



Global Knowledge®

Expert Reference Series of White Papers

How to Update IBM WebSphere Portal Using the Scripting Interface Tool

How to Update IBM WebSphere Portal Using the Scripting Interface Tool

G. David Wilkerson, Global Knowledge instructor

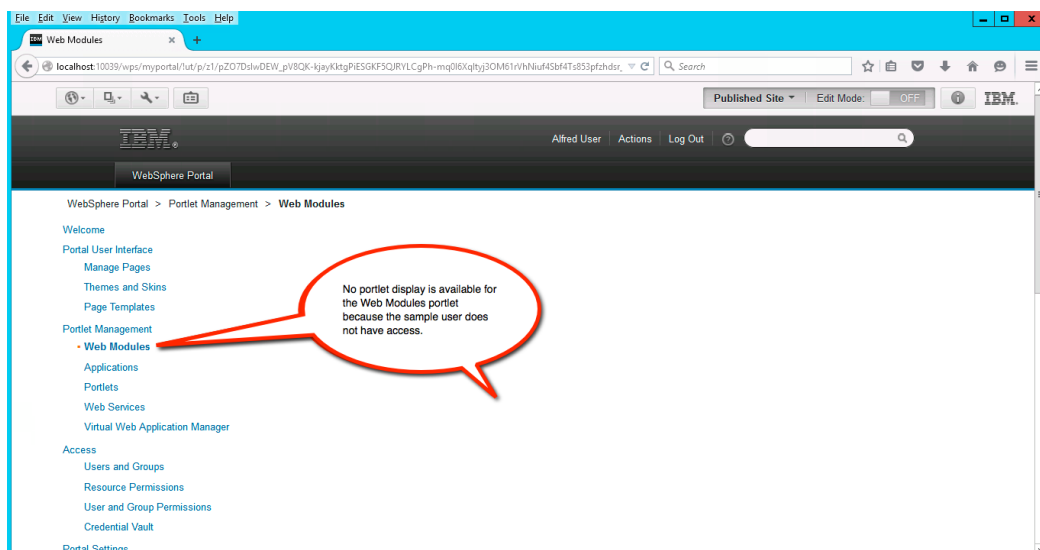
You are wrapping things up for the day when you get an email stating that the Europe, Middle East and Africa (EMEA) development team needs new pages with some portlets configured on the QA portal. The moment you make a mental note to do this first thing in the morning, your boss sends you an instant message telling you that the Asia Pacific (AP) QA team plans to do some overnight testing. As you top off your coffee mug, you consider giving developers rights to use the XML configuration interface, but then you remember your organization's security mandates prohibit such a broad administrative scope.

By briefly comparing the portal administrator's tools we can identify some use cases where the scripting interface may be the tool of choice. Next, we'll review how to launch the tool which can help you understand and select options for its use. This puts us in position to learn a few basic commands. Finally, we will take a look at creating and testing our own custom script.

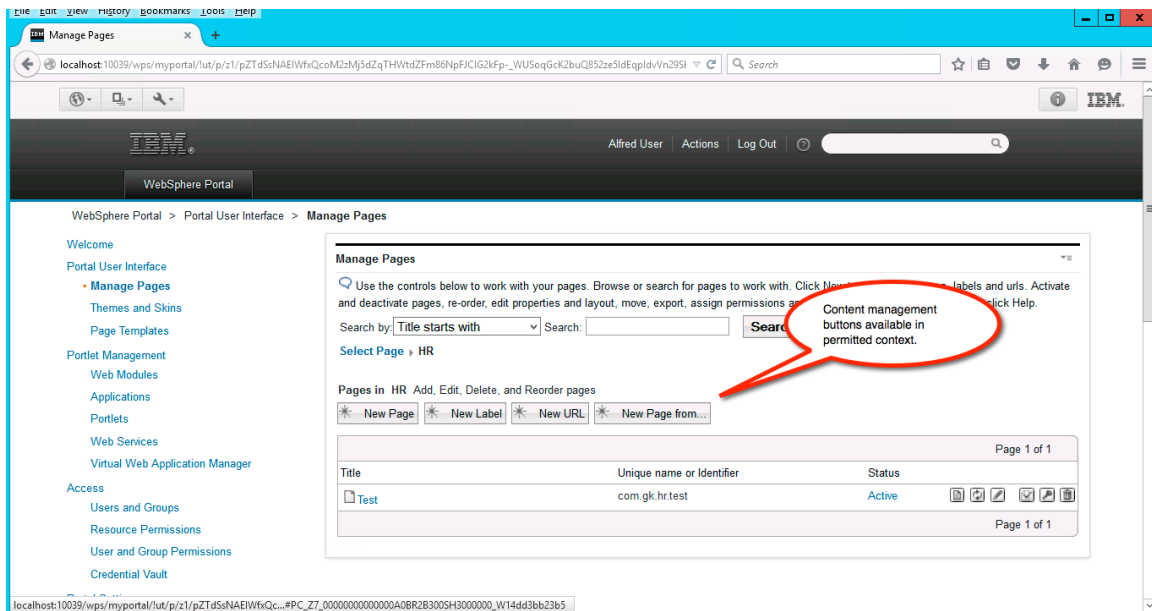
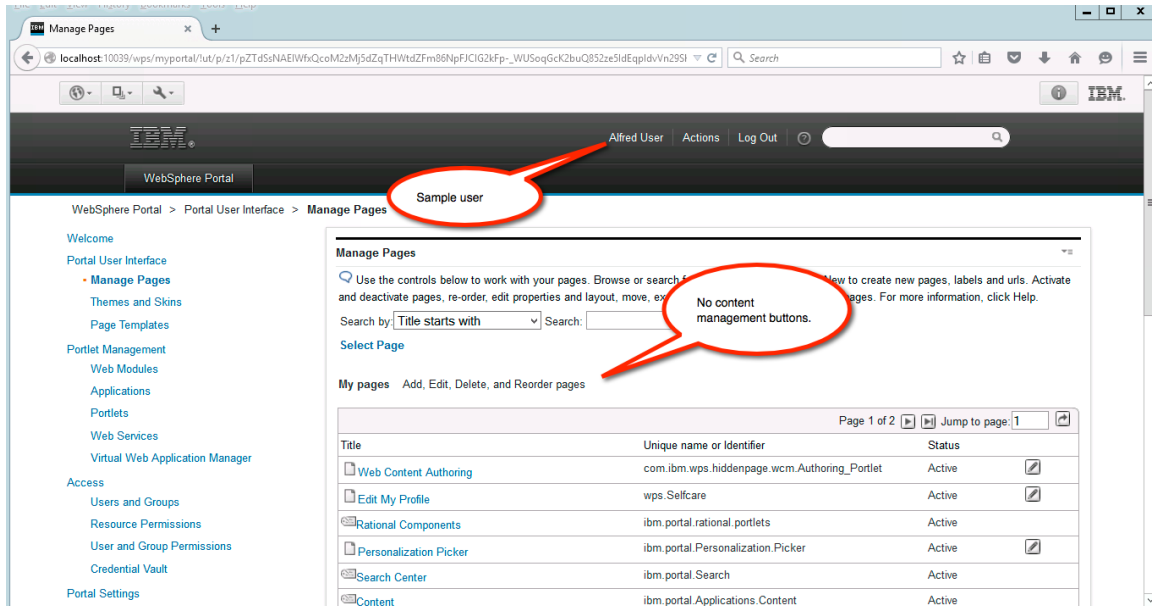
Compare Tools

