PASW® Collaboration and Deployment Services 4.1 Customization Reference

SPSS Inc.233 South Wacker Drive, 11th Floor
Chicago, IL 60606-6412
Tel: (312) 651-3000
Fax: (312) 651-3668

Printed in the United States of America.

Preface

PASW Collaboration and Deployment Services is an enterprise-level application that enables widespread use and deployment of predictive analytics. PASW Collaboration and Deployment Services provides centralized, secure, and auditable storage of analytical assets, advanced capabilities for management and control of predictive analytic processes, as well as sophisticated mechanisms of delivering the results of analytical processing to the end users. The benefits of PASW Collaboration and Deployment Services include safeguarding the value of analytical assets, ensuring compliance with regulatory requirements, improving the productivity of analysts, and minimizing the IT costs of managing analytics.

Technical Support

The services of SPSS Inc. Technical Support are available to registered customers of SPSS Inc.. Customers may contact Technical Support for assistance in using SPSS Inc. products or for installation help for one of the supported hardware environments. To reach Technical Support, see the SPSS Inc. Web site at *http://www.spss.com*, or contact your local office, listed on the SPSS Inc. Web site at *http://www.spss.com/worldwide*. Be prepared to identify yourself, your organization, and the serial number of your system.

Tell Us Your Thoughts

Your comments are important. Please let us know about your experiences with SPSS Inc. products. Please send e-mail to *suggest@us.ibm.com*, or write to SPSS Inc., Attn: Director of Product Planning, 233 South Wacker Drive, 11th Floor, Chicago IL 60606-6412.

Contents

1 Customization Overview 1

Prerequisites 1

2 Deployment Portal Customization 3

Customizing the User Interface 3
Authentication Customization 5

3 URL Parameters 7

Base Path 7
Query String 7
- Common Parameters 8
- Report Parameters 14
- Scoring Parameters 16
- Custom Dialog Parameters 18

HTML Techniques 25

4 PASW Tag Library 28

JavaServer Pages Architecture 29
Supported Items 30
- Reports 30
- Jobs 31
- Scoring Models 31
- Custom Dialogs 32

Building an Application 32
Implementation Details 35
Public JavaScript API 36
- runRepositoryItem Function 36
- getBookmarkedValues Function 37
- retrievePromptValues 37

PASW Tag Library Tag Reference 38
credential Tag 38
repositoryItem Tag 41
repositoryItemPrompt Tag 45
report Tag 46
reportPrompt Tag 46
outputLocation Tag 46
sourceLinkPrompt Tag 47
sourceLinkRepositoryItem Tag 50
sourceLinkReport Tag 50
sourceLinkVariable Tag 51
actionHandler Tag 52
actionParameter Tag 53
Tag Library Beans 53
Credential Bean 54
ReportBean Bean 54
SearchBean Bean 55
PevMetaDataBean Bean 56
ScoringBean Bean 57
JavaServer Pages Samples 58

5 Portal Integration *60*

Installation 61
Configuration 62

6 Scripting *67*

Command Line Scripting 67
Global Keywords 67
Repository Connections 68
Content Repository Functions 68
Process Management Functions 86
API Reference 89
Content Repository APIs 90
Process Management APIs 124

7 HTML Archive 132

File Structure 132
Creating HTMLC Files 133
Custom HTMLC File Example 133

8 Customization Example 134

PASW Tag Library 134
Report Definitions 134
Running PASW BIRT Report Designer Reports 135
Running Visualization Reports 138
Javascript API 138
Visualization Report Interactivity 138
URL Fragments 139
Tab Extension Framework 140

Index 142

Chapter
1

Customization Overview

Deployment Portal serves as a thin-client interface into the repository, allowing any user with a browser and valid credentials to work with content stored within the repository. However, the default appearance and functionality may not be optimal for all users. For example, you may want to modify the appearance of the browser interface to better match a corporate standard. Alternatively, you may wish to create your own interface to repository content.

PASW Collaboration and Deployment Services offers a variety of approaches for customizing the interaction with content stored in the repository.

- Modify package components, such as images and stylesheets, to control the Deployment Portal appearance. For more information, see *Deployment Portal Customization*.
- Reference repository content directly using uniform resource locators (URL) parameters. For more information, see *URL Parameters*.
- Create custom web pages based on information obtained from reports and queries stored in the repository using Java Server Page tags. For more information, see *PASW Tag Library*.
- Embed repository content, such as reports, on portal pages. For more information, see *Portal Integration*.
- Perform batch processing of repository content using Python scripting. For more information, see *Scripting*.

Prerequisites

For proper processing of custom dialogs, the following requirements must be satisfied:

- A PASW Statistics server must be set up in Deployment Manager and then designated as the default server for executing custom dialog syntax using browser-based Deployment Manager. It is also possible to configure individual custom dialogs to use a PASW Statistics server different from the system default.
- The user must be assigned the *Run Custom Dialogs* action to be able to execute custom dialogs.
- PASW Statistics save file access is enabled by PASW Statistics Data File Driver Service, which must be installed, started, and then designated as the driver for PASW Statistics data using browser-based Deployment Manager. The software is available as a download to SPSS Inc. customers.

Important! PASW Statistics Data File Driver Service must run on host with the same operating system type as the repository host. For example, it is impossible to use PASW Collaboration and Deployment Services running on a Linux server in conjunction with PASW Statistics Data File Driver Service running on a Windows server.

For information about PASW Collaboration and Deployment Services system configuration and actions, see the administrator's documentation.

Chapter 2

Deployment Portal Customization

Administrators can customize certain elements of the Deployment Portal user interface by modifying various files in a repository package (*peb-webcontent.package*) and redeploying the package with the Package Manager utility. Experience with stylesheets (*.css*) is recommended.

