Syntax

```
<npspm showIfMember="group1|...|groupN" [negate="negateFlag"]> ... </npspm>
```

As *group1* to *groupN* group names delimited by vertical bar characters can be specified. It is also possible to specify no group at all. *negateFlag* can be set to `true`, `yes`, `on`, `1` (for `true`), or `false`, `no`, `off`, `0` (for `false`). The `negate` attribute is optional.

Example

```
<npspm showIfMember="users|admins">
  You are a member of the "users" or "admins" group or of both groups!
<npspm showIfMember="users" negate="true">
  You are no member of the "users", therefore you must be an admin!
</npspm>
</npspm>
```

3.1.5 showIfLoggedIn

Task

This element causes content to be displayed only if a logged-in user (`showIfLoggedIn="true"`) or anonymous user (`showIfLoggedIn="false"`) requests the page.

Syntax

```
<npspm showIfLoggedIn="knownFlag"> ... </npspm>
```

For *knownFlag* the values `true`, `yes`, `on`, `1` (for `true`) und `false`, `no`, `off`, `0` (for `false`) can be used.

Example

```
<npspm showIfLoggedIn="true">
  You are logged into the portal!
</npspm>
```

3.1.6 showIfLanguage

Task

The `npspm` instruction `showIfLanguage` makes it possible to show or hide parts of the content, depending on the language setting of the user:

Syntax

```
<npspm showIfLanguage="lang"> ... </npspm>
```

For *lang* one of the language codes consisting of two characters (such as `de`, `en`, `fr`, etc.) can be specified.

How the user's language is determined depends on the configuration. If no single language has been associated to the host, the language will be determined either by means of the user's browser settings, an optional attribute of the logged-in user, or the `lang` request parameter. The behaviour can be configured via the `LanguageFilter` in the file `WEB-INF/web.xml`.

Example

```
<npspm showIfLanguage="de">Deutscher Inhalt</npspm>
<npspm showIfLanguage="de" negate="true">English content</npspm>
```

For German-language users "Deutscher Inhalt" is displayed while "English content" is displayed for all other users.

3.1.7 Parameterizing npspm Elements

All `npspm` instructions used in the layout files of the Content Management server except `showIfLoggedIn` can be parameterized. It might be necessary to do this in layout files, in which `npspm` elements are used for the elements of a list which is generated only at runtime (a `toclist` or a list of free links). Parameterizing the `npspm` instructions means using variables in place of known fields of the exported file. Thus, the access permission of the files determined by the list instruction can be checked:

```
<npsobj list="toclist">
  <npspm showIfAccessible="@self"> ... </npspm>
</npsobj>
```

To write HTML text (oder other text) to the output file only if a user is a member of at least one of several user groups, the following `npspm` instruction can be used:

```
<npspm showIfMember="@memberList"> ... </npspm>
```

In the example above, `memberList` represents a multi-selection field that contains group names. As described in [showIfMember](#), the group names may also be specified directly using the vertical bar character as a separator. This is the full sample code with which a list containing permission checks for each element can be generated:

```
<ul>
  <npsobj list="toclist">
```

```

<npspm showIfMember="@memberList">
  <li>
    <npsobj insertvalue="anchor" name="self">
      <npsobj insertvalue="var" name="title" />
    </npsobj>
  </li>
</npspm>
</npsobj>
</ul>

```

The generated list contains only elements the user is permitted to access.

3.1.8 Error Handling with Wrong npspm Tags

If an `npspm` instruction in a document cannot be interpreted because it contains errors (an undefined attribute value, for example), an error message is displayed instead of the document.

Additionally, for making it easy to find the error, the location where the error occurred is written to the log file of the application concerned (see the `log` directory of the application concerned).

3.2 Creating News and News Feeds in the Editorial System

3.2.1 Functions for Creating News Feeds

The following functions for creating news feeds are available in the CMS:

- In the system settings, channels can be defined. Channels serve to group news by topic, i.e. to categorize them with respect to what their contents is about. This is similar to categorizing product descriptions by means of a field named `ProductGroup`. The channel settings can be accessed via the Content Navigator's *System Configuration* menu item available in the *Extras* menu.
- The versions of files of the *Folder* and *Document* type have a built-in field named `channels`. This is a multi-selection field; therefore, any subset of the channels that have been defined in the system configuration can be assigned to it. This is how content and a set of channels are associated with each other.
- File formats have the `canCreateNewsItem` field (*Mark as new on the live server*). If this option has been chosen, files based on this format are added to an internal news list upon release unless the `channels` field of the released version is empty.
- Lists of news article files can be created in which the articles are assigned to any set of channels you determine. For this, an `NPSOBJ` instruction and a `Tcl` command is available. The `NPSOBJ` instruction returns, analogous to the `toclist` instruction, a context list, so that the fields of the version can be queried. The `Tcl` command is [news](#).

3.2.2 Using the npsobj newslst Command

For creating news lists in layouts the `npsobj newslst` command can be used in three ways:

- Creating a list of all news articles in all channels:

```

<npsobj newslst="all" length="20">
  Text evaluated for each news article
</npsobj>

```

- Creating a news list where each article is contained in at least one of several channels assigned to a field of the document being exported:

```
<npsobj newslst="selected" name="Channel-Feld" length="20">
  Text evaluated for each article belonging to the channels specified indirectly
</npsobj>
```

The field must be either of the `string`, `text`, `selection`, or `multiselect` type. With the `string` and `text` types, the channels must be a comma-separated list. With `selection` and `multiselect` fields, the field values are used unmodified as channels.

- Creating a list of news assigned to channels that are specified directly:

```
<npsobj newslst="selected" value="ch1, ch2, ..." length="20">
  Text evaluated for each article belonging to the channels specified directly
</npsobj>
```

Even if a news article has been assigned to more than one of the channels specified directly or indirectly, it is never contained more than once in the news list generated.

Generated news lists only include articles whose publication date lies in the past (relative to the export time of the document containing the list). Of course, the news list internally contains all of the articles to which a channel has been assigned and that were chosen for publication via the `canCreateNewsItem` file format field. Initially, the publication date equals the creation date of the news file. If required, this date can be modified via the `validFrom` field of the draft version.

A news article that was published accidentally can therefore be removed from the news list by specifying a future date as the publication date. However, after this change it is still contained in the internal news list unless the news file itself is deleted or all channels are removed from the `channels` field of the article. A file is also removed from the news list if the channels to which the article is assigned are deleted from the system configuration. However, this does not modify the `channels` field of versions, meaning that it may contain the names of nonexistent channels. News articles assigned to nonexistent channels are not placed into the internal news list again if the missing channel is created again.

If the Template Engine is used, all the files containing an `npsobj newslst` instruction are exported again after changes have been made to files or the channel configuration so that the news lists generated are always up-to-date. (A `usesAll` [dependency](#) is assigned to the files.)

3.2.3 Examples

Creating a list of 10 most recently published articles

- Create a channel named *sitenews*.
- Create a file format named `newsitem` and activate its `canCreateNewsItems` (*Mark as new on the live server*) option.
- Create a file named `news1` based on the `newsitem` format anywhere in the folder hierarchy.
- Into the layout of the start page insert code like the following:


```

<ul>
  <npsobj newslst="all" lenght="10">
    <li>
      <npsobj insertvalue="anchor" destination="self">
        <npsobj insertvalue="var" name="title" />
      </npsobj>
    </npsobj>
  </ul>

```

Creating an RSS feed for politics news

- Create a channel named `politics`;
- Create a file format named `newsitem` and activate its `canCreateNewsItems` (*Mark as new on the live server*) option.
- Create a field named `description` and a file format named `feed` and add `description` to the format. Do not activate `canCreateNewsItems`.
- Create a folder named `rssfeeds`.
- In this folder create a base layout (`mastertemplate`) that outputs an RSS-Feed (see below).
- In the same folder, create a file named `politicsfeed` based on the `feed` format. Enter as `description` "Latest company politics news". As Channels specify `politics`.
- Set the main content of the base layout to the following:

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.92//EN"
  "http://my.netscape.com/publish/formats/rss-0.92.dtd">
<rss version="0.92">
<channel>
  <description>
    <npsobj insertvalue="var" name="description" />
  </description>
  <language>de</language>
  <title><npsobj insertvalue="var" name="title" /></title>
  <link>http://www.mysite.com/</link>
  <copyright>Copyright by Infopark AG</copyright>
  <generator>NPS 6.0</generator>
  <ttl>60</ttl>
  <npsobj newslst="selected" name="channels" length="20">
    <item>
      <title><npsobj insertvalue="var" name="title"/></title>
      <link>http://www.infopark.de<npsobj
        insertvalue="var" name="visiblePath" />
      </link>
      <description>
        <npsobj insertvalue="var" name="description"/>
      </description>
    </item>
  </npsobj>
</channel>

```