The portal administrator's toolkit includes administrative portlets, the XML configuration interface, ReleaseBuilder, and the Portal Scripting Interface. IBM's administrative portlets provided for IBM WebSphere Portal are a collection of graphical user interface (GUI) tools available for a wide variety of portal administration tasks. Out of the box, access is limited to members of the portal administration group. However, access to the pages and portlets can be discretely delegated to selected users.

In the following example, I've created a sample user named "Alfred User" who is granted access to a subset of the administrative interface. I have granted Alfred access to the Manage Pages portlet to carry out actions using that portlet. But, I did not give Alfred access to other portlets, such as Web Modules.



In addition to managing the tools available to our user, we are also able to manage the resources the user can create or manage. In this example I have given Alfred the role of manager for a subset of the portal's node hierarchy under a label named HR. When the user accesses the Manage Pages portlet he does not see buttons to create or delete nodes until he navigates to the HR label context.



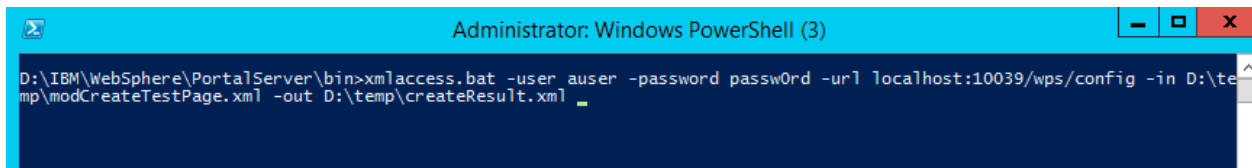
This example is somewhat limited but serves to illustrate an important point, which is your ability to manage what tools and resources are exposed to users. As with other graphical tools, the portal administration portlets are well suited for ad hoc work but not appropriate where precise consistency is required. For example, if we create and configure a page in one environment, how do we repeat that effort in another without introducing the credible likelihood that a step may be skipped or that a setting may be improperly configured? This can be a

serious limitation.

XML Configuration Interface

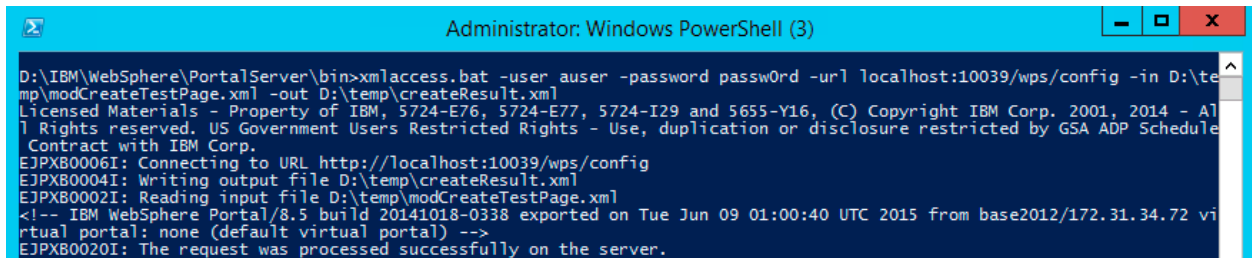
Another tool available to the administrator is the XML configuration interface, commonly called XMLAccess after the script used to launch the tool. Benefits of using the tool derive from a user's ability to export and import entire portal configurations or subsets thereof. The commands are defined by an XML syntax contained in a file and, as such, function much like a script. The tool allows for a task or a collection of tasks to be performed repetitively with predictable outcomes. However, unlike the portal administration portlets, it is not possible to limit the scope of work performed by users of this tool. Access to the tool is granted, by default, to portal administrators. Users of the tool must have a manager role on the virtual resource XML_ACCESS and the security administrator role on the virtual resource PORTAL.

Continuing with the previous example, I have granted a user the required roles on two virtual resources: PORTAL and XML_ACCESS. Using a sample script, modCreateTestPage.xml, I launched the XML configuration interface:



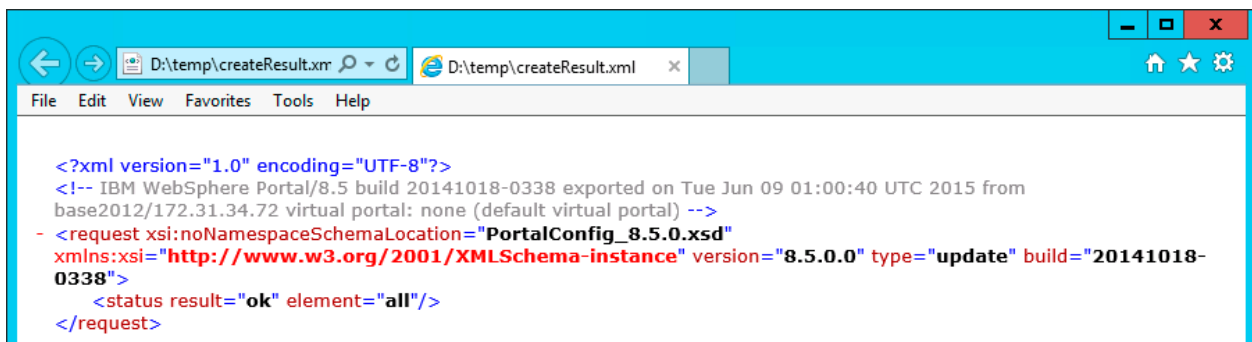
```
Administrator: Windows PowerShell (3)
D:\IBM\WebSphere\PortalServer\bin>xmlaccess.bat -user auser -password passw0rd -url localhost:10039/wps/config -in D:\temp\modCreateTestPage.xml -out D:\temp\createResult.xml
```

When the script is completed a reference to the outcome is displayed in the console:



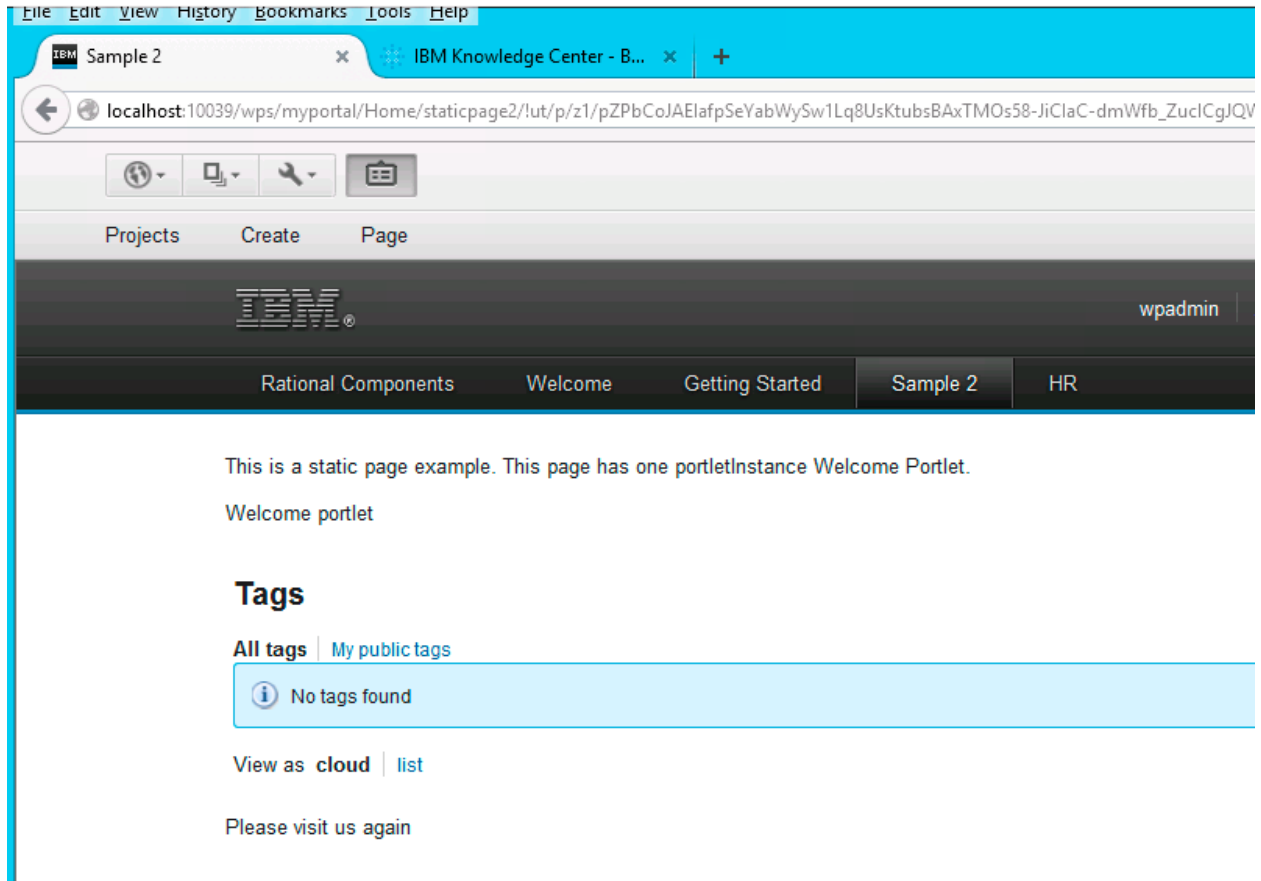
```
Administrator: Windows PowerShell (3)
D:\IBM\WebSphere\PortalServer\bin>xmlaccess.bat -user auser -password passw0rd -url localhost:10039/wps/config -in D:\temp\modCreateTestPage.xml -out D:\temp\createResult.xml
EJPCB0004I: Writing output file D:\temp\createResult.xml
EJPCB0002I: Reading input file D:\temp\modCreateTestPage.xml
<!-- IBM WebSphere Portal/8.5 build 20141018-0338 exported on Tue Jun 09 01:00:40 UTC 2015 from base2012/172.31.34.72 virtual portal: none (default virtual portal) -->
EJPCB0020I: The request was processed successfully on the server.
```

The output file reports the successful update:



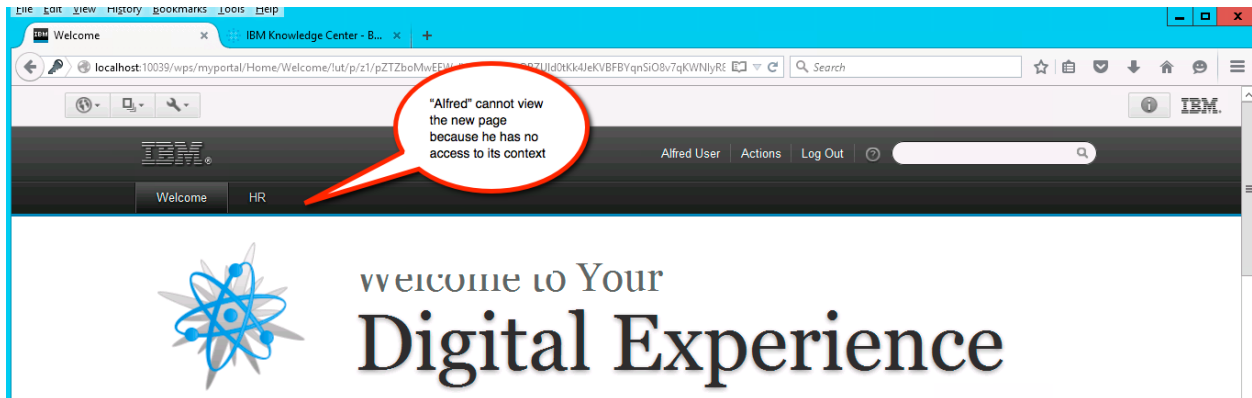
```
D:\temp\createResult.xml
File Edit View Favorites Tools Help
<?xml version="1.0" encoding="UTF-8"?>
<!-- IBM WebSphere Portal/8.5 build 20141018-0338 exported on Tue Jun 09 01:00:40 UTC 2015 from
base2012/172.31.34.72 virtual portal: none (default virtual portal) -->
- <request xsi:noNamespaceSchemaLocation="PortalConfig_8.5.0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="8.5.0.0" type="update" build="20141018-
0338">
  <status result="ok" element="all"/>
</request>
```

And, after logging into the portal as a member of the portal administrative group, we see the new sample page named Sample 2:



The striking outcome is that our sample user, Alfred, has no access to this page and it has been created in a context to which the user does not have access. This is not a surprising result, given the added role assignments. However, it is inconsistent with our objective of providing a script-based solution to our user that constrains the scope of his management.