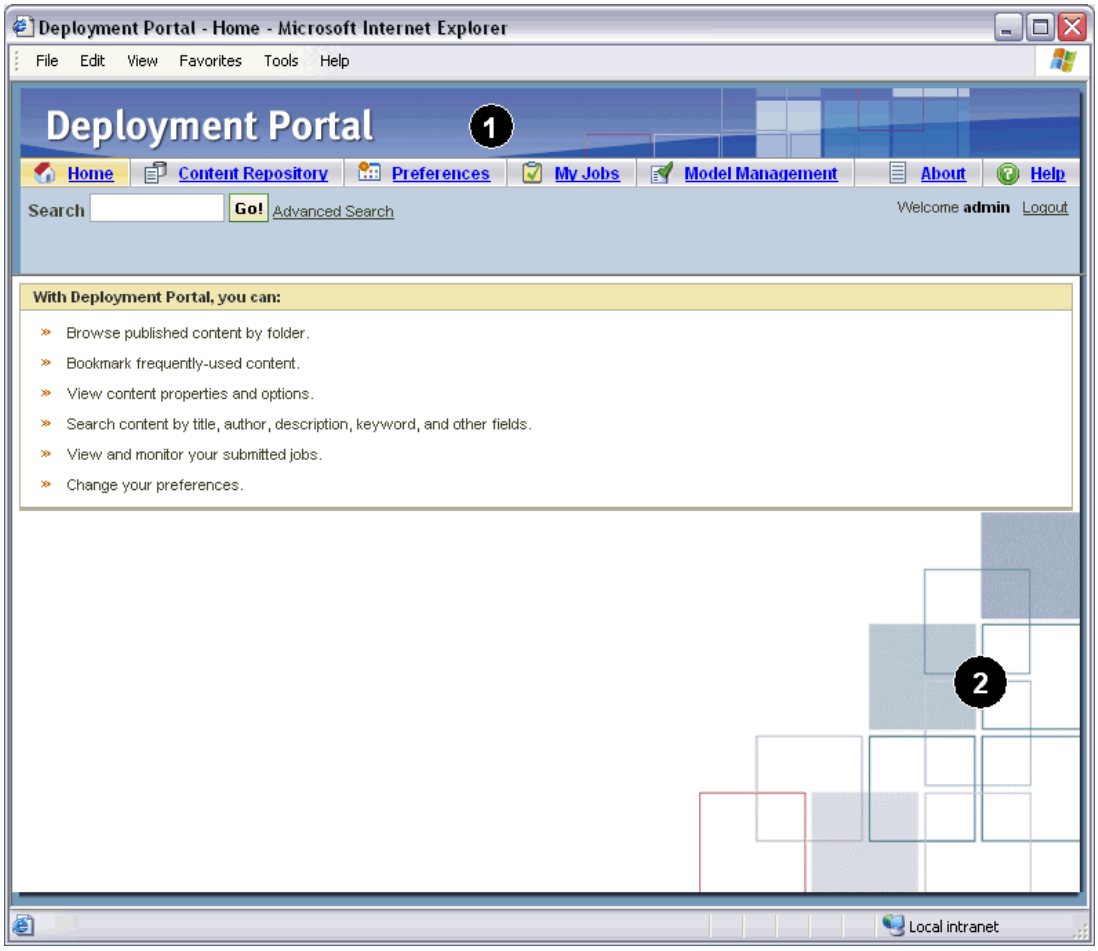
In addition, the system can be configured to use a custom authentication mechanism, eliminating the need to manually enter credentials when accessing the Deployment Portal.

Customizing the User Interface

To customize the Deployment Portal user interface:

1. In the *staging* directory of your repository installation, copy the file *peb-webcontent.package* and store it in a separate directory. Use this file as a back up if you want to revert to the default user interface in the future.
2. In the *staging* directory of your repository installation, open *peb-webcontent.package* with a file compression utility such as WinZip® and extract its contents to a temporary directory.
3. In the extracted *images* directory, modify or replace any of the following graphics as desired:
   - *headerbanner.gif*: The main banner on the top of the Deployment Portal user interface containing the company logo and product name. See Figure 2-1, #1. To display properly, the banner should be a maximum of 50 pixels tall.
   - *bottombackground.gif*: The grid-style graphic in the bottom-right corner of the Deployment Portal Home screen. See Figure 2-1, #2.
   - *floatingsquares.gif*: The grid-style graphic in the bottom-left corner of the Deployment Portal Login screen.

Figure 2-1
Default user interface



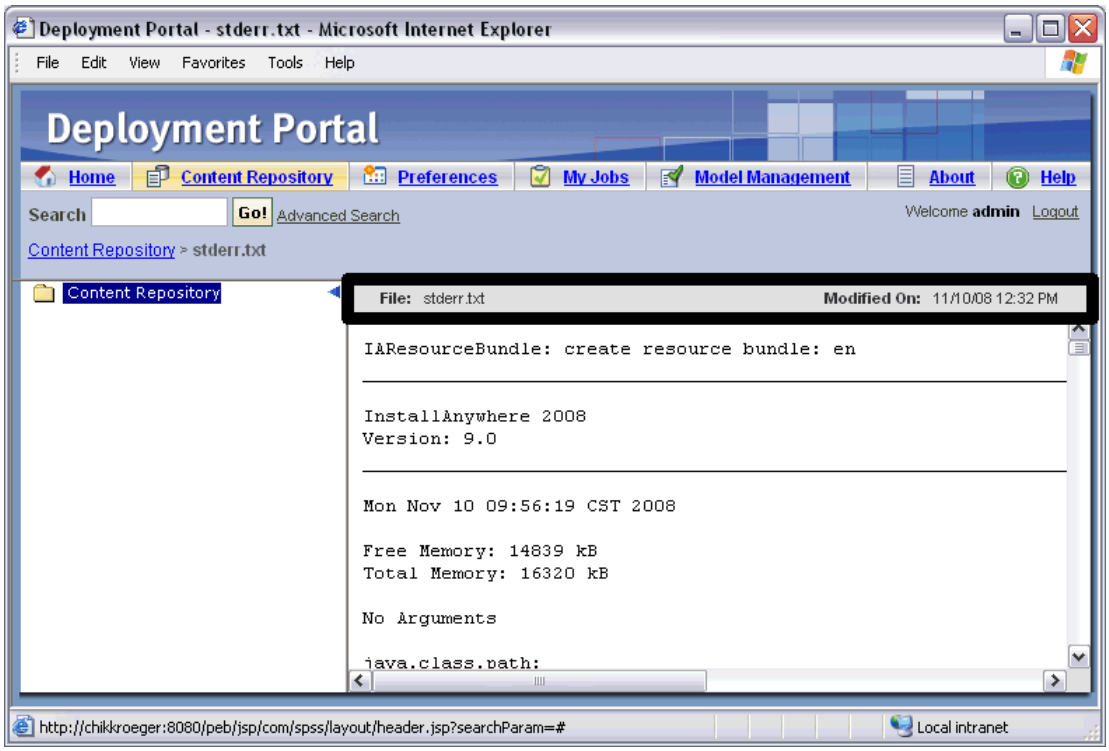
4. In the extracted *config* directory, open the *UIConfig.xml* file in a simple text editor such as Notepad. Modify settings to suppress certain elements of the user interface as desired. When finished, save and close the file. Following are common elements to suppress.