```
</rss>
```

For creating an RSS file in a different version (2.0, for example), the layout file can be adapted as desired.

To create another feed, sports news, by means of the layout file above:

- Create a channel named `sports`.
- In the folder `rssfeeds` create a file named `sportsfeed` based on the `feed` format and enter "Sports News" into its `description` field. Select `sports` as channels.
- Now create files based on the `newsitem` format and assign to them the `sports` channel. They will appear in the sports news feed.

Sending out Newsletters

- Create a folder named `newsletters`.
- Write a base layout that generates the contents of the news letter by, for example, reading out fields of the files concerned.
- Create a file folder named `newsletter`.
- In the folder `newsletters` create a file named `politics` based on the `newsletters` format.
- Into the main content of `politics` enter the e-mail addresses, on address per line.
- Write a that sends the `exportBlob` field of `politics` to each e-mail address contained in the main content of `politics`.

4

4 Installing and Configuring the Portal Manager

4.1 Using the Portal Manager with another Application Server

During the installation of Infopark CMS Fiona, the Trifork Application Server is installed as well. Next to other applications, the supplied GUI and the Infopark Portal Manager are running in this application server. Therefore, no other third-party software is required for operating these web applications.

The Portal Manager and the portlets supplied by Infopark – however, not the GUI – can also be operated in a different application server, if the following prerequisites are met.

1. The application server needs to fully support J2EE 1.4.
2. The application server needs to support the optional feature *cross-context dispatching* and it must have been activated.

4.1.1 Using the Portal Manager with Tomcat 5.x

Please proceed as described in the following instructions if you wish to operate the Infopark Portal Manager in conjunction with the Tomcat Application Server, version 5.5. For Tomcat 5.0, the procedure is similar. However, you require the installation package for Tomcat 5.0. It is not necessary to install the compatibility package for using the JDK 1.4.

1. Download Tomcat
 1. Download the `apache-tomcat-5.5.17.tar.gz` package from the [Apache Tomcat website](#).
 2. If Tomcat is to be operated with the JDK 1.4, please also download the `apache-tomcat-5.5.17-compat.tar.gz` package.
2. Install Tomcat
 1. Unpack the package you downloaded in step 1.1. In the following, the directory created is referred to as `tomcatInstallDir`.
 2. If Tomcat is operated using the JDK 1.4, please also unpack the package you downloaded in step 1.2 and use the same target directory as for the Tomcat package because additional jar files are placed into the `tomcatInstallDir/common/lib` directory.
3. Adapt the `PATH` environment variable so that the JDK to be used can be found.
4. If on the same machine a Trifork server is installed which uses the standard ports, adapt three ports in the `tomcatInstallDir/conf/server.xml` file:

```
<Connector port="8080" maxHttpHeaderSize="8192"
  maxThreads="150" minSpareThreads="25"
  maxSpareThreads="75"
  enableLookups="false" redirectPort="8443"
```

```

acceptCount="100"
connectionTimeout="20000" disableUploadTimeout="true" />

<Connector port="8009"
  enableLookups="false" redirectPort="8443"
  protocol="AJP/1.3" />

<Connector port="8082"
  maxThreads="150" minSpareThreads="25"
  maxSpareThreads="75"
  enableLookups="false" acceptCount="100"
  connectionTimeout="20000"
  proxyPort="80" disableUploadTimeout="true" />

```

5. Copy the [JavaBeans™ Activation Framework](#) from *fionaInstallDir/share/lib/activation.jar* to *tomcatInstallDir/common/lib*.
6. Copy [JavaMail](#) from *fionaInstallDir/share/lib/mail.jar* to *tomcatInstallDir/common/lib*.
7. Recursively copy the PM and PM-PL web applications from the *fionaInstallDir/instance/myInstance/webapps* directory to *tomcatInstallDir/webapps/*.
8. In the files *tomcatInstallDir/webapps/PM/WEB-INF/web.xml* and *tomcatInstallDir/webapps/PM-PL/WEB-INF/web.xml*, adapt the path to the *license.xml* file contained in the *config* directory of your CMS instance. If the CMS is running on a different machine, also copy the licence file to a location where Tomcat can find it.
9. In the *tomcatInstallDir/webapps/PM/WEB-INF/logging.xml* and *tomcatInstallDir/webapps/PM-PL/WEB-INF/logging.xml* files, enter the path to the directory where the log files of the web applications are located. Example:

```
<param name="File" value="tomcatInstallDir/logs/PM.log"/>
```

10. Delete the file *trifork-app-conf.xml* from the *META-INF* directories of both web applications, PM and PM-PL. Into each directory place a version of the file *context.xml*, adapted to the respective web application. For PM:

```
<Context docBase="${catalina.home}/webapps/PM" path="/PM" crossContext="true" />
```

For PM-PL:

```
<Context docBase="${catalina.home}/webapps/PM-PL" path="/PM-PL" crossContext="true" />
```

11. Prepare PM and PM-PL for live operation:

1. If you use version 6.0.x of the Portal Manager, please deactivate the *com.infopark.pm.PreviewDocumentSource* bean in the *documentManager* bean contained in the *tomcatInstallDir/webapps/PM/WEB-INF/pm.xml* file (by commenting it out)
2. Activate the *com.infopark.pm.FileDocumentSource* bean (by removing the comment characters). In the *documentRoot* property, specify the path to the export directory.
3. In the *userManager* bean contained in the *tomcatInstallDir/webapps/PM/WEB-INF/pm.xml* file, activate the user manager bean you wish to use and configure it. Deactivate the other user managers.
4. In the *searchEngine* bean contained in the *tomcatInstallDir/webapps/PM-PL/WEB-INF/pm.xml* file, deactivate the bean in the

`com.infopark.libs.search.cm.AdvancedCmSearchEngine` class and activate this bean in the `com.infopark.libs.search.ses.SesSearchEngine` class. Adapt the host and port properties so that they match the Search Server ports of the CMS instance (see the `server.xml` file in the `config` directory of the instance).

5. Change the `portletPathMapping` specified in the `tomcatInstallDir/webapps/PM-PL/WEB-INF/pm.xml` file to `/PM-PL = /PM-PL`.

12. Start the Tomcat server by running `tomcatInstallDir/bin/startup.sh`. You can now open the following URL to access your exported pages containing portlets:

```
http://myTomcatHost:myTomcatPort/PM/
```

13. Should problems arise, please look into the log files contained in the `tomcatInstallDir/logs` directory. With general errors, the `catalina.out` and `catalina.yyyy-mm-dd.log` files are instructive. With application-specific errors, please examine the files configured in step 9 above.

4.1.2 Using the Portal Manager with WebSphere Application Server 5.1

Please proceed as follows to use the Infopark Portal Manager with WebSphere Application Server 5.1.

1. Copy the license file `license.xml` to `webapps/PM/WEB-INF`.
2. In the `webapps/PM/WEB-INF/web.xml` configuration file set the value of the `licenseFile` context parameter to `/WEB-INF/license.xml`.
3. Store the contents of the `webapps/PM` directory in a ZIP file and name it `PM.war`.
4. Deploy the `PM.war` archive by means of the *WebSphere Administrative Console* into `/PM`.

Please repeat the steps above analogously for `webapps/PM-PL`.

4.2 Standard Configuration

4.2.1 Servlet and Filter Definition

The Portal Manager can be configured by means of the two files `web.xml` and `pm.xml`. These files can be found in the `WEB-INF` directory located below the web application directory of the Portal Managers. In the CMS Standard installation, the path is (relative to the CMS directory):