When logging in as "Alfred" we do not see the page (Please note: I removed roles for the two virtual resources before logging in to view the page):



I've included ReleaseBuilder in this discussion only as a means of noting the role it plays in portal administration. Essentially, this tool provides a differential file that extends the use of XMLAccess for managing portal configurations. The security scenario for XMLAccess applies. A user would export the configuration of two environments. One, considered the target, will eventually be updated. Another, considered the source, contains changes not yet reflected in the target. ReleaseBuilder receives the two export files, compares them, and generates a differential file that, in turn, can be imported on the target using XMLAccess.

Portal Scripting Interface

The Portal Scripting Interface is a command line interface (CLI) tool and is often referred to by the name of the script used to launch it: `wpscript`. In general, the Portal Scripting Interface is managed in the same way as portal administration portlets. The significance is that delegated administration is provided through access control. Working with `wpscript` requires a user to have access to the portal and the resources the user intends to administer.

It allows implicit derivation while performing administrative work. What is derivation? When a portal resource such as a page is specialized by users, its definition may be derived from another page. This means that a user of the Portal Scripting Interface can create derivations of a resource in the same process, depending on the user's access rights. Whether a page is unique to an individual user, sometimes called a private page, or common to a group of users, depends on the user's role on the initial page. That is, if a user is assigned the editor role on a source page, the derived page is available to others. Conversely, if a user's role is privileged user, the derived page is private and viewable only by the creator. There is a significant contrast between page creation using `wpscript` and XMLAccess, at this point. XMLAccess does not facilitate creation of derived resources. Conversely, `wpscript` does. Keep in mind the nature of roles and access control. XMLAccess users are acting as full portal admins. No area of the portal is excluded from management. `Wpscript` can be bound by discrete or delegated access, XMLAccess is a blunt instrument.

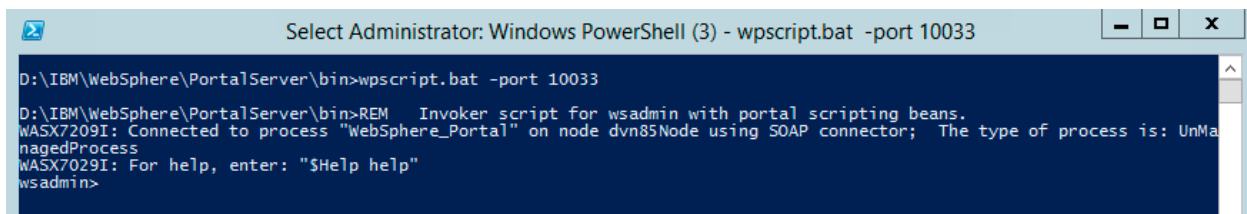
Use Case

A good use case for the Portal Scripting Interface, `wpscript`, is one in which delegated access is needed in a context in which repeatable tasks can be performed or scripted. Herein is our example. Our user "Alfred" is a developer needing access to a sandbox server. He needs to create pages, place portlets, etc. for testing. However, there are other developers with a similar need. We could certainly delegate access and provide the portal administration portlets, but the developer's time would be poorly spent. In addition, it is plausible that he would need to delete and recreate pages and other resources repeatedly. It is also plausible that the pages and portlets he defines would need to be promoted to a QA environment.

The Portal Scripting Interface is the tool of choice. Once a script is defined it is reusable. The outcome is predictable. Users with delegated access can work concurrently without negatively impacting others. Finally, providing template scripts can minimize the time consumed by individuals. All that remains, then, is to discuss how to use the tool.

Launch the Tool

The scripting interface is derived from `wsadmin`, the administrator's scripting tool for WebSphere® Application Server. Launching the tool, at a minimum, consists of invoking the script and passing the port number for the Simple Object Access Protocol (SOAP) port of the host system. Doing so opens the interface to the local host using the default language and connection type. Notice in the screenshot that the command line is a `wsadmin` prompt.

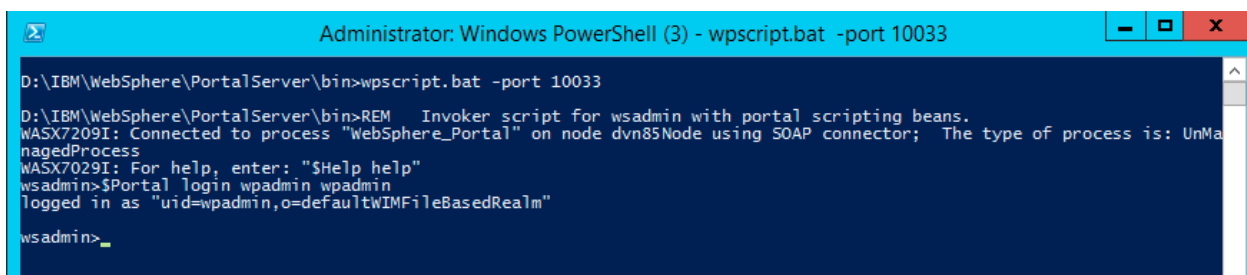


```
D:\IBM\WebSphere\PortalServer\bin>wpscript.bat -port 10033
D:\IBM\WebSphere\PortalServer\bin>REM Invoker script for wsadmin with portal scripting beans.
WASX7209I: Connected to process "WebSphere_Portal" on node dvn85Node using SOAP connector; The type of process is: UnManagedProcess
WASX7029I: For help, enter: "$Help help"
wsadmin>
```

As was just pointed out, the interactive mode does not require any additional arguments. I prefer the interactive mode when developing a script because it allows me interact directly and dynamically with the portal to perform simple administrative tasks. Another reason to use interactive mode is when you expect to carry out an ad hoc task to be executed only once. Suppose, for example, that I want to set role assignments for my developer, "Alfred". I could do so by modifying the permissions of a node interactively.

The scripting interface works with a variety of portal management beans. Management beans are implementations of the Java Management Extension (JMX) specification and are implemented for the management of WebSphere Application Server. Using `wpscript` provides access to the extended beans.

To access the beans our first task is to log in to the portal. In the screenshot we are logging in using Java Application Control Language (JACL), the default language. We'll discuss the language options in a moment.



```
D:\IBM\WebSphere\PortalServer\bin>wpscript.bat -port 10033
D:\IBM\WebSphere\PortalServer\bin>REM Invoker script for wsadmin with portal scripting beans.
WASX7209I: Connected to process "WebSphere_Portal" on node dvn85Node using SOAP connector; The type of process is: UnManagedProcess
WASX7029I: For help, enter: "$Help help"
wsadmin>$Portal login wpadmin wpadmin
Logged in as "uid=wpadmin,o=defaultWIMFileBasedRealm"
wsadmin>
```

Once we have logged in to the portal, we can perform a variety of administrative tasks such as creating a portal page. In addition to the interactive mode are the command and script modes.