   - `Footer`: The bottom footer bar containing the "Powered by" logo (suppressed by default). Change value from `false` to `true` to display.
   - `FileHeader`: The gray file information bar containing the file name and date/time last modified, displayed when viewing a document. See Figure 2-2. Change value from `true` to `false` to suppress.

```
<component-configuration>
    <component-name>Footer</component-name>
    <component-visibile>false</component-visibile>
</component-configuration>

<component-configuration>
    <component-name>FileHeader</component-name>
    <component-visibile>true</component-visibile>
</component-configuration>
```

Figure 2-2
File header



5. In the extracted *css* directory, open the *spssStyles.css* file in a simple text editor such as Notepad. Modify style settings as desired. When finished, save and close the file.

6. Using a file compression utility such as WinZip, compress all folders and customized files that were extracted previously. Save the file as *peb-webcontent.package* (be sure to use a *.package* file extension, not default *.zip*).

7. Stop the repository server.

8. Use the Package Manager utility to install the customized *peb-webcontent.package* file you saved. For instructions, see the configuration documentation. When finished, log out and close Package Manager.

9. Start the repository server.

10. Wait several minutes and open Deployment Portal (*http://<hostname>:<port>/peb*) to verify your changes.

Authentication Customization

If single sign-on is configured and enabled at your site, manual entry of security credentials is not necessary for accessing Deployment Portal. However, if single sign-on is not enabled, the system can still be customized to avoid manual credential entry.

The Deployment Portal architecture includes a Java interface named AuthenticationCriteriaInterface. This interface includes the following methods:

```
public String getUserName()
public String getPassword()
public String getProvider()
```

To customize authentication, you must first create a Java class that implements this interface, such as *com.spss.AuthenticationCriteriaImpl*. Your class must provide the username, password, and provider authentication information. These values may be supplied by a variety of sources, such as a previously authenticated intranet site or portal. The class should be placed in the classpath for the application server.

Next, configure the system to use your class. In the browser-based Deployment Manager, modify the following Deployment Portal configuration settings:

- *Configured Authentication Criteria Class*. Supply the name of your custom class.
- *Use Configured Authentication Criteria*. Select this option to enable the use of your class.

After configuring you authentication class, users should be able to enter the Deployment Portal without entering credentials, provided that the credentials supplied by the custom class are valid. In addition, URL references to repository items will not need to include credential information.

Chapter
3

URL Parameters

You can access Deployment Portal reports and other repository objects using direct URLs (Uniform Resource Locators). With URLs, you can directly share reporting information in different ways such as embedding reporting into your external web sites and applications. This reference document lists various Deployment Portal URL parameters and contains some tips for building and using Deployment Portal URL query strings. For assistance, contact Technical Support.

The URL parameters outlined in this document are unrelated to the URLs available in Deployment Manager and described in the Deployment Manager documentation.

Base Path

The base path for all requests is `http://hostname:port/peb/view/<content repository path>` OR `http://hostname:port/peb/view?id=<object-id>`

where:

- `hostname` is the name or IP address of server Deployment Portal is installed on
- `port` is the port number
- `<content repository path>` is the resource path of the repository object on which to act
- `<object-id>` is the resource ID of the repository object on which to act

Examples

```
http://yourserver:8080/peb/view/sample/employee.dbq
```

```
http://yourserver:8080/peb/view?id=0a58c3461e885d240000010f4cc607188375
```

Query String

The base path for the URL reference can be followed by a query string containing parameters that provide additional processing information. The query string begins with a question mark and contains parameter/value pairs separated by ampersands (&). Note that if a repository item is referenced by its resource identifier, the question mark initiating the query string is already present for the `id` parameter and should not be repeated for any other parameters.

At a minimum, a URL must contain the content repository path in the base path or the `id` parameter. Other parameters are optional. Unless otherwise stated, parameters and their values are case sensitive. Some parameters, such as username and password, are used in virtually all URL queries, while the use of other parameters may depend on the type of item being referenced in the query. Note that the system can be configured to use a custom authentication mechanism

to eliminate the need to supply security credential parameters in the query string. For more information, see the topic Authentication Customization in Chapter 2 on p. 5.

Reserved characters like & and excluded US-ASCII characters like # should be URL encoded before being specifying as a parameter value in the query string. However, characters in the reserved set are not reserved in all contexts. In general, a character is reserved if the semantics of the URI changes if the character is replaced with its escaped US-ASCII encoding. Hence some characters (like ?, =, and :) are not reserved in the parameter values, but characters like & and # are, and hence need to be URL encoded.

For example, the & character should be URL encoded as %26. Thus, the following URL:

http://yourserver:8080/peb/view/sample/employee.dbq?username=testuser&admin

should be specified as

http://yourserver:8080/peb/view/sample/employee.dbq?username=testuser%26admin

The following sections describe each parameter.

Common Parameters

Common parameters are used in virtually all URL references, or are used across multiple types of repository items.

id

The id parameter specifies the repository identifier for the item on which to act.

Syntax

```
id=<identifier>
```
where <identifier> is the repository object identifier

Examples

```
http://yourserver:8080/peb/view?id=0a58c3461e885d240000010f4cc607188375
```

version

The version parameter specifies the version of the repository object on which to act. Special characters, such as spaces, must be escaped. Eliminate this parameter to display the LATEST version.

Syntax

```
version=m.<version marker>
```
where <version marker> is the version of the repository object.

OR

version=l.<label>
where <label> is the version label of the repository object.

Examples

```
http://yourserver:8080/peb/view/sample/employee.dbq
?version=m.1:2006-12-04%2020:39:17.995

http://yourserver:8080/peb/view/sample/employee.dbq
?version=l.firstVersion
```

username

The username parameter specifies the user ID with which to log in to Deployment Portal.