```
instance/default/webapps/PM/WEB-INF
```

Just as in other web applications, the `web.xml` file serves to declare the servlets, servlet filters etc. used and to specify their URL schemas.

4.2.2 Portal Manager Configuration

The current configuration of the Portal Manager can be found in the `pm.xml` file. This file is a standard [Spring bean](#) file. In the standard configuration the following beans exist:

- `hostConfig` (`com.infopark.pm.HostConfig`)
Determines the hosts for which requests are accepted as well as the languages supported. See also [<npspm showIfLanguage>](#) and the portlet configuration in the `WEB-INF/portlet.xml` file.
- `userManager` (`com.infopark.pm.user.UserManager`)

The user manager of the system, queried by the `AuthenticationFilter` for accessing user data.

- `portletContainer` (`com.infopark.pm.portlet.PortletContainer`)
Manages the portlets defined in the file `WEB-INF/portlet.xml` and makes them accessible for handling dynamic content.
- `documentManager` (`com.infopark.pm.DocumentManager`)
Provides the content for the `ContentServlet`. For this, the supplied implementation makes use of a configurable `com.infopark.pm.DocumentSource`.
- `permissionManager` (`com.infopark.pm.PermissionManager`)
Provides the `PermissionFilter` with the names of the groups whose users are permitted to access content to be delivered. From version 6.7.1, the `permissionManager` does no longer exist since the `AuthorizationManager` covers its functions.
- `authorizationManager` (`com.infopark.pm.user.AuthorizationManager`)
Uses the specified methods (`com.infopark.pm.user.Authorizer`) to check whether the user may access content.
- `templateEngine` (`com.infopark.pm.TemplateEngine`)
Prepares dynamic content so that the `ContentServlet` can compute the resulting document.
- `contentHandlerMap` (`java.util.Map`)
Determines which content is to be delivered by the `ContentServlet` and how this is done (dynamically or statically). For each MIME type a bean of the `com.infopark.pm.doc.ContentHandler` type can be specified. Individual implementations are used for static and dynamic content.

4.2.3 Configuring a Portlet Web Application

The Portlets contained in a web application can be configured by means of the `WEB-INF/portlet.xml` file which has the following structure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<portlet-app version="1.0"
  xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
    http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd">
  <portlet>
    ...
  </portlet>
  ...
  <custom-portlet-mode>
    ...
  </custom-portlet-mode>
  ...
  <custom-window-state>
    ...
  </custom-window-state>
  ...
  <user-attribute>
    ...
  </user-attribute>
  ...
</portlet-app>
```

The elements have the following meaning:

- `portlet` configures a portlet
- `custom-portlet-mode` configures a non-standard portlet mode such as `about` or `print`.
- `custom-window-state` configures a non-standard window state.

- `user-attribute` defines optional [user properties](#) made available to the portlets.

Security constraints (`security-constraint`) are not supported by the Infopark Portal Manager.

4.2.4 Configuring a Portlet

Every portlet can be configured by means of a `portlet` element. The following example includes the most important elements:

```
...
<portlet>
  <portlet-name>example</portlet-name>
  <portlet-class>com.infopark.example.Portlet</portlet-class>
  <init-param>
    <description>my init param description</description>
    <name>host</name>
    <value>127.0.0.1</value>
  </init-param>
  <expiration-cache>0</expiration-cache>
  <supports>
    <mime-type>text/html</mime-type>
    <portlet-mode>edit</portlet-mode>
  </supports>
  <supported-locale>en</supported-locale>
  <supported-locale>de</supported-locale>
  <resource-bundle>com.infopark.example.localizer</resource-bundle>
  <portlet-preferences>
    <preference>
      <name>readOnlyPreference</name>
      <value>foo</value>
      <value>bar</value>
      <read-only>true</read-only>
    </preference>
    <preferences-validator>com.infopark.example.Validator</preferences-validator>
  </portlet-preferences>
</portlet>
...
```

The elements have the following meaning:

- `portlet-name`: The portlet identifier. The identifier must be unique in the web application. It is used to [include the portlet](#).
- `portlet-class`: The Java class that provides the portlet functionality. This class must be contained in a jar archive in the `WEB-INF/lib` directory or as a compiled class in `WEB-INF/classes`.
- `expiration-cache`: This option allows the Portal Manager to cache the portlet contents for the number of seconds specified, meaning that the content is not recomputed during this period of time. 0 deactivates this function, -1 permits unlimited caching (until an action is performed).
- `init-param`: A configuration parameter for this portlet. Any number of `init-param` elements may be specified for a portlet.
 - `description`: Optional description
 - `name`: The parameter name
 - `description`: The value of the parameter
- `supports`: Configures supported MIME types and portlet modes:
 - `mime-type`: Configures a supported MIME type. For each type, an individual element needs to be specified, the MIME type is obligatory.
 - `portlet-mode`: Other portlet modes than `VIEW` can be configured here.

- **supported-locale:** A supported language. Typically, only an ISO language code is specified here, the country code, separated by an underscore character is optional. A portlet can support one or more languages, i.e. the element can be specified more than once.
- **resource-bundle:** The name under which the resources can be found (without the language or the `.properties` suffix).
- **portlet-info:** As an alternative to resources, a title, a short title, and keywords can be specified directly using this element. The corresponding subelements are: `title`, `short-title`, and `keywords`.
- **portlet-preferences:** For every portlet, any number of presets can be configured. The values of these settings can be modified at runtime unless the setting is flagged as read-only. Additionally, a validator class can be specified.

The `expiration-cache` option is supported from version 6.5.2 of the CMS. With computationally intensive portlets, this option may improve the performance of the portal.

4.2.5 Portlet Modes and Window States

From version 6.5.1, you can use your own portlet modes and window states (not defined in the JSR168 specification). To do this, please create the file `portlet.xml` in the *WEB-INF* directory of the PM web application and declare your custom portlet modes and windows states in it.

Example:

```
...
<custom-portlet-mode>
  <portlet-mode>preview</portlet-mode>
</custom-portlet-mode>
<custom-portlet-mode>
  <portlet-mode>urn:javax.portlet:mode:custom:about</portlet-mode>
</custom-portlet-mode>
<custom-window-state>
  <window-state>solo</window-state>
</custom-window-state>
...
```

4.3 User Management in the Portal

4.3.1 Using the Portal Manager with LDAP or ADS

To connect the Portal Manager to an ADS server, please configure the `AdsUserDirectory` bean in the `pm.xml` file located in the Portal Manager web application.

Please adapt the properties of the `userDirectory` as needed. The individual parameters are described in the configuration file.

4.3.2 Providing User Properties in Portlets

From version 6.5.1 of the Infopark CMS, user properties provided, for example, by a [directory service](#) can be accessed in a portlet by means of the `javax.portlet.PortletRequest.USER_INFO` request attribute.

The attributes provided by the portal are listed in the `portlet.xml` file. In the following example, this is shown for two attributes:

```
<user-attribute>
  <name>user.name.real</name>
</user-attribute>
<user-attribute>
  <name>user.business-info.online.email</name>
</user-attribute>
```

However, in most cases these attributes will not be provided under their standard names. Therefore, the Infopark Portal Manager offers the possibility to map standard names to the names under which they are made accessible. For this, the `userAttributeMapper` property in the `portletContainer` bean defined in the `pm.xml` file needs to be set.

This is shown in the following example for mapping the attributes above to two attributes that also exist in the Content Manager:

```
<property name="userAttributeMapper">
  <bean class="com.infopark.pm.user.SimpleAttributeMapper">
    <property name="mapping">
      <value>
        user.business-info.online.email = email
        user.name.real = realName
      </value>
    </property>
  </bean>
</property>
```

By means of this configuration, a portlet can access the `email` user property using the following code:

```
Map userInfo = (Map)request.getAttribute(PortletRequest.USER_INFO);
String email = (String)userInfo.get("user.business-info.online.email");
```

4.3.3 Retrieving User Properties from the Directory Service

With Infopark CMS Fiona from version 6.5.0, further user properties maintained in an LDAP or ADS directory service can be retrieved and used in the Portal Manager. For this, the attributes to be retrieved need to be added to the `attributes` property of the bean configuration of the directory service. Example:

```
...
<bean class="com.infopark.pm.user.AdsUserDirectory">
  ...
  <property name="attributes">
    <map>
      <entry key="mail">
        <map>
          <entry key="type"><value>single</value></entry>
        </map>
      </entry>
      <entry key="name">
        <map>
          <entry key="type"><value>single</value></entry>
        </map>
      </entry>
    </map>
  </property>
```

```
...
</bean>
...
```

The value of `key` in an `attributes` entry is interpreted as the corresponding name of the attribute in which the user property is stored.

As the `type` of an attribute either `single` (a return value) or `list` (a list of return values) can be specified.

4.4 Using the Portal Manager with Named Virtual Hosts

With Infopark CMS Fiona, several websites can be operated on name-based virtual hosts. The websites are maintained with the CMS as separate instances and are served dynamically by the Portal Manager. It is possible to use the same instance for several virtual hosts.

4.4.1 Prerequisites and Mode of Operation

For operating the virtual hosts, Apache webserver with `mod_jk`, Trifork Application Server, and Infopark Portal Manager are required.