Of the two practical modes, the most common for the use case is the script mode. In the remainder of this paper, we will look at setting up the tool to use a preferred language, connection type, and simplify script processing with a profile.

Jython vs. JACL

Among the decisions we must make when using `wpscript` is whether to use the default JACL language or the preferred Jython language. In reality, the decision is a personal one based on experience and familiarity. For example, if you were already using JACL for your `wsadmin` scripts, you would certainly wish to continue using it for `wpscript`.

Personally, I prefer to use Jython because the dot notation of the language and the Python style syntax are more familiar. In this paper, we will use Jython, but there are abundant examples in the product documentation. To override the language is as simple as setting your preferences when launching the tool.

Tool Options

Options include connection protocol, language, port, script file, and profile. The first option, connection protocol is identified by a parameter named `-conntype`. By default the script uses SOAP. `Wsadmin` recognizes three types, SOAP, Remote Method Invocation (RMI), and NONE. However, `wpscript` only works with SOAP or RMI.

The language option may be provided in the command to launch the script. However, if it is not specified the interpreter will evaluate the command syntax and if the language is recognized, the interpreter will execute commands in the supplied language. This includes the ability to identify language from the file extension of any script files such as `.jacl` or `.py`. If a profile is used, the language will be inferred from the extension of the provided profile.

The port is simply the numerical value of the network port associated with a socket such as 10033 on an out-of-the-box 8.5 WebSphere Portal Server. This can be obtained from the IBM Integrated Solutions Console by examining the configuration of the target application server. Alternatively, it can be obtained from a file named `serverindex.xml` found in the application server's configuration repository.

Scripts, as demonstrated earlier, are identified to `wpscript` with the `-f` parameter. They may be written in either JACL or Jython, and they may be combined with a profile "script" to simplify setting up the `wpscript` session.

Profiles are passed to `wpscript` using the `-p` parameter. These are used before running a task-related script or before entering interactive mode. The purpose they serve is to set up the environment based on a user's specific information. A common example is to provide the portal login in the profile script.

```
# scripting profile
# contains log-in procedure on portal with disabled security
if len(sys.argv) != 2:
    print "invocation syntax: wpscript -f testme.py -profile
mylogin.py user_ID password"
    sys.exit(1)

user = argv[0]
pwd = argv[1]
Portal.login(user, pwd)
```

A profile is a script that runs before the main script, or before entering interactive mode. Profiles can be used to set up environment-specific behavior or user-specific data. Profiles are specified when invoking `wpscript`, using the

`-profile` parameter. For example, the login command can be placed in a profile.

Basic Objects and Commands

Working with `wpscript` consists of understanding how to access the management bean, the related scripting objects, and what operations can be performed with them. A management bean is an implementation of the Java Management Extension, or JMX API. The IBM WebSphere Application Server implements management beans as a feature of the Java Enterprise Edition (Java EE) specification. These beans are powerful tools for administering the application server. For the benefit of portal administrators IBM has provided an extension accessed through `wpscript`. This bean is named `Portal`.

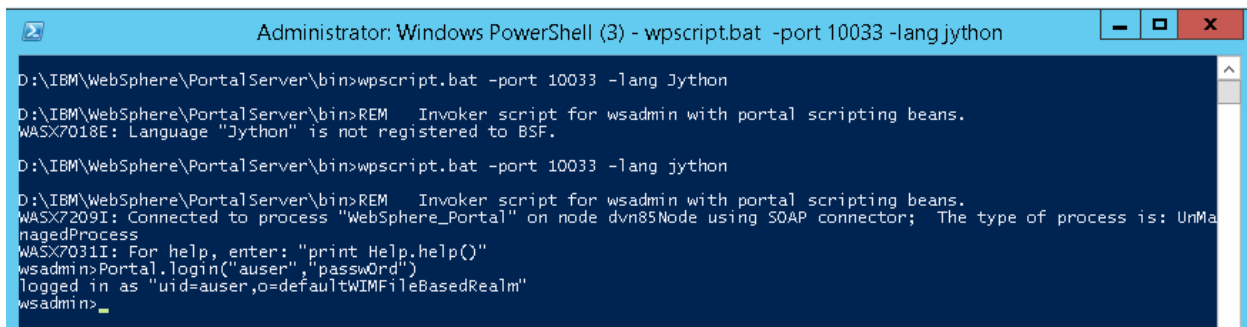
To access the management bean first, you launch the `wpscript` tool and authenticate with the application server. Next, you access the management bean by invoking the object name and the login method or operation. The JACL syntax for this is `$Portal login <user> <password>`. In most cases, users log in using a particular form for their identity. In our example we are using the User ID (UID). This is formed by concatenating the user's first initial with their last name as in "auser". In our example "Alfred User" would log into the bean as follows:

```
$Portal login auser passw0rd
```

The Jython example is:

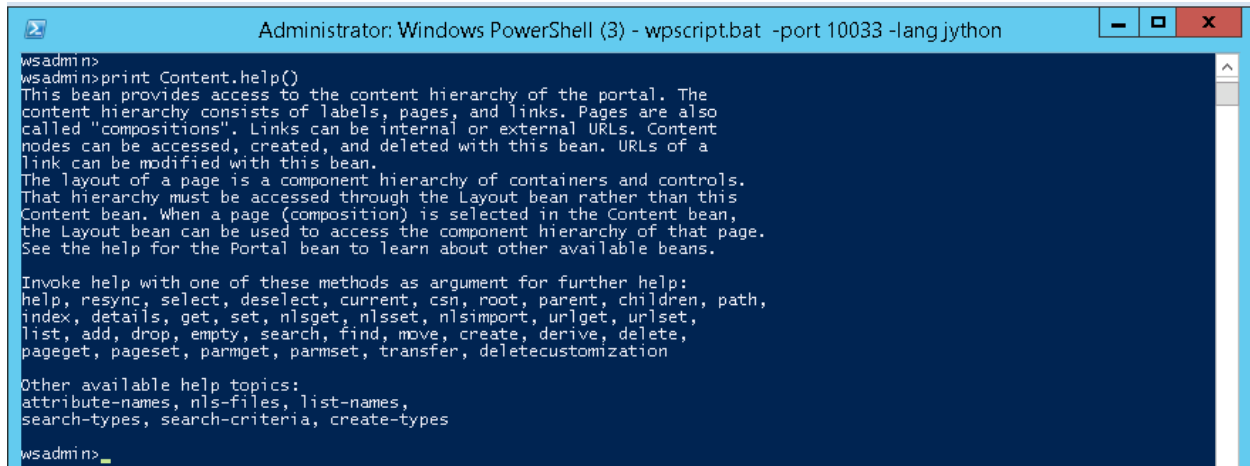
```
Portal.login("auser", "passw0rd")
```