Syntax

username=user_ID
where user_ID is the user ID of the person logging in to Deployment Portal.

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq?username=validUser
```

password

The password parameter specifies the password with which to log on to Deployment Portal.

Syntax

password=password
where password is the password of the person logging on to the server.

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq?username=validUser&password=pass
```

provider

The provider parameter specifies the security provider with which to log on to Deployment Portal. A value for provider must be specified if the username and password parameters are used.

Syntax

provider=<provider>
where <provider> is the security provider for Deployment Portal. Valid values include:

- *Native* for the built-in provider
- *AD/<domain>* for Active Directory, where *<domain>* corresponds to the DNS namespace

- *ADL/<domain>* for Active Directory with local override, where *<domain>* corresponds to the DNS namespace
- *iSeries* for iSeries
- *devldapOpenLDAP* for OpenLDAP

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq
?username=validUser&password=pass&provider=Native
```

promptstate

The `promptstate` parameter specifies whether to suppress the runtime prompt dialog for prompted variable values that are not specified in the query string.

Syntax

`promptstate=x`
where:
`1` will *suppress* the runtime prompt dialog and use the specified default variable value for any prompted variables that are not specified.
`2` will *display* the runtime prompt dialog for any prompted variables that are not specified.
Alternately, you can eliminate this parameter to allow the prompt dialog to be displayed.

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq?&username=validUser
&password=pass&provider=Native&fragment=true&outputtype=html
&var_EmployeeID=1&promptstate=1
```

waitstate

The `waitstate` parameter specifies whether to suppress the Wait screen while a report is running.

Syntax

`waitstate=x`
where `1` will suppress the Wait screen. Eliminate this parameter to display the Wait screen.

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq?username=validUser
&password=pass&provider=Native&fragment=true&outputtype=html
&var_EmployeeID=1&promptstate=1&waitstate=1&fragment=true
```

partId

The `partId` parameter identifies a specific part of the repository object being referenced. For HTMLC files, this parameter can reference a specific file within the archive. For PASW Statistics output files (*.spw*), the parameter corresponds to the index as shown in the outline for the file. For example, to get the first part, specify `partId=0`.

Syntax

```
partId=<reference_id>
```

where `<reference_id>` is either:

- the relative path and name of a file within an HTMLC file
- the index of the desired output within an *.spw* file

Example

```
http://yourserver:8080/peb/view/output.htmlc?username=validUser
&password=pass&provider=Native&partId=img/chart.png

http://yourserver:8080/peb/view/output.spw?username=validUser
&password=pass&provider=Native&partId=1
```

outputtype

The `outputtype` parameter specifies the file type of the result set.

Syntax

```
outputtype=file_type
```

where file_type corresponds to one of the values in the following table.

Table 3-1
Output types

Report Type	Value	Returns
Showcase	*html*	HTML
	pdf	Portable Document Format
	wk4	Lotus 1-2-3
	text	text
	csv	comma separated values
	biff8	Microsoft Excel
	xlsx	Microsoft Excel 2007 XML format
	rptdocument	PASW BIRT Report Designer report document

Report Type	Value	Returns
PASW BIRT Report Designer	*HTML*	HTML
	Excel 97/2003	Microsoft Excel
	PowerPoint	Microsoft PowerPoint
	Word Document	Microsoft Word
	PDF	Portable Document Format
	PDF - Fit to Page Width	Portable Document Format using width magnification
	PDF - Page Break Pagination Only	Portable Document Format using page break pagination
	PDF - Fit to Whole Page	Portable Document Format using page magnification
	PostScript	PostScript format
	PostScript - Fit to Page Width	PostScript format using width magnification
	PostScript - Page Break Pagination Only	PostScript format using page break pagination
	PostScript - Fit to Whole Page	PostScript format using page magnification
	BIRT RPT Document	PASW BIRT Report Designer report document
	ask	a prompt for the user at runtime to specify an output format
Visualization	*png*	Portable Network Graphics format
	emf	Enhanced Metafile format
	jpeg	JPEG
	html	HTML. This is a valid output format for visualization reports only when the output is a table. If HTML is specified as the format for a visualization report that does not produce a table, the output is converted to a PNG image.
	pdf	PDF
	ask	a prompt for the user at runtime to specify an output format
Custom dialogs	*SPW*	PASW Statistics web output viewer
	HTML	HTML

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq?username=validUser
&password=pass&provider=Native&fragment=true&outputtype=html
```

format

The `format` parameter specifies whether to return the original file stored to the repository, rather than running the file.

Syntax

```
format=raw
```

where `raw` will return the original file. For example, in the case of a ShowCase Query definition, using the `format` parameter will download the original *.*dbq* file instead of running the query dynamically.

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq?username=validUser
&password=pass&provider=Native&format=raw
```

fragment

The `fragment` parameter specifies whether to display the Deployment Portal user interface elements (i.e., header, footer, Content Repository tree) with the report results.

Syntax

```
fragment=true
```

where `true` will suppress the Deployment Portal interface elements. Eliminate this parameter to display the Deployment Portal interface.

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq?username=validUser
&password=pass&provider=Native&fragment=true
```

Parameters for Variables

For non-report repository items that use variables, such as jobs, the value for a variable can be specified by including the variable name and value in the URL query string. For custom dialogs, jobs, and scoring, variable value prompts will appear for all variables or no variables, depending on the value of the parameter.

For report items, the variable name must be preceded by the `var_` prefix. For more information, see the topic Parameters for Variables on p. 16.

Syntax

```
<variable>=<value>
```

where:

- `<variable>` is the name of the variable to satisfy
- `<value>` is the value to use to satisfy the specified report variable

Example

```
http://yourserver:8080/peb/view/sample/myJob?username=validUser
&password=pass&provider=Native&region=1
```

Report Parameters

Report parameters are used in references to reports stored within the repository. The reports may be visualization reports, PASW BIRT Report Designer reports, or ShowCase reports.

dbcredential_datasourcename

The `dbcredential_datasourcename` parameter specifies the credential with which to log on to the data source. This is used if the data source user ID differs from the Deployment Portal user ID.