For Apache webserver to accept requests sent to the virtual hosts and redirect them to the Trifork server using `mod_jk`, the following directive in the Apache configuration is used:

```
jkMount /* triforkWorker
```

The Trifork server in turn redirects the requests to the Portal Manager web application which, based on its configuration, determines the content assigned to the virtual host. For this, the `VirtualHostConfig` configuration element is used.

Since only one document root directory exists in the Portal Manager configuration (`documentSource`) for all virtual hosts, it is set to any directory in which then a symbolic link is created for each virtual host. The symbolic links point to the content associated with the corresponding virtual hosts.

4.4.2 Sample Configuration

Apache

```
# /etc/httpd/httpd.conf
...
NameVirtualHost 10.1.110.40:80
NameVirtualHost 10.1.110.40:443
<VirtualHost 10.1.110.40:80>
    ServerName www.infopark.de
    ServerAlias infopark.de
    ErrorLog /var/log/httpd/error_log
    CustomLog /var/log/httpd/www.infopark.de-access_log combined
    CustomLog /var/log/httpd/www.infopark.de-redirect_log redirectlog
    JkMount /* triforkWorker
    ...
</VirtualHost>
<VirtualHost 10.1.110.40:80>
    ServerName www2.iico.de
    ErrorLog /var/log/httpd/error_log
    CustomLog /var/log/httpd/www.iico.de-access_log combined
    JkMount /* triforkWorker
```

```
</VirtualHost>
```

```
# /etc/httpd/workers.properties

worker.list=triforkWorker
# Set properties for triforkWorker (ajp13)
worker.triforkWorker.type=ajp13
worker.triforkWorker.host=localhost
worker.triforkWorker.port=8009
worker.triforkWorker.lbfactor=250
worker.triforkWorker.cachesize=50
worker.triforkWorker.cache_timeout=300
worker.triforkWorker.socket_keepalive=1
worker.triforkWorker.recycle_timeout=300
```

Portal Manager

Here, one Portal Manager is used for all the virtual hosts. However, you might also define several `workers` in the Apache configuration to assign the virtual hosts to any number of Portal Managers. This might be done to improve the performance with Portal Managers running on individual computers.

```
# /opt/Infopark/CMS-Fiona/instance/internet/webapps/PM/WEB-INF/pm.xml
...
<bean id="hostConfig" class="com.infopark.pm.VirtualHostConfig">
  <property name="properties">
    <props>
      <prop key="infopark_de.locales">de</prop>
      <prop key="infopark_de.http.enable">true</prop>
      <prop key="infopark_de.http.host">www2.infopark.de</prop>
      <prop key="infopark_de.http.port">80</prop>
      <prop key="infopark_com.locales">en</prop>
      <prop key="infopark_com.http.enable">true</prop>
      <prop key="infopark_com.http.host">www2.infopark.com</prop>
      <prop key="infopark_com.http.port">80</prop>
      <prop key="iico_de.locales">de</prop>
      <prop key="iico_de.http.enable">true</prop>
      <prop key="iico_de.http.host">www2.iico.de</prop>
      <prop key="iico_de.http.port">80</prop>
    </props>
  </property>
</bean>
...
<bean id="documentSource" class="com.infopark.pm.doc.WebappDocumentSource">
  <property name="documentSource">
    <bean class="com.infopark.pm.FileDocumentSource">
      <property name="documentRoot" value="/var/www/vhosts" />
      <property name="inputEncoding" value="UTF-8" />
    </bean>
  </property>
  <property name="inputEncoding" value="ISO-8859-1" />
</bean>
...
```

Creating Symbolic Links

Afterwards, the symbolic links are created in the directory that has been specified as the `documentSource`.

```
cd /var/www/vhosts
```

```
ln -s /opt/Infopark/CMS-Fiona/instance/internet/export/online/docs infopark_de
ln -s /opt/Infopark/CMS-Fiona/instance/internet/export/online/docs infopark_com
ln -s /opt/Infopark/CMS-Fiona/instance/iico/export/online/docs/iico iico_de
```

Deploying the Portal Manager

Finally, the Portal Manager needs to be deployed again for the configuration changes to come into effect. For this, run the following command from the instance directory concerned:

```
./instance/instancename/bin/rc.npsd deploy PM
```

4.5 Providing Portlets via WSRP

From version 6.5.1 of Infopark CMS Fiona, the Infopark Portal Manager is able to provide remote portlets from a [WSRP producer](#). This can be done by means of a virtual context path that is specified in the Portal Manager. Such a virtual context path refers by means of an URL to a WSDL file (WSDL = Web Service Description Language) which describes how the service can be accessed.

Since remote portlets are made available in this way, they behave like portlets of the Infopark Portal Manager and can be addressed using normal `npspm`-Syntax.

To make WSRP portlets available, please extend the `ProxyPortletContainer` bean located in the `webapps/PM/WEB-INF/pm.xml` configuration file of the CMS instance concerned. Please add the `wsrpPathMapping` property and a value subelement to the bean configuration. As the value of the value element specify for each WSRP producer a line containing the virtual context path and the WSDL URL to be used. Example:

```
/WS-ORA = http://portalstandards.oracle.com/portletapp/portlets?WSDL
```

If the URL points to a WSDL file which contains references to external sources not available for some reason (firewall access restrictions, for example), please [download the WSDL file](#) and make the reference point to its path in the local file system using `file:///`. Example:

```
/WS-IBM = file:///var/temp/portalserver-ibm.wsdl
```

The following is a sample excerpt from the `pm.xml` file:

```
<bean id="portletContainer" class="com.infopark.pm.portlet.ProxyPortletContainer">
  <property name="preferencesStorage">
    <bean class="com.infopark.pm.FilesystemPreferencesStorage" />
  </property>
  <property name="cacheManager" ref="cacheManager" />
  <property name="wsrpPathMapping">
    <value>
      /WS-ORA = http://portalstandards.oracle.com/portletapp/portlets?WSDL
      /WS-IBM = file:///var/temp/portalserver-ibm.wsdl
    </value>
  </property>
  <property name="webserviceRefreshInterval" value="0" />
</bean>
```

Changes to the configuration must be completed by restarting the Application Server. If the Portal Manager and the Producer are able to communicate with each other, the handles of the portlets can then be retrieved using the following URL:

`http://my.server:8080/PM/debug/`

Once a portlet has been identified, it can be embedded into the content analogously to the following example, whereby *portletHandle* refers to the handle of the portlet to be included:

```
<npspm includeportlet="/WS-ORA/portletHandle" />
```

Other Properties

- `webserviceRefreshInterval`: The interval for updating the list of the remote portlets (in seconds).

4.6 Making WSDL files Available

The WSDL file describing the WSRP service normally contains references to schemes or name spaces located on internet servers, for example `http://www.oasis-open.org/committees/wsrp/specifications/version1/wsrp_v1_bindings.wsdl`, `http://www.w3.org/XML/1998/namespace`). If your network configuration prevents the Infopark Portal Manager to connect via HTTP/HTTPS to servers outside the intranet or the DMZ, the targets of the references need to be made available differently because the consumer requires the complete description of the remote services. For this, the following methods can be used:

4.6.1 Using a Proxy Server

If your network policy permits the Infopark Portal Manager to connect to the internet *and* to the producer via a proxy server, you might want to start the Application Server running the Portal Manager with the `proxyHost` and `proxyPort` Java options.

If you use the Trifork Server, these options can be added to the `rc.npsd.conf` configuration file located in the `bin` directory of the CMS instance concerned. Example:

```
set conf(triforkArgs) [list server start -devel -vmargs=-server\
  -vmargs=-Xmx256m -vmargs=-Xms256m\
  -vmargs=-XX:MaxPermSize=128m -DproxyHost=mein.server \
  -DproxyPort=3128]
```

Please note that this permits all applications to connect to the internet via the proxy server.

4.6.2 Making Included Files Locally Available at the Producer

Configure your producer in such a way that it only includes XML files available locally, or adapt the WSDL file accordingly. The references in the WSDL file can then be resolved by means of requests to the producer.

4.6.3 Making Included files Locally Available at the Consumer

If changes cannot be made to the producer, the WSDL file and all the XML files it includes can be made available locally. This is the least flexible method because changes to the producer always need to be reconstructed locally.

To do this, please use a browser to get the WSDL file via its URL and save it to your local file system. Then edit this file and replace all `imports` from external web servers with references to local files,

meaning that you need to download all the external references. Repeat this for all downloaded files until all external references are available locally and are included instead of their remote counterparts. Place all the files into a single directory of the application server and configure this producer's `wsrpPathMapping` in the `pm.xml` as a local WSDL file.