A great way to test this is to use the interactive mode. First, we launch a `wpscript` session and specify the port and language. Then, once we've authenticated with the WebSphere Application Server, we enter our interactive login command. Here's an example using Jython:



```
Administrator: Windows PowerShell (3) - wpscript.bat -port 10033 -lang jython
D:\IBM\WebSphere\PortalServer\bin>wpscript.bat -port 10033 -lang Jython
D:\IBM\WebSphere\PortalServer\bin>REM Invoker script for wsadmin with portal scripting beans.
WASX7018E: Language "Jython" is not registered to BSF.
D:\IBM\WebSphere\PortalServer\bin>wpscript.bat -port 10033 -lang jython
D:\IBM\WebSphere\PortalServer\bin>REM Invoker script for wsadmin with portal scripting beans.
WASX7209I: Connected to process "WebSphere_Portal" on node dvn85Node using SOAP connector; The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>Portal.login("auser","passw0rd")
logged in as "uid=auser,o=defaultWIMFileBasedRealm"
wsadmin>
```

From this point we can access other scripting beans such as the Content bean. A great way to sort out the use of scripting beans is to access the Help bean:

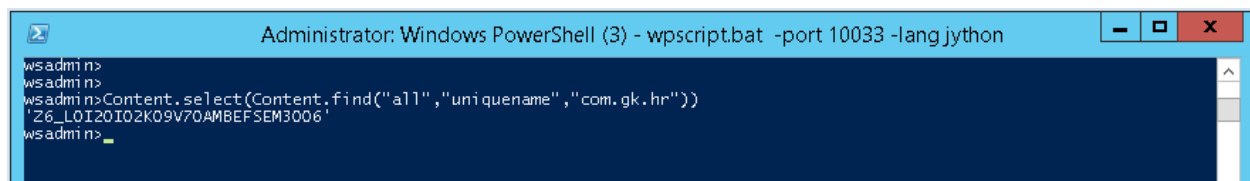


```
Administrator: Windows PowerShell (3) - wpscript.bat -port 10033 -lang jython
wsadmin>
wsadmin>print Content.help()
This bean provides access to the content hierarchy of the portal. The
content hierarchy consists of labels, pages, and links. Pages are also
called "compositions". Links can be internal or external URLs. Content
nodes can be accessed, created, and deleted with this bean. URLs of a
link can be modified with this bean.
The layout of a page is a component hierarchy of containers and controls.
That hierarchy must be accessed through the Layout bean rather than this
Content bean. When a page (composition) is selected in the Content bean,
the Layout bean can be used to access the component hierarchy of that page.
See the help for the Portal bean to learn about other available beans.

Invoke help with one of these methods as argument for further help:
help, resync, select, deselect, current, csn, root, parent, children, path,
index, details, get, set, nlsgget, nlsgset, nlsgimport, urlget, urlset,
list, add, drop, empty, search, find, move, create, derive, delete,
pageget, pageset, parmget, parmset, transfer, deletecustomization

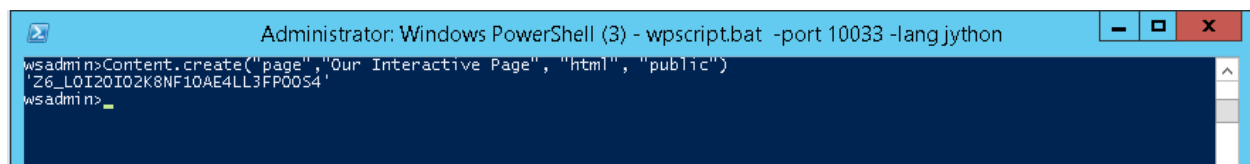
Other available help topics:
attribute-names, nls-files, list-names,
search-types, search-criteria, create-types
wsadmin>
```

The Content scripting bean provides a means of locating and selecting nodes such as pages or labels. By selecting a node we set the context for a future action. Here's an example of working with the Content scripting bean to find and select a particular node, the HR label, in our sample portal node hierarchy:



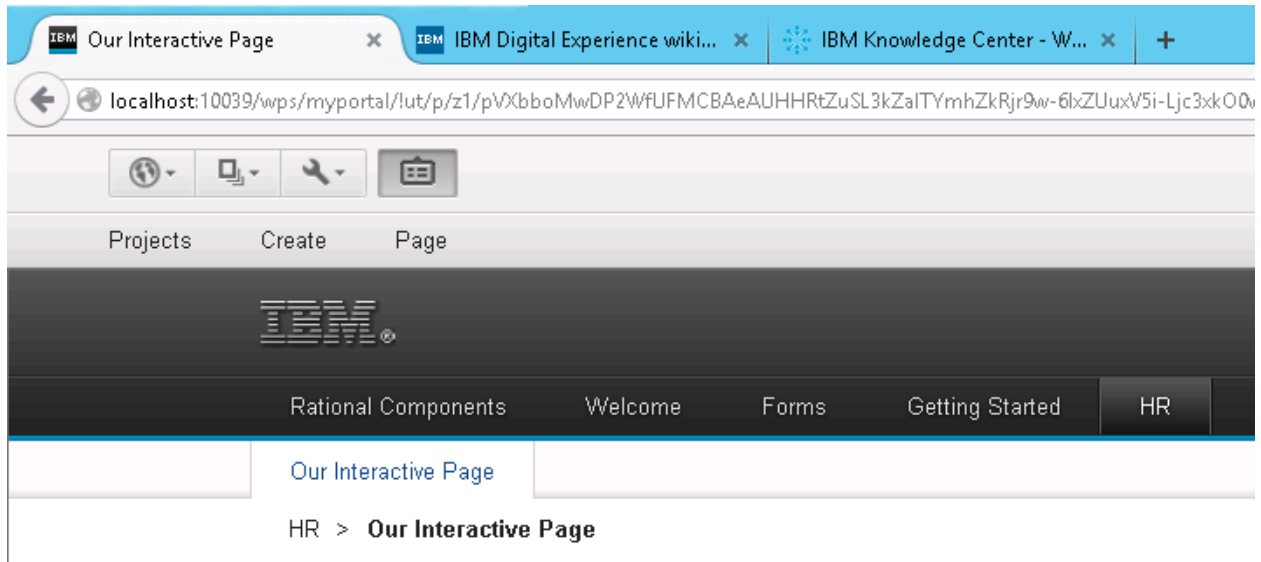
```
Administrator: Windows PowerShell (3) - wpscript.bat -port 10033 -lang jython
wsadmin>
wsadmin>
wsadmin>Content.select(Content.find("all","uniquename","com.gk.hr"))
'Z6_LOI20IO2K09V70AMBEPSEM3006'
wsadmin>
```

Following this, we are able to create a new node. Here we create a new page named "Our Interactive Page" using the Content scripting bean:



```
Administrator: Windows PowerShell (3) - wpscript.bat -port 10033 -lang jython
wsadmin>Content.create("page","Our Interactive Page", "html", "public")
'Z6_LOI20IO2K8NF10AE4LL3FP00S4'
wsadmin>
```

A view of the rendered page:



By this point, it should be clear that we can work interactively to carry out efforts to build scripts by issuing interactive commands, capturing them by creating scripts. Of course scripts will consist of far more than simply creating a page. For example, we will want to define the layout of a page and place portlets on it. By following the pattern we just described, you can identify the syntax of each command needed.