Syntax

`dbcredential_datasourcename=<credential id>`
where `datasourcename` is the name of the given data source and `<credential id>` is the id of the credential object to be used for logging on to the data source.

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq
?dbcredential_yourDS=0a58c346cd5b72010000010f3df6d5e28130
```

dbuser_datasourcename

The `dbuser_datasourcename` parameter specifies the user ID with which to log on to the data source. This is used if the data source user ID differs from the Deployment Portal user ID.

Syntax

`dbuser_datasourcename=user_ID`
where `datasourcename` is the name of the given data source and `user_ID` is the user ID of the person logging on to the data source.

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq?dbuser_yourDS=sa
```

dbpwd_datasourcename

The `dbpwd_datasourcename` parameter specifies the password with which to log on to the data source. This is used if the data source user ID differs from the Deployment Portal user ID.

Syntax

`dbpwd_datasourcename=password`
where `datasourcename` is the name of the given data source and `password` is the password of the person logging on to the data source.

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq
?dbuser_yourDB=sa&dbpwd_yourDB=sa
```

Note

If the `dbcredential_datasourcename` parameter has been specified, then that parameter will be considered for logging on to the data source before the `dbuser_datasourcename` and `dbpwd_datasourcename` parameters.

width

The width parameter specifies width of the resulting image or graph. This parameter is used specifically with visualization reports.

For reports containing height and width specifications, both height and width parameters must be provided. If either parameter is missing, the graph would be rendered with its default height and width.

Syntax

`width=x`
where x is the integer value in pixels.

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq?username=validUser
&password=pass&provider=Native&fragment=true&outputtype=html
&var_EmployeeID=1&promptstate=1&waitstate=1&width=500&height=1000
```

height

The height parameter specifies height of the resulting image or graph. This parameter is used specifically with visualization reports.

For reports containing height and width specifications, both height and width parameters must be provided. If either parameter is missing, the graph would be rendered with its default height and width.

Syntax

`height=x`
where x is the integer value in pixels.

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq?username=validUser
&password=pass&provider=Native&fragment=true&outputtype=html
&var_EmployeeID=1&PROMPTSTATE=1&waitstate=1&width=500&height=1000
```

var_variable

The `var_variable` parameter specifies the value to use to satisfy the specified report variable.

Syntax

`var_variable=value`
where:
`variable` is the name of the variable to satisfy. To locate the variable name, in ShowCase Query or Report Writer, from the Query menu, select Variables. A list of variable names for the current report is displayed.
`value` is the value to use to satisfy the specified report variable

Example

```
http://yourserver:8080/peb/view/sample/employee.dbq?username=validUser
&password=pass&provider=Native&fragment=true&outputtype=html&var_EmployeeID=1
```

Notes

- For reports, specifying a variable value on the URL will suppress the runtime prompt for that variable.
- To specify a single variable value (=), use the syntax `var_Lastname=Curtis`
- To specify multiple variable values (`IN`), use the syntax `var_Lastname=Curtis&var_Lastname=McLind`
- To specify a range of variable values (`BETWEEN`), use the syntax `var_Dateship=3-1-2007&var_Dateship=3-31-2007`
- To specify values for multiple variables, use the syntax `var_Lastname=Curtis&var_Dateship=3-1-2007&var_Dateship=3-31-2007`

Scoring Parameters

Scoring parameters are used when referencing scoring configurations to generate scores.

dataset

The dataset parameter specifies the location of a SQL data provider definition that will be used for batch scoring. The value of this parameter will be a relative path within the repository.

Syntax

`dataset=dpd_location`
where `dpd_location` is the path to the data provider definition in the repository.

Example

```
http://yourserver:8080/peb/view/myPMML.xml?username=validUser
&password=pass&scoring_configuration=testConfig
```

```
&dataset=/datasets/dataset.sqldpd
```

dataset_label

The dataset_label parameter allows the user to specify the appropriate version of the dataset. The specified dataset version must be compatible with the data provider defined in the scoring configuration. If not specified, the *LATEST* version is used.

Syntax

```
dataset_label=myLabel
```
where `myLabel` is the label for the desired dataset version.

Example

```
http://yourserver:8080/peb/view/myPMML.xml?username=validUser
&password=pass&scoring_configuration=testConfig
&dataset=/datasets/dataset.sqldpd&dataset_label=PRODUCTION
```

dataset_rowlimit

The user may limit the amount of data processed from the dataset for batch scoring. This will help prevent long running processes. The dataset_rowlimit specifies the number of rows of data that will be extracted from the dataset.

Syntax

```
dataset_rowlimit=x
```
where `x` is the number of dataset rows to be extracted.

Example

```
http://yourserver:8080/peb/view/myPMML.xml?username=validUser
&password=pass&scoring_configuration=testConfig
&dataset=/datasets/dataset.sqldpd&dataset_rowlimit=1000
```

scoring_configuration

The scoring_configuration parameter specifies the scoring configuration used by the scoring engine to score the specified model.

Syntax

```
scoring_configuration=configName
```
where `configName` is the name of scoring configuration to use for scoring. The specified configuration must be able to process a scoring request. A reference to a suspended configuration will be unable to produce scores.

Example

```
http://yourserver:8080/peb/view/myPMML.xml?username=validUser
&password=pass&scoring_configuration=testConfig
&dataset=/datasets/dataset.sqldpd
```

batch_type

The batch_type parameter specifies which scoring input prompts should be displayed. If the parameter specifies *dataset*, the scoring interface will generate the input prompts for the dataset and label. If the batch_type is not specified and parameter inputs are not defined, the interface based on scoring parameters is used.

Syntax

```
batch_type=inputPrompt
```

where `inputPrompt` indicates the source for the input prompts. Currently, the only supported source is *dataset*. Omit this parameter to prompt the user for input values based on parameters.

Example

```
http://yourserver:8080/peb/view/myPMML.xml?username=validUser
&password=pass&scoring_configuration=testConfig&batch_type=dataset
```

Custom Dialog Parameters

Custom dialog parameters are used when referencing custom dialog (*.spd*) files.

Note: This functionality requires PASW Statistics adapters in the PASW Collaboration and Deployment Services environment. For more information, see the PASW Statistics installation documentation.

dataset.uri

The URI of the dataset to be use by the custom dialog. For DPDs and *.sav* files in the repository, the URI can be specified as a repository path or the resource ID. When the URI references a file on the file system, the path to the file must be valid from the PASW Statistics Data File Driver server that is used to retrieve the variable metadata. It must also be a valid path on the PASW Statistics Server that will execute the syntax. If a repository dataset object is used, the version of the object can be appended to the URI either as a version maker or a label.

Syntax