5

5 Portlets

This section describes the function and configuration of portlets that can be used in conjunction with the Infopark Portal Manager.

If not stated otherwise in the individual portlet descriptions, a separate licence is required for using the portlets (except for the login portlet) described in this document and its subdocuments.

5.1 News Portlet

Many companies provide latest news as RSS news feeds. RSS (really simple syndication) is a standardized method for distributing information. An RSS feed is an XML document which contains structured data (for example a title, a summary, and a link) so that it can be processed by computers.

With Infopark CMS Fiona, such feeds can be generated automatically in such a way that the access permissions of the portal users are taken into account when the feeds are delivered by the Portal Manager. This means that the news feeds can be personalized. All RSS versions (0.90, 0.91, 0.92, 0.93, 0.94, 1.0, and 2.0) are supported.

For logged-in portal users, the portlet marks the articles that have already been read. In the overview, these articles can then be hidden or displayed in a simplified manner. For the formats RSS 0.93, 0.94, and 2.0 and Atom 0.3 and 1.0, Infopark CMS Fiona 6.7.0 and later also compares the publication dates of the articles with the dates at which the user has read them. This is done for the purpose of marking updated articles as unread again.

Using this portlet requires a separate license.

5.1.1 Operation

The news portlet has two modes, viewing (*view*) and editing (*edit*). You can switch from one mode to the other by means of a button in the title bar. In the view mode, the portlet displays the articles of the (*items*) of the currently selected news feed. If several news feeds have been configured, you can use a selector to choose one:



Like a bookmark, each news feed consists of a name and the news feed URL. In editing mode, this configuration data can be changed and news feeds can be resorted, added, or deleted. For this, the portlet displays a list of the feeds:



You can only switch to the editing mode, if the portlet is displayed with a border around it. The following screenshot illustrates how the title and the URL can be edited.



5.1.2 Configuration

The portlet is named `news` and its configuration can be found in the `portlet.xml` file. Three optional initialization parameters exist:

- `updateInterval` defines the time in seconds after which the news feed is fetched again.
- `maxItems` defines the maximum number of articles per user for which the portlet stores the *read/unread* state. Please choose a value twice the number of articles in your news feed. Otherwise news may reappear as unread although the portlet user has read them.

- `maxLength` defines the maximum text length of an article description. If the length of a description exceeds this value, it will be trimmed and '...' is appended to it.

Example for Setting Initialization Parameters

```
<init-param>
  <name>updateInterval</name>
  <value>5</value>
</init-param>
<init-param>
  <name>maxItems</name>
  <value>5</value>
</init-param>
<init-param>
  <name>maxLength</name>
  <value>160</value>
</init-param>
```

Furthermore, a setting exists, `feeds` with which any number of feeds can be preset for new users. Each feed definition consists of a name and the URL of the feed, separated by the `|` character. Therefore, neither the name nor the URL must contain this character. Example:

```
<portlet-preferences>
  <preference>
    <name>feeds</name>
    <value>
      Infopark News|http://www.infopark.de/rss
    </value>
    <value>
      Slashdot News|http://slashdot.org/index.rss
    </value>
    <value>
      Yahoo! News|http://rss.news.yahoo.com/rss/topstories
    </value>
  </preference>
</portlet-preferences>
```

5.1.3 Usage

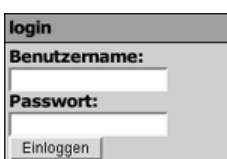
By means of the following code the portlet can be placed into layout files:

```
<npspm includePortlet="/PM-PL/news" ... />
```

If the portlet is located in another web application, please specify its name instead of `PM-PL`.

5.2 Login Portlet

The login portlet allows the portal users to log into the portal:



Furthermore, the portlet allows logging off from the portal:

login
Benutzername: test
<input type="button" value="Ausloggen"/>

5.2.1 Configuration

The users are authenticated by means of the user manager configured in the `/instance/instName/webapps/PM/WEB-INF/pm.xml` file.

5.2.2 Usage

The portlet can be placed into layout files by means of the following instruction:

```
<npspm includePortlet="/PM-PL/login" ... />
```

5.3 Portlet for Editing the Current Page

The edit-page portlet allows the user to edit the document currently displayed in the portal by means of the editorial system, i.e. the Content Navigator.

Using this portlet requires a separate license.

5.3.1 Operation

editPage
Diese Seite mit Infopark CMS Fiona bearbeiten

Clicking the link displayed by the portlet causes a Content Navigator page to be opened in which the corresponding document is selected. If the user is logged into the portal, the portlet tries to log the user into the Content Navigator using the user's user name and password. If this fails, the log-in page is displayed.

This portlet is very useful if the portal users also work as editors in the CMS.

5.3.2 Configuration

As with all other portlets, the layout of this portlet is defined by means of Velocity templates located in the `/WEB-INF/templates` directory of the PM web application. The layout can be adapted as desired.

5.3.3 Usage

The portlet can be placed into layout files by means of the following instruction:

```
<npspm includePortlet="/PM-IP/editPage" ... />
```

5.4 Form Portlet

The form portlet offers a simple way of providing a sequence of form pages.

Using this portlet requires a separate licence.

By means of the corresponding buttons, the user can move to the previous or next page of the sequence of forms. When the last form is submitted, normally an action is performed. For example, the form data can be sent via e-mail.

The forms and their sequence as well as the action to be performed at the end, are defined in an XML file. The dynamic display of the forms on the web page can be controlled by means of velocity templates.

Each sequence is defined in an individual directory below `WEB-INF/templates/flow`. The portlet can be integrated into the content by specifying the desired sequence definition directory. The definition always includes an XML file, `flow.xml`, in which the form pages and their fields are specified. Also, for each form page, a Velocity template used to generate the form's HTML code needs to be placed into the definition directory. These templates are referenced by the XML file. Furthermore, the definition includes the action mentioned above. This action is implemented by a Java class. For the purpose of illustrating a typical use case, the sample files for sending the form data via e-mail are included.

5.4.1 Configuration

Form Definitions

The form pages are defined in the file `flow.xml`. The root element of this file is always `flow`. Optionally, this element can have an attribute named `result` which defines the result template. If `result` has not been specified, `result.vm` will be used. Inside `flow`, each form page is defined by means of a `form` element, and the final action is specified using an `action` element.

Each `form` element defines exactly one form page. By means of its `template` attribute, the name of the Velocity template is specified that serves to display this form page. This template file must be located in the same folder as the `flow.xml` definition file. An example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<flow>
  <form template="contact.vm">
    <field name="gender" type="select"
      required="true" flags="menu">
      <item default="true">female</item>
      <item>male</item>
    </field>
    <field name="firstName" type="text"
      regex="([\p{Lu}][\p{Ll}].-]* *)+" error="errorInvalidName" />
    <field name="lastName" type="text" required="true"
      regex="([- \p{Lu}\p{Ll}]+ *)+" error="errorInvalidName" />
    <field name="email" type="text"
      required="true" validator="email" />
    <field name="subject" type="text" required="true" />
    <field name="message" type="text"
      required="true" flags="area" />
  </form>
  <action name="sendEmail">
    <param name="receiver">playland@infopark.de</param>
    <param name="template">email.vm</param>
```