Custom Script

Scripts consist of some number of statements organized in a manner that combines the logic needed to manage flow and the commands needed to execute tasks such as page creation. Both JACL and Jython support the creation of “functions,” conditional logic, and iteration. In the following sample JACL script obtained from the IBM product documentation, we define a function with the keyword “proc.” This function is named “create_multi_col_page” and accepts two arguments. The first argument is for the page name and the other is for an array of portlet names:

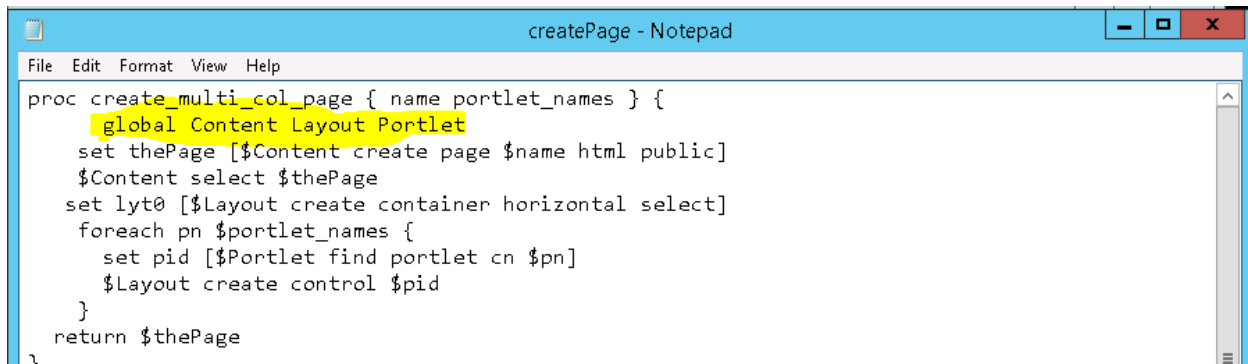
```
proc create_multi_col_page { name portlet_names } {
  global Content Layout Portlet
  set thePage [$Content create page $name html public]
  $Content select $thePage
  set lyt0 [$Layout create container horizontal select]
  foreach pn $portlet_names {
    set pid [$Portlet find portlet cn $pn]
    $Layout create control $pid
  }
  return $thePage
}

$Content select [$Content find all uniqueness "com.gk.hr"]

set newPage [create_multi_col_page "Sample Script Page" {
  "Welcome_to_WebSphere_Portal" } ]

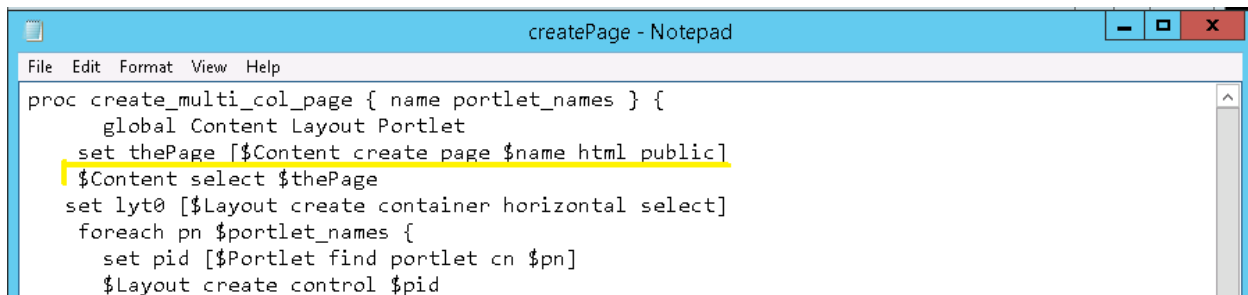
puts "ok, we are done."
```

The body of the function is set off by curly braces, "{}." Within the body we declare three variables named Content, Layout, and Portlet using the "global" identifier. Doing so allows references to these variables outside the function.



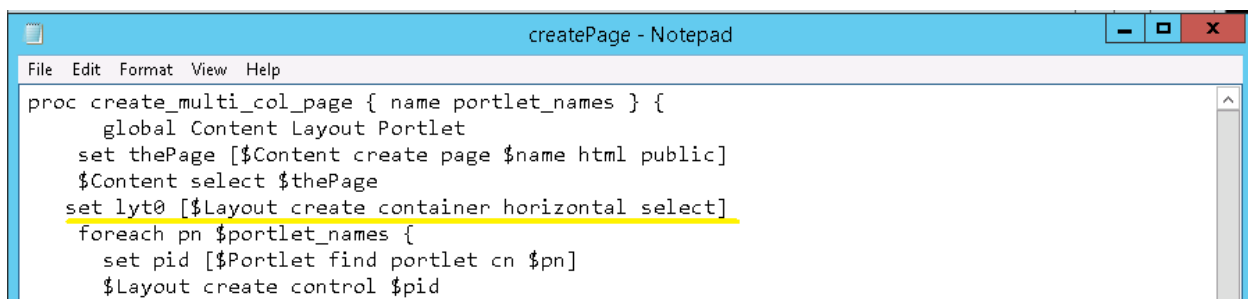
```
proc create_multi_col_page { name portlet_names } {
    global Content Layout Portlet
    set thePage [$Content create page $name html public]
    $Content select $thePage
    set lyt0 [$Layout create container horizontal select]
    foreach pn $portlet_names {
        set pid [$Portlet find portlet cn $pn]
        $Layout create control $pid
    }
    return $thePage
}
```

Next, a variable is declared and populated with the result of a create operation performed on the Content scripting-bean. This operation receives a variable declared in the function signature, "name" and two attributes. The first attribute is the markup type associated with the page and the second defines the visibility of the page. By default, all pages created with the Content scripting-bean are private. Setting the attribute to "public" creates a page available to any authorized user.