```
dataset.uri=myURI
```

where *myURI* is the URI for the dataset.

Example

```
http://yourserver:8080/peb/view/myDialog.spd
```

```
?dataset.uri=spsscr:///Datasets/SpecificURI.sav

http://yourserver:8080/peb/view/myDialog.spd
?dataset.uri=spsscr:///?id=0a30063bc975ede40000011cafb8deda8327.

http://yourserver:8080/peb/view/myDialog.spd
?dataset.uri=file:///C:/Program%20Files/SPSSInc/Samples/accidents.sav
```

dataset.table

For Enterprise View data sources, the table to be used by the custom dialog. If no name is specified, the user will be prompted to select from the list of tables available in the DPD.

Syntax

```
dataset.table=myTable
```

where *myTable* is the table to use.

Example

```
http://yourserver:8080/peb/view/myDialog.spd
?dataset.uri=spsscr:///DPDs/myDPD&dataset.table=myTableName
```

dataset.prompt

Indicates that the user will be forced to select a dataset for the custom dialogs. Otherwise, the dataset selected for the first dialog opened by the user that contains matching search criteria during a session will be used for any subsequent custom dialogs that are not configured to use a specific dataset.

Syntax

```
dataset.prompt=indicator
```

where *indicator* is either *true* or *false*.

Example

```
http://yourserver:8080/peb/view/myDialog.spd?dataset.prompt=true
```

dataset.search.criteria

Search criteria to be used for generating a list of data sets at run time. The entire search string must be entered on a single line. Multiple conditions may be combined using parenthesis and and/or logic.

Search criteria

$$repository/title_field_name=<Object name>

$$search/mimetype=<Object MIME type>

$$repository/version_created_by_field=<Created by user stamp>

$$repository/version_created_date_field=<Version created date>

$$repository/description_field_name=<Object description>

$$repository/object_last_modified_by=<Created by user stamp>

Syntax

```
dataset.search.criteria=myCriteria
```

where *myCriteria* is the search expression.

Example

```
    # locates all DPDs
http://yourserver:8080/peb/view/myDialog.spd
?dataset.search.criteria='$$search/mimetype%3Dapplication/x-vnd.spss-data-provider'

# locates all SAV files
http://yourserver:8080/peb/view/myDialog.spd
?dataset.search.criteria='$$search/mimetype%3Dapplication/x-vnd.spss-spss-data%20or%20
$$search/mimetype%3Dapplication/x-vnd.spss-statistics-data'

# locates all files that match the keyword SPECIAL_DATASET
http://yourserver:8080/peb/view/myDialog.spd
?dataset.search.criteria='$$repository/keyword_field_name%3D%3DSPECIAL_DATASET'
```

variable.display

Whether to show variable names or labels.

Syntax

```
variable.display=<type>
```

where <*type*> is either:

- *names* to show variable names
- *labels* to show variable labels

Example

```
    http://yourserver:8080/peb/view/myDialog.spd
?dataset.uri=spsscr:///Datasets/SpecificURI.sav&variable.display=labels
```

variable.sort

The sort criterion used for ordering variables.

Syntax

```
variable.sort=<myCriteria>
```

where <*myCriteria*> is:

- *none* to do no additional sorting beyond the original order in the data
- *alphanumeric* for an alphanumeric sort of field names or labels, whichever is displayed
- *measurement* to sort by the field measurement levels

Example

```
http://yourserver:8080/peb/view/myDialog.spd
?dataset.uri=spsscr:///Datasets/SpecificURI.sav&variable.sort=alphanumeric
```

stylesheet.url

If you are using a CSS style sheet stored in the repository, the repository URL of the style sheet.

Syntax

```
stylesheet.url=myURL
```

where *myURL* is the URL for the style sheet.

Example

```
http://yourserver:8080/peb/view/myDialog.spd
?stylesheet.url=/peb/view/EditBox_pes.css&fragment=true
```

stylesheet.name

If you are using a CSS style sheet embedded in the custom dialog file, the name of the style sheet. The style sheet file can be added to the custom dialog file using compressed archive software, such as WinZip.

Syntax

```
stylesheet.name=myStyles.css
```

where *myStyles.css* is the name of the style sheet.

Example

```
http://yourserver:8080/peb/view/myDialog.spd
?stylesheet.name=EditBox.css
```

javascript.url

If you are using a JavaScript stored in the repository, the repository URL of the script file.

Syntax

```
javascript.url=myURL
```

where *myURL* is the URL for the JavaScript file.

Example

```
http://yourserver:8080/peb/view/myDialog.spd
?javascript.url=/peb/view/EditBox_pes.js&fragment=true
```

javascript.name

If you are using a JavaScript sheet embedded in the custom dialog file, the name of the script file.

Syntax

```
javascript.name=myFile
```

where *myFile* is the name of the JavaScript file.

Example

```
http://yourserver:8080/peb/view/myDialog.spd?javascript.name=EditBox.js
```

validate.method

A validation method from the specified JavaScript file to call before a page is submitted. The form that is being submitted should be the only parameter for the method. Upon evaluating the form input, the method should return a Boolean value. The method should return true if everything is valid and false if the submit should be cancelled.

Syntax

```
validate.method=myMethod
```

where *myMethod* is the name of the method in the JavaScript file to use for validation.

Example

```
http://yourserver:8080/peb/view/myDialog.spd?javascript.name=EditBox.js
&validate.method=myValidate
```

output.format

The format of the output to create. Default format is PASW Statistics Web Output viewer format (*.spw*. In some cases, it may be appropriate to create HTML instead. The output format is case sensitive.

This parameter specifies the same information as the `outputtype` parameter, but is honored only for custom dialogs.

Syntax

```
output.format=myFormat
```

where *myFormat* is the format for the output. Valid values include:

- *SPW* for PASW Statistics web output viewer
- *HTML* for HTML output

Example