```
<param name="senderField">email</param>
<param name="subjectField">subject</param>
</action>
</flow>
```

A `form` element may include any number of `field` elements. Each of them represents a field of the form page concerned. `field` elements can have the following attributes:

- `name`: The name of the field which must be unique on the page.
- `type`: The field type. The permitted types are defined in the `WEB-INF/flow.properties` file:
 - `text`: simple text
 - `select`: selection
 - `multiselect`: multiple selection
 - `file`: a file to be uploaded
 - `boolean`: a boolean value
 - `date`: a date
 - `month`: month
- `required`: specifies whether the field is obligatory.
- `flags`: a comma-separated list of display properties.
- `value`: a preset value for the `text` type
- `validator`: a test for the value. Permitted tests are defined in the `WEB-INF/flow.properties` file.
- `regex`: a regular expression for testing the value of `text` fields
- `error`: the error which is output if the `regex` test fails, otherwise `errorInvalid`.

The `name` and `type` attributes are obligatory. You can specify either the `validator` or `regex` attribute, not both. For the `select` and `multiselect` types, the available values can be specified by means of `item` elements in the `field` element. For `select` and `multiselect` the preselected value or values, respectively, can be defined by means of the `default="true"` attribute of the `item` element concerned.

If the `item` element has a `value` attribute, its value is used as field value if the user has selected the element. The contents of the element, on the other hand, serves as display value. If `value` has not been specified, the localized contents of the `item` element is used as display value while the contents itself is used as the actual field value.

The final action needs to be defined by means of an `action` element. Its obligatory `name` attribute determines the action; the permitted names are again defined in the `WEB-INF/flow.properties` file. As for the tests, the action references a Java class which implements the action. The arguments to be passed to the Java code can be defined by means of `param` elements inside the `action` element. Each `param` element needs to have a unique name which can be specified as the value of its `name` attribute. The contents of such an element is the argument value.

Displaying the Forms with Velocity Templates

Each form page is displayed by means of a Velocity template. In the current context the following variables for accessing dynamic content are available:

- `$fields`: all fields of the form page via their name
- `$errors`: the list of all the errors that occurred

- `$text`: localizations (see below)
- `$page`: the number of the current page, beginning with 1
- `$pages`: the total number of pages
- `$action`: the form URL

For displaying the fields, several macros are available, originating from the `WEB-INF/templates/flow/macros.vm` file:

- `#renderLabel`: displays a localized field title
- `#renderField`: displays the fields in accordance with the type and the flags (see above)
- `#renderButtons`: displays a list of buttons
- `#renderErrors`: displays the errors that occurred

How the fields are displayed can be influenced by flags in the following way:

- `area (text)`: multi-line text input
- `password (text)`: invisible (hidden) text input
- `sorted (select, multiselect)`: sort entries alphabetically
- `menu (select, multiselect)`: selection from a drop-down menu

The following example template displays a contact form:

```
<form action="$action" method="post" class="contact">
  #renderErrors()
  <table cellpadding="0" cellspacing="0" border="0">
    <tr>
      <td class="contact">#renderLabel($fields.gender)
        <div>#renderSelectField($fields.gender)</div>
      </td>
      <td class="contact">#renderLabel($fields.lastName)<br>
        #renderTextField($fields.lastName)
      </td>
      <td class="contact">#renderLabel($fields.firstName)<br>
        #renderTextField($fields.firstName)
      </td>
    </tr>
    <tr>
      <td class="contact" colspan="3">
        #renderLabel($fields.email)<br>
        #renderTextField($fields.email)
      </td>
    </tr>
    <tr>
      <td class="contact" colspan="3">
        #renderLabel($fields.subject)<br>
        #renderTextField($fields.subject)
      </td>
    </tr>
    <tr>
      <td class="contact" colspan="3">
        #renderLabel($fields.message)<br>
        #renderTextField($fields.message)
      </td>
    </tr>
    <tr>
      <td class="contact" colspan="3">
        #renderButtons(["Ok"])
      </td>
    </tr>
  </table>
</form>
```

Please also take a look at the [description of the Velocity syntax](#).

Localisation

In the form definition folder the localization for field titles, buttons, errors, and other texts are stored. You can access the localized strings inside templates by means of the `text` context variable. The localizations are stored in the directory as a „Java resource bundle“ named `localizer`. It consists of one file for each language. This file is named `localizer_Language.properties`, with *Language* standing for a two-character language code such as `en` or `de`.

Buttons are normally prefixed with `button`, errors have the prefix `error`, and field titles are prefixed with the name of the corresponding field. Optionally, additional messages, for example for the action result, can be added.

5.4.2 Usage

The portlet can be included in the content by means of the following code:

```
<npspm includePortlet="flow" instance="flowname" />
```

`flowname` stands for the name of the directory containing the form sequence definitions.

When the portlet is executed it displays the first form page inside the portlet. Switches are used for navigating. Typically, these switches are displayed by means of `#renderButtons`. The meaning of the following named switches is predefined:

- **Cancel:** aborts the form sequence, clears all data, and returns to the first page.
- **Back:** jumps to the previous page if it exists. Entered data is preserved.

All other switches have the function of *Continue* oder *OK*. If such a switch is used on the last form page, the defined action is performed. If no error occurs during this action, the result is displayed by means of the result template.

If a field or the action itself causes an error, the current page is not left but displayed again including the errors. The errors should be displayed at a proper place on the page using the `#renderErrors` macro.

Example: Sending E-mail Messages

As a typical use case, an action for sending the form data as an e-mail message has already been implemented. The action `sendEmail` can be adapted to different requirements. For this, several arguments are available in the `action` element (see the example definition above):

- **receiver:** the recipient or recipients of the message. If this argument is missing, the e-mail is sent to the sender.
- **template:** the name of the e-mail template.
- **sender / senderField:** the sender or the field from which the sender is taken. Please note that the sender must be accepted by the mail server!
- **subject / subjectField:** the subject or the field, respectively, from which the subject of the message is taken.

Example: Sending E-Mail Messages and a Confirmation Message

In addition to the `sendEmail` action, `sendEmailAndConfirmation` has been implemented. This action can be used to send a confirmation message in addition to the e-mail message itself.

In the action element, `sendEmailAndConfirmation` has the same parameters as `sendEmail`, plus the following ones:

- `confirmationReceiverField`: the field from which the recipient of the confirmation message is taken.
- `confirmationTemplate`: the name of the template used for the confirmation message.
- `confirmationSubjectField`: the field from which the subject of the confirmation message is taken.

Typically, the `#renderValue` and (from version 6.7.2) `#renderHtmlEscapedValue` macros are used to take over the field values into the template.

5.5 Search Portlet

By means of the search portlet, search forms and results lists can be integrated into a website.

Both the function and the layout of the search can be adapted to meet individual needs without having to modify Java code. The portlet supports sending requests to the Content Management Server as well as to the Search Engine Server. It uses the search engine configured in the `searchEngine` entry of the `pm.xml` configuration file.

Each search is represented by an individual directory below the instance-specific web application directory `WEB-INF/templates/search`.

5.5.1 Usage

The search portlet can be included in layout files in the following way:

```
<npspm includeportlet="/PM-PL/search" instance="dealer" language="de" />
```

The instance name, `dealer`, corresponds to the name of the directory containing the search definition. `language` optionally lets you define the language to be used by the portlet. If it is not specified, the language set in the browser is used.

5.5.2 Configuration

Definition of a Search

Any number of searches can be defined. Each of them is defined by means of two Velocity templates located in the web application directory mentioned above. The `view.vm` file determines the layout of the search form and the results list while `config.vm` determines the search query to be executed by the portlet.

Since the portlet is capable of displaying itself in several languages, a localization file, `localizer_xy.properties`, is required for each locale of the Verity search engine. In the file name mentioned, `xy` stands for the language code concerned. In these files, named strings are defined in the following form:

```

title: Dealer
buttonSearch: Search
errorNoResults: Your search query did not return any results.

```

Please note that the localizer files need to be UTF-8-encoded.

Structure of the `config.vm` Configuration File

After the search form generated by the portlet has been sent, the portlet loads and evaluates the `config.vm` Velocity template to generate the search query from the data given in the input elements of the form. The parameters from the search form are available in the template under their names.

The evaluation of the template yields an XML document the Search Server is able to process. The portlet sends this document to the Search Server and receives as response an XML document containing the search results to be displayed.

The XML document generated by the Velocity template is required to have the following structure:

```

<query>
  <condition>...</condition>
  <result-fields>...</result-fields>
  <sort>...</sort>
  <collections>...</collections>
  <page-size>...</page-size>
  <max-hits>...</max-hits>
  <min-score>...</min-score>
</query>

```

All elements except `condition` are optional. In the following `config.vm` example file, a search for documents containing a search term is defined. Only documents without any access restrictions are returned:

```

<query>
  <condition>
    <and>
      <vql-statement>
        &lt;#MANY&gt;&lt;#STEM&gt;${searchText}
      </vql-statement>
      <vql-statement>
        &lt;#MANY&gt;&lt;#STEM&gt;free &lt;#IN&gt;
        noPermissionLiveServerRead
      </vql-statement>
    </and>
  </condition>
  <result-fields>
    <field>title</field>
    <field>name</field>
    <field>visiblePath</field>
    <field>ip_abstract</field>
  </result-fields>
  <sort>
    <criteria>
      <field>pl_PLZ</field>
      <ascending/>
    </criteria>
    <criteria>
      <field>name</field>
      <ascending/>
    </criteria>
  </sort>
  <page-size>5</page-size>
  <collections>
    <collection>cm-contents-${language}</collection>
  </collections>
</query>

```