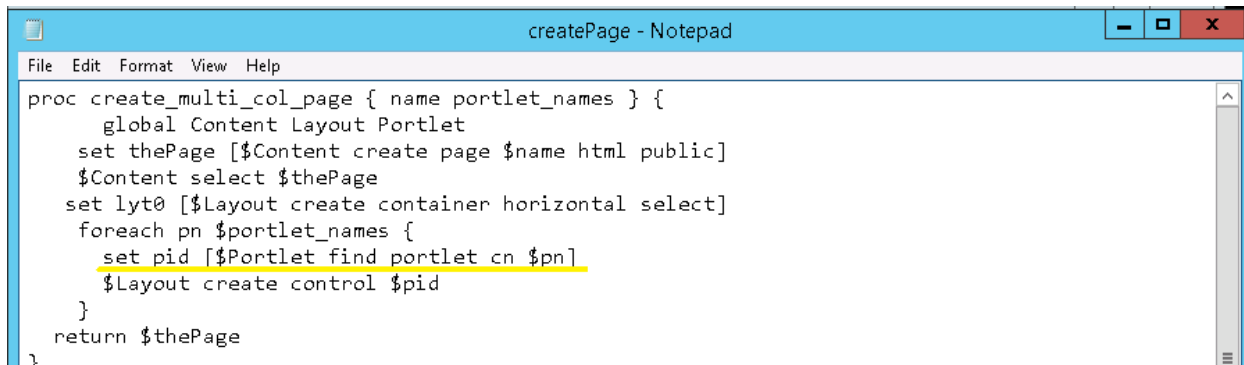
```
proc create_multi_col_page { name portlet_names } {
    global Content Layout Portlet
    set thePage [$Content create page $name html public]
    $Content select $thePage
    set lyt0 [$Layout create container horizontal select]
    foreach pn $portlet_names {
        set pid [$Portlet find portlet cn $pn]
        $Layout create control $pid
    }
}
```

The next task the script performs is to select the newly completed page. We are doing this for two reasons. First, creating a page does not automatically select the object for additional processing. Second, modifying the page layout requires another scripting-bean, Layout. In the next code sample, you will see the statement that sets a variable to the result of an action on a Layout bean in which a horizontal container is created and selected.



```
proc create_multi_col_page { name portlet_names } {
    global Content Layout Portlet
    set thePage [$Content create page $name html public]
    $Content select $thePage
    set lyt0 [$Layout create container horizontal select]
    foreach pn $portlet_names {
        set pid [$Portlet find portlet cn $pn]
        $Layout create control $pid
    }
}
```

Now that a container is defined, we can add columns, or more correctly, controls. In the snippet you will see that the controls are defined within a “foreach” loop that uses an array of portlet names that have been passed to the function. Within the body of the loop, the third of our three global variables, Portlet, is used to find portlets by common name, cn.

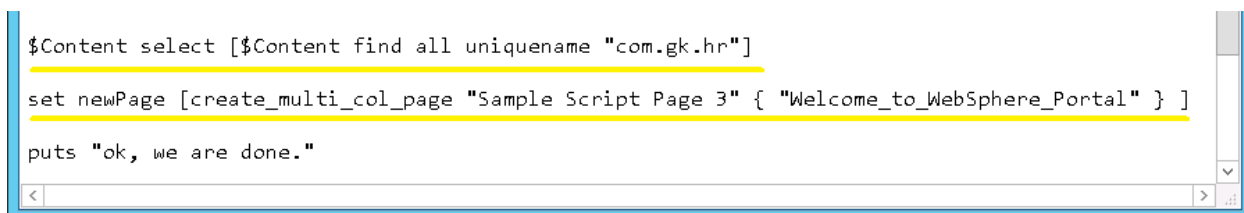


```
File Edit Format View Help
proc create_multi_col_page { name portlet_names } {
    global Content Layout Portlet
    set thePage [$Content create page $name html public]
    $Content select $thePage
    set lyt0 [$Layout create container horizontal select]
    foreach pn $portlet_names {
        set pid [$Portlet find portlet cn $pn]
        $Layout create control $pid
    }
    return $thePage
}
```

Once the portlet has been found, it is passed to the Layout scripting-bean, and a create control task is performed, effectively placing the portlet within the control. At this point the body of the loop is complete, and the function returns the newly created page as a result.

By performing the tasks in this manner, we are able to call the function as many times as needed within the larger script. We could, for example, use conditional logic to call the function or we could create another loop to create some number of pages.

A final examination reveals how simple it is to call the function:



```
$Content select [$Content find all uniquenessname "com.gk.hr"]
set newPage [create_multi_col_page "Sample Script Page 3" { "Welcome_to_WebSphere_Portal" } ]
puts "ok, we are done."
```

In this sample the administrator has selected a node in the portal page hierarchy whose unique name is “com.gk.hr.” Then, the function is called by setting the result of the function to a variable named “newPage.” Within the statement (reading from right to left) we see the array of portlet names consists of one portlet. We see that the name of the page is “Sample Script Page 3”, and we see the function “create_multi_col_page” is called. Once the function has returned a result, a “puts” statement prints a message to the console.

Of course a production quality script would perform additional tasks. These would include setting a unique name for the page, providing alternative markup support if needed, specifying additional portlet names, and providing basic error checking.

Conclusion

We briefly compared tools available to the portal administrator and identified some use cases where the scripting interface may be the tool of choice. We reviewed how to launch the tool and explored how to understand and select options for its use. After a review of a few basic commands, we were able to take a look at creating our own custom script. The use case that we took particular interest in leveraged the ability to provide discrete access to the portal node hierarchy in order to constrain the ability of a developer to introduce changes to the larger scope of portal resources.

This makes the Portal Scripting Interface a unique tool unlike the XMLAccess, which is a blunt instrument, or the portal administration portlets, which do not provide a mechanism to achieve reliably repeatable and consistent outcomes.

Learn More

Learn more about how you can improve productivity, enhance efficiency, and sharpen your competitive edge through training.

[Developing Applications for IBM WebSphere Portal 8.0 using IBM RAD 8.5](#)

[Installation and Administration of IBM WebSphere Portal 8.5.0 on Linux](#)

[WebSphere Application Server V8.5.5 Administration](#)

[WebSphere Application Server V8.5 Scripting and Automation](#)

Visit www.globalknowledge.com or call **1-800-COURSES (1-800-268-7737)** to speak with a Global Knowledge training advisor.

Resources

Additional resources are available to help you walk through any portal scripting questions or issues.

[IBM Project Support](#)

[IBM Web Content Manager Support](#)

About the Author

G. David Wilkerson is a certified IBM instructor with over 20 years industry experience with IBM Digital Experience technologies and supporting platforms such as WebSphere Application Server. He has more than 75 IBM certificates including WebSphere Application Server, IBM WebSphere Portal, IBM Domino, IBM Sametime, IBM Connections, and IBM Forms. As a practicing architect and engineer David worked with private sector companies in finance, insurance, health, and manufacturing as well with public sector organizations such as the U.S. Department of Defense, U.S. Department of the Treasury, U.S. Department of Justice, and U.S. Department of Agriculture.