```

</collections>
<log>
  <context>search</context>
  #if ($user.isLoggedIn())
    <login>${user.login}</login>
  #end
  <query>${searchText}</query>
</log>
</query>

```

In the following, the meaning of the elements of which the search query consists is explained:

- **condition:** This defines the search query. The `condition` may contain the following operators:
 - **and:** this element combines the elements contained in it with a logical *and*.
 - **contains:** searches files, whose field with name `field` contains the string `value`. Example:

```

<contains>
  <field>keywords</field>
  <value>service</value>
</contains>

```

- **contains-match:** searches files, where the wildcard pattern `value` matches the content of the field named `field`. Valid wildcards are asterisk (*) and question mark (?).
- **equals:** searches files, where the field named `field` equals exactly the string `value`. Example:

```

<equals>
  <field>name</field>
  <value>mastertemplate</value>
</equals>

```

- **or:** combines the elements contained in it using *or*.
- **starts-with:** searches files, where the field with name `field` begins with the string `value`.
- **vql-statement:** contains the search query in the explicit Verity query language. For a detailed explanation of the syntax, please refer to the [Infopark Search Server Manual](#). Note that special characters need to be specified as HTML entities, for example `<` as `<`.
- **result-fields:** This element specifies in its `field` subelements the fields of the resulting documents to be added to the search results. This makes it possible to display the values of these fields on the result pages.
- **sort:** Here, one or more criteria for sorting the search results can be specified. Each criterion consists of a field name, `field`, whose contents is used for sorting, as well as optionally either ascending or descending to define the sort order.
- **page-size:** By means of this element, the number of hits to be displayed on each result page can be specified.
- **collections:** Specify here the collections to be searched. If this information is omitted, all collections are searched.
- **log:** Specify here the elements to be tracked. Permitted elements are `context`, `login`, and `query`. `context` must be specified, otherwise [tracking](#) is disabled.

Please note that the fields to be searched and also to be returned by the search engine (see `result-fields`, `sort` above) must have been defined in the Verity configuration prior to creating the collection concerned (see the [Infopark Search Cartridge](#) documentation).

Structure of the view.vm Configuration File

This Velocity template serves to generate the HTML text of the search form and the results list.

In the template, you can use the following keywords to access objects in the Velocity context:

- `$action`: The action to use in the search form.
- `$params`: The list of search parameters used so far. Internal parameters (whose name begins with a underscore character) do not show up in this list.
- `$text`: offers access to localized texts (originating from the localization files in the definition directory).
- `$locale`: the current locale.
- `$search`: Tool with which URLs can be generated. The following methods are available:
 - `String getPageUrl(Page page)`: returns an URL for jumping to the specified page of the results list.
 - `String getSearchUrl(String parameter, String value)`: returns an URL for performing a search using the given parameter.
 - `String highlight(String word, String text, String prefix, String suffix)`: encloses each occurrence of the specified word in text into the strings prefix and suffix ein.
- `$result`: allows accessing the results list by means of the following methods:
 - `int getHitCount()`: returns the number of hits.
 - `List getPages()`: returns the list of Page objects in the search result.
 - `Page getCurrentPage()`: returns the Page object which represents the current page of the results list.

The properties of the the objects listed above are described in the portlet's javadoc documentation. Example (starting with the form, then comes the Velocity template):

```
<form method="get" action="$action">
  <input type="text" name="searchText" value="$!params.searchText"/>
  <input type="submit" name="_buttonSearch"
    value="$text.buttonSearch"/>
</form>

#if ($result)
  #if ($result.hitCount == 0)
    $text.errorNoResults
  #else
    <ul>
      #foreach ($item in $result.currentPage.hits)
        #set($path = $document.getUrl($item.visiblePath))
        <li><a href="$path">$item.title</a></li>
      #end
    </ul>
  #end
#end
```

Configuring Access to the Search Engine

The search engine is configured in the `pm.xml` file as a the `searchEngine` bean. Two implementations are available, direct access via the Search Server, and access via the Content Management Server.

SesSearchEngine: For the live system search the search queries are sent directly to the Search Server running on the live system:

```
<bean id="searchEngine"
  class="com.infopark.libs.search.ses.SesSearchEngine">
  <property name="host"><value>localhost</value></property>
  <property name="port"><value>3011</value></property>
</bean>
```

AdvancedCmSearchEngine: When searching via the Content Management Server, the editorial system is searched. Der Content Management Server redirects the queries to the Search Server associated with the editorial system. However, it adds to each hit the path of the CMS file found:

```
<bean id="searchEngine"
  class="com.infopark.libs.search.cm.AdvancedCmSearchEngine">
  <property name="host"><value>localhost</value></property>
  <property name="port"><value>3002</value></property>
  <property name="user"><value>root</value></property>
  <property name="tokenManager">
    <ref bean="tokenManager"/>
  </property>
</bean>
```

5.5.3 Example

The following example illustrates how a new search can be created in the portal included in the demo content. The search term that was entered is to be found in the main content, the title, or the abstract, i.e. in the fields named `blob`, `title`, and `ip_abstract`. On the result pages, the fields `title` and `ip_abstract` are to be output. The abstract is to be displayed only if it contains text, i.e. if it is not empty.

Preparing the Search Engine Indexes

The fields mentioned above can only be used after the configuration of the search engine on the live server side has been extended so that the additional fields are available and filled-in when documents are indexed.

For this, open the file belonging to the collection which is to be searched, located in the instance concerned.

```
instance/instanceName/config/vdk/styles/collectionName/style.ufl
```

Add the following fields:

```
# -----
# Specify additional application-specific fields here in their own
# data-table[s].

data-table: nps
{
  **** Existing fields
  ****
  **** New fields

  varwidth:   ip_abstract dxa
  autoval:    collection DBNAME
}
```

Stop the Search Engine Server:

```
./rc.npsd stop SES
```

For the changes to become effective, the collection needs to be created again and the documents need to be reindexed. To do this, start the Search Server in single mode:

```
./SES -single
```

Now, perform the following steps for the collection concerned:

```
SES>deleteCollection collectionName
SES>createCollection collectionName
SES>exit
```

Start the Search Engine Server:

```
./rc.npsd start SES
```

Now, connect to the Content Management Server:

```
./client localhost 3001 root demo
```

Reindex all NPS files using the following command:

```
CM>indexAllObjects
CM>exit
```

Create the new Search

Change to the directory

```
/instance/instanceName/webapps/PM-PL/WEB-INF/templates/search
```

and make a copy of the existing directory `nameTitle` and name it `bodySubjects`. Then change to this directory and edit the files named `config.vm` and `view.vm`.

In the file `config.vm` the search query is configured. For this purpose change the contents of the `condition` element so that it contains only the following `vql-statement` element:

```
<vql-statement>
  (&quot;${suche}&quot; &lt;#IN&gt; blob) &lt;#OR&gt;
  (title &lt;#SUBSTRING&gt; &quot;${suche}&quot;) &lt;#OR&gt;
  (ip_abstract &lt;#SUBSTRING&gt; &quot;${suche}&quot;)
</vql-statement>
```

Please note that the search query consists of several parts (one per field), all of which are combined with OR. Since the main content (`blob`) is a document zone and not a field (like `title` and `ip_abstract`), the operator `IN`, which searches document zones, is used for searching it. The `SUBSTRING` operator, on the other hand, searches the contents of fields.

Since the search results are to include the contents of the `ip_abstract` field, it needs to be added to the `result-fields` element:

```
<result-fields>
  <field>objId</field>
  <field>name</field>
  <field>title</field>
  <field>score</field>
  <field>visiblePath</field>
  <field>ip_abstract</field>
</result-fields>
```

Creating the Search and Results Page

The search and results page are created in the file `view.vm`. Its upper section defines the search form:

```
<form action="$action" method="post">
  <div>
    Ihre Suche <input type="text" name="suche"
      value="$!params.suche" />
  </div>
  <input type="submit" name="dialog.buttonOk"
    value="$text.buttonOk" />
</form>
```

The lower section of the file `view.vm` makes it possible to display the results after the user has submitted the search form. All results are to be displayed as a list. For each hit, the (linked) title and the abstract of the document is to be displayed:

```
<ul>
  <li>Verlinkter Titel<br />
  ip_abstract</li>
  <li>...</li>
  <li>...</li>
</ul>
```

To achieve this display format, the following code is used:

```
#if ($result)
  #if ($result.hitCount == 0)
    <div>$text.noHits</div>
  #else
    <!-- Number of results -->
    <div>$text.hitCount $result.hitCount</div>
    <!-- Begin of results list generation -->
    <ul>
      #foreach ($item in $result.currentPage.hits)
        <li>
          #set ($path = $document.getUrl($item.visiblePath))
          #if ($path) <a href="$path">#end
            $item.title
          #if ($path) </a>#end
          #if ($item.ip_abstract) <br /> $item.ip_abstract#end
        </li>
      #end
    </ul>
    <!-- End of results list generation -->
    <!-- Create links to other results pages -->
    <div>
      #foreach ($page in $result.pages)
        #if ($page.isCurrent())
```

```

        $page.number
    #else
        <a href="$search.getPageUrl($page)">$page.number</a>
    #end
#end
</div>
<div>$text.page $result.currentPage.number $text.pageOf
    $result.pages.size()
</div>
<!-- End of results pages overview -->
#end
#end

```

In order to remove the first directory from the paths in the results list (so that the root folder is not displayed), use the following code in the foreach loop:

```

## Search for directory delimiter (slash)
#set( prefix $path.indexOf("/", 1) )
## Do not use $path to access the path, instead use:
$path.substring($prefix + 1)

```

Including the Portlet in a Web Page

To include the portlet in a web page, add the following line to a layout file used for generating the pages to be displayed:

```
<npspm includePortlet="/PM-PL/search" instance="bodySubjects" />
```

The portlet is now displayed in the separate preview and can be used.

Clearing the Input Fields of the Search Form

The search portlet stores the values contained in its input fields. If, after a search, a website visitor reenters the search page (the page containing the search portlet), the form contains the values that were entered before.

Since this behavior is normally unwanted, you can have the portlet clear the input fields. For this, up to version 6.5.0, add the `reset=true` parameter to the URL that links to the search page. Example:

```
<a href="/suchen.html?reset=true">Search</a>
```

From version 6.5.1, add a link to the remote control portlet instead. Example.

```

<npspm includePortlet="/PM-PL/remoteControl" instance="search">
targetUrl=/search.html
emptySearch=true
emptySearchLinkText=Search
</npspm>

```

The linked text is configured in the portlet itself and reads "New Search". If you wish to be able to specify the linked text in each call to the portlet, using the parameter `emptySearchLinkText` like in the example above, replace in the `webapps/PM-PL/WEB-INF/templates/remote/search/view.vm` template file the line

```
<a href="javascript:document.emptySearch.submit();">$text.emptySearch</a>
```

with the following code:

```
#if ($params.emptySearchLinkText)
    #set ($linkText = $params.emptySearchLinkText)
#else
    #set ($linkText = $text.emptySearch)
#end
<a href="javascript:document.emptySearch.submit();">$linkText</a>
```

5.6 Portlet for Displaying and Sorting a Table

This portlet displays a table and makes it possible to sort the table rows.

Using this portlet requires a separate licence.

5.6.1 Operation

The table rows can be sorted by clicking a column header in the portlet. This causes the rows to be sorted alphabetically by the values in the clicked column. Another click on this column header reverses the sort order. If more rows than the configured number of lines per page exist, links for browsing the table are displayed.

5.6.2 Configuration

The layout of this portlet (and all other portlets as well) can be specified by means of a Velocity template. This template is located in the PM-PL web application, in the `/WEB-INF/templates/com/infopark/portlet/gridview` directory. The layout can be adapted as desired.

In order to display a table, include it as an XML element in the `npspm` element. To avoid conflicts with editors, a CDATA section should be used as shown in the example below.

5.6.3 Usage

The portlet can be integrated into layout files according to the following example:

```
<npspm includePortlet="/PM-PL/gridview" ... ><![CDATA[
  <grid rowsPerPage="10">
    <row><cell>Titel 1</cell><cell>Titel 2</cell></row>
    <row><cell>1.1</cell><cell>1.2</cell></row>
    <row><cell>2.1</cell><cell>2.2</cell></row>
    <row><cell>3.1</cell><cell>3.2</cell></row>
  </grid>
]]></npspm>
```

You can optionally use the `rowsPerPage` attribute of the `grid` element to configure the number of table rows to be displayed in the portlet.

5.7 Storing User-Specific Portlet Preferences

Portlets can store user-specific settings (used for personalization purposes, for example) in a number of ways. Infopark Portal Manager offers three storage variants for such data:

- **In RAM (`MemoryPreferencesStorage`):** This is the fastest method. However, the stored data is lost when the application server is restarted.
- **In the file system (`FilesystemPreferencesStorage`):** The settings are stored as Java preferences in the home directory of the user who has started Trifork server. With many users or many portlets, the storage capabilities may be insufficient. Under Linux, for example, a directory may not contain more than 32.768 directories.
- **In a database (`DatabasePreferencesStorage`) (from version 6.7.1):** Storing the user data in a database is the most flexible method. However, it increases the administrative effort required.

The storage method can be specified by means of a bean in the `preferencesStorage` property which can be found in the `pm.xml` file of the portlet web application. Only one of these beans must be used. Therefore, two beans are commented out in the following example.

```
<property name="preferencesStorage">
  <!-- Store in memory only. Prefs will not survive appserver restart -->
  <!-- bean class="com.infopark.pm.MemoryPreferencesStorage" /> -->
  <bean class="com.infopark.pm.FilesystemPreferencesStorage" />
  <!-- Store in database table. Configure a dataSource bean -->
  <!--
  <bean class="com.infopark.pm.DatabasePreferencesStorage">
    <property name="dataSource" ref="portletPrefsDataSource"/>
    <property name="tableName" value="portlet_prefs"/>
    <property name="portletColumn" value="portlet"/>
    <property name="loginColumn" value="login"/>
    <property name="keyColumn" value="key"/>
    <property name="indexColumn" value="idx"/>
    <property name="valueColumn" value="value"/>
  </bean>
  -->
</property>
```

No configuration is required for the beans used to store the user settings in RAM or in the file system.

The bean for storing the data in a database assumes that the data source exists and is accessible. The database can be configured by means of the `com.infopark.pm.DatabasePreferencesStorage` bean which has the following `property` elements.

- **dataSource:** Use the `ref` attribute to refer to a bean, for example `portletPrefsDataSource`, which specifies and configures the class to be used for accessing the database. In this example, all access is done via JDBC:

```
<bean id="portletPrefsDataSource"
  class="org.springframework.jndi.JndiObjectFactoryBean">
  <property name="jndiName">
    <value>jdbc/portletDB</value>
  </property>
</bean>
```

- **tableName:** The name of the database table.
- **indexColumn:** Name of the primary key column.
- **keyColumn:** Name of the column in which the name of the saved value is stored (the portlet and user-specific parameter name).
- **portletColumn:** The name of the column in which the portlet ID is stored.
- **loginColumn:** The name of the column in which the user name is stored.

- **valueColumn:** The name of the column in which the parameter value is stored.

The database interface is only used by the [News Portlet](#) included in Infopark Portal Manager. To use the interface for your own portlets in the way shown above, make sure that the following prerequisites are met:

- **Database is available:** The database must have been set up and supplied with the database table specified above. The values of all columns except `indexColumn` have the `VARCHAR` type. The values of `indexColumn` have the `INT` type. The following database command creates such a table:

```
CREATE TABLE portlet_prefs (
  idx INT,
  key VARCHAR,
  login VARCHAR,
  portlet VARCHAR,
  value VARCHAR
)
```

- **JDBC drivers have been installed:** The JDBC driver for your database must have been installed in Trifork server. You can upload the driver by means of the server console.
- **Data source has been set up:** In Trifork server, you require a `dataSource` and a `pooledDataSource` to access your database via the JDBC driver.

After the database and access to it have been configured, you can store and retrieve user-specific settings via the API of the bean, analogously to the news portlet.

5.8 Note on Developing Portlets

From version 6.0.4 the Portal Manager includes a Java API which is required for developing portlets. The API can be found in the `/share/doc/javadoc/portlet-api` directory, the documentation of the classes is located in the `/share/doc/javadoc/pm` directory. Please open the file `index.html` in this directory and also take a look at the packet overview ([overview](#)).

In the installation directory, you can find in the `examples/portlet` directory a sample portlet web application including a couple of sample portlets. This web application can be deployed to `/PM-EX` using the supplied ant build file.

Each portlet web application of the Infopark Portal Manager includes a servlet with which the portlets can be tested without embedding them into the content. The path of this servlet is `/webapp/debug/`, with `webapp` standing for the web application, for example `/PM-EX/debug/`. It outputs links to the portlets contained in the web application so that they can be tested by clicking them. From Version 6.5.1, this servlet is also available in the Portal Manager web application under `/PM/debug/`. Its purpose is to test the [WSRP functionality](#).