```
http://yourserver:8080/peb/view/myDialog.spd?output.format=SPW
```

output.filename

The name of the output file. If not specified, the output file will be generated with the same name as the custom dialog file name but without the *.spw* extension.

Syntax

```
output.filename=myFile
```

where *myFile* is the name for the output file.

Example

```
http://yourserver:8080/peb/view/myDialog.spd
?output.filename=MyOutputName.spw
```

showOutline

Indicates whether the outline should be displayed. Default is true.

Syntax

```
showOutline=indicator
```

where *indicator* is either *true* or *false*.

Example

```
http://yourserver:8080/peb/view/myDialog.spd?showOutline=true
```

allowPivoting

Indicates whether table manipulation should be allowed. When the option is disabled, the user will not be allowed to pivot, flip, or change layers, save views or open data in a new window. Default is true.

Syntax

```
allowPivoting=indicator
```

where *indicator* is either *true* or *false*.

Example

```
http://yourserver:8080/peb/view/myDialog.spd?allowPivoting=true
```

allowPrinterFriendly

Indicates whether the printer friendly display can be opened for a particular table. Default is true.

Syntax

```
allowPrinterFriendly=indicator
```

where *indicator* is either *true* or *false*.

Example

```
http://yourserver:8080/peb/view/myDialog.spd?allowPrinterFriendly=true
```

allowDownload

Indicates whether the data can be downloaded to a local data file. Default is true.

Syntax

```
allowDownload=indicator
```

where *indicator* is either *true* or *false*.

Example

```
http://yourserver:8080/peb/view/myDialog.spd?allowDownload=true
```

showLogs

Indicates whether log entries should be shown in the output. Default is true.

Syntax

```
showLogs=indicator
```

where *indicator* is either *true* or *false*.

Example

```
http://yourserver:8080/peb/view/myDialog.spd?showLogs=true
```

statistics.server

PASW Statistics server used to execute the syntax of the custom dialog. The value may be a URI or a name that references a server defined in PASW Collaboration and Deployment Services. If you have multiple servers, this value can specify the URI or name of a PASW Collaboration and Deployment Services cluster.

Syntax

```
statistics.server=serverIdentifier
```

where *serverIdentifier* identifies the server to use for execution.

Example

```
http://yourserver:8080/peb/view/myDialog.spd?
statistics.server=spsscr:///?id=0a30063bc975ede40000011cafb8deda8327

http://yourserver:8080/peb/view/myDialog.spd
?statistics.server=localStatisticsServer

http://yourserver:8080/peb/view/myDialog.spd
?statistics.server=copServerCluster
```

statistics.server.credential

The credential that should be used to connect to the PASW Statisticsserver when executing syntax. The value may be a URI or a name that references a PASW Collaboration and Deployment Services credential.

Syntax

```
statistics.server.credential=myCredential
```

where *myCredential* identifies the credential under which execution occurs.

Example

```
http://yourserver:8080/peb/view/myDialog.spd?statistics.server=localStatisticsServer
&statistics.server.credential=spsscr:///?id=0a30063bc975ede40000011cafb8deda8327.

http://yourserver:8080/peb/view/myDialog.spd?statistics.server=localStatisticsServer
&statistics.server.credential=administrator
```

HTML Techniques

Use an HTML editor

Many HTML editors can simplify the creation of URL query strings and insert the proper delimiters between parameters.

Use HTML forms to submit requests

Deployment Portal requests can be submitted from HTML forms included on a web page. For example, a form can be used to allow a user to:

- Select from a list of available reports
- Select an output file type
- Specify prompted variables prior to submitting the report request
- Supply an ID and password prior to running a report

The following example references a custom dialog file in the action for a form.

```
<form name='AnalyzeOptions' method='POST' target='Iframe_1'
   action='/peb/view/SamplesStatistics/SPD/Simple.spd?fragment=true&promptstate=1&waitstate=1'>
   <input type='hidden' name='username' value='userA'/>
   <input type='hidden' name='password' value='passwordA/>
   <input type='hidden' name='provider' value='Native'/>
   <input type='hidden' name='dataset.uri' value='spsscr:///SamplesStatistics/SAV/multipleResponseData.sav'/>
   <input type='hidden' name='allowPivoting' value='false'/>

   <input name='PromptParameter1' type='checkbox' value='true'/>
   Check the box to select parameter 1

   <br>
   <input type='submit' value='Run Report' />
</form>
```

Use the repository to store custom web pages containing relative paths

The repository can be used as a central location for storing all files for a custom web site. Relative or absolute paths can be used within the custom web site to link to items such as *.css* style sheets, images, Deployment Portal reporting objects, and JavaScript.

For example, you might store a folder called *MyWebPage* in the repository containing a custom web page called *MyWebPage.htm* and resources such as images, stylesheets, and JavaScript files. *MyWebPage.htm* can contain **relative** references to the resources such as the following:

```
<img src="MyLogo.gif?fragment=true">
<script language="javascript" src="MyJS.js?fragment=true">
</script>
<LINK REL="StyleSheet" HREF="MyStyles.css?fragment=true"
TYPE="text/css" MEDIA="screen" />
```

Note that for such relative references to work properly, the web page needs to be accessed using the parameter `fragment=true` in the URL. For example:

```
http://yourserver:port/peb/view/MyWebPage/MyWebPage.htm?
username=validUser&password=pass&provider=Native&fragment=true
```

If you want to store the resources for your web site in a different repository location from where your web page is stored, they can be referenced from your web page (for example, *MyWebPage.htm*) using **absolute** paths as follows:

```
<img src="/peb/view/MyWebPage/images/MyLogo.gif?fragment=true">
<script language="javascript"
src="/peb/view/MyWebPage/js/MyJS.js?fragment=true">
</script>
<LINK REL="StyleSheet"
HREF="/peb/view/MyWebPage/CSS/MyStyles.css?fragment=true"
TYPE="text/css" MEDIA="screen" />
```

Or, they can be referenced by using the full host name and port in the path:

```
<img src="http://yourserver:8080/peb/view/MyWebPage/images/MyLogo.gif?
fragment=true">
<script language="javascript" src="http://yourserver:8080/peb/view/
MyWebPage/js/MyJS.js?fragment=true">
</script>
<LINK REL="StyleSheet" HREF="http://yourserver:8080/peb/view/
MyWebPage/CSS/MyStyles.css?fragment=true" TYPE="text/css"
MEDIA="screen"/>
```

Chapter
4

PASW Tag Library

A JavaServer Pages (JSP) tag library is provided with PASW Collaboration and Deployment Services for administrators and advanced users who want to create relationships between repository items and create custom Web pages (*.jsp* pages) containing items that can feed values to one another. The tag library provides the following basic functionality:

Authentication: You can set the user, password, and security provider and share across any items or prompts defined on the page. Authentication is required to access the items in the repository and for data source authentication.

Items: You can specify the definition of items, including the target "container" (`<div>` or `<iframe>` element). The items will run using a POST request for IFRAME targets and using AJAX (Asynchronous JavaScript and XML) for DIV targets.

Prompts: You can use prompts to dynamically adjust the parameters used to run items. The prompt location is only restricted to a location on the current page. Prompts can either be user defined or a selected parameter from an existing item definition.

Linking Relationships: You can define relationships between either:

- source report items and target report, job, scoring, or custom dialog items
- a list of prompts and a target item. Both the activation location (DIV or IFRAME) and the timing (ONDEMAND, ONLOAD, or NONE) are supported.

The tag library framework is made up of three main parts:

- Public JavaScript API.
- Custom tags and their interactions with each other.
- Tag library beans for data set retrieval. For more information, see the topic Tag Library Beans on p. 53.

This document describes each tag function available in the JSP tag library provided with PASW Collaboration and Deployment Services, and includes usage examples. After reading this document, we recommend reviewing the sample *.jsp* files shipped with the tag library before creating your own custom pages. For more information, see the topic JavaServer Pages Samples on p. 58.

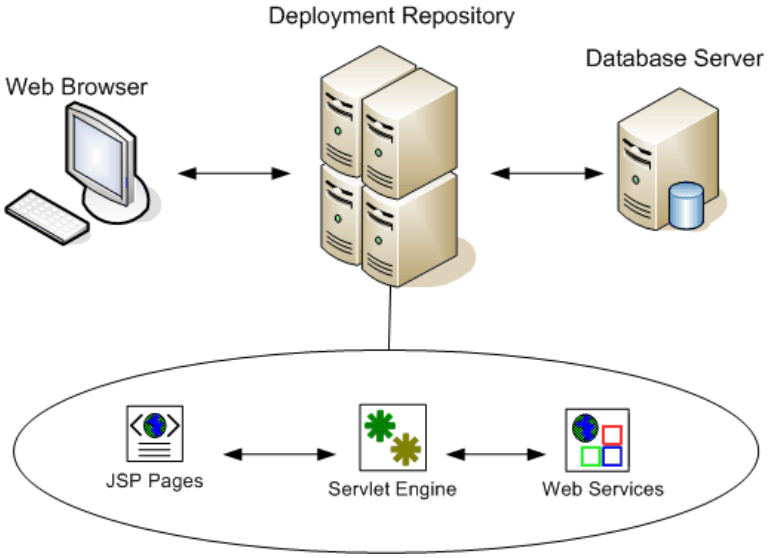
Upgrading to PASW Tag Library

Note that previous versions of PASW Collaboration and Deployment Services used a *.tld* file named *reporting-taglib.tld*. Any existing JSP pages using that name should be updated to reference *pasw-taglib.tld*.

JavaServer Pages Architecture

The "JSP Architecture" figure illustrates the architecture underlying the use the tag library. The application server hosting the repository includes a servlet engine that transforms the information contained in the library tags into input for web services included in PASW Collaboration and Deployment Services.

Figure 4-1
JSP Architecture



In general, the process of running items using the tag library is as follows:

- The JSP developer uses custom tags to define credentials, prompts, items, and item relationships in a *.jsp* file and stores the file on the application server hosting repository.
- When a client accesses the *.jsp* page, the server evaluates the tags and generates XML data islands or HTML elements as appropriate, which are used by the JavaScript components of the framework to identify and manage relationships between items and prompts.
- Validations are carried out in each tag handler and appropriate error messages are displayed on the page so the user is aware of any errors at each stage of page creation.
- A servlet provides support for running items and processing and returning the output.
- The web service associated with the item type is invoked to run the item and perform various validations.

Supported Items

A variety of repository items can be referenced in JSP pages using the PASW Tag Library. When processing the page, the MIME type of the item determines how the item gets processed. Valid items include:

- Reports
- Jobs
- Scoring models
- Custom interface definitions

Reports

For a report, the repository item must reference one of the following types of files:

- PASW BIRT Report Designer report design (**.rptdesign*)
- Showcase Suite report (**.dbq* or **.rpt*)
- Visualization definition (**.viz*)

The following properties should be considered when working with report items:

Output. A report item typically generates a single output. Visualization reports, however, generate an imagemap in addition to the visualization. The output for the item can be delivered in a variety of formats that depend on the report type. Available formats include:

- HyperText Markup Language (**.html*)
- Portable document format (**.pdf*)
- Report document (**.rptdocument*)
- HTML Complete (**.htmlc*)
- MIME HTML (**.mht*)
- Microsoft Word document (**.doc*)
- Microsoft PowerPoint (**.ppt*)
- Portable Network Graphic (**.png*)
- Enhanced Metafile (**.emf*)
- Joint Photographic Experts Group (**.jpeg*)

Prompts. When processed, the item will prompt for values for any variables defined in the report.

Location restrictions. Output of the **.rptdocument* type can only be displayed in an IFRAME.

Item linking. Report items can be used as sources for subsequent items or as targets of other items.

Supported tags. Report items do not support the `outputLocation` tag. All other tags in the tag library are supported.

The item may include additional information controlling the output display, such as the window title or the presence of a toolbar.

Jobs

For a job, the repository item must reference a PASW Collaboration and Deployment Services job, which has a MIME type of *application/x-vnd.spss-prms-job*. The following properties should be considered when working with job items:

Output. A job item can generate any number of outputs of varying types. The output produced depends on the steps contained within the job.

Prompts. When processed, the item will prompt for values for any job parameters defined for the job.

Location restrictions. Output from the individual steps within the job must be explicitly defined.

Item linking. Job items can be used as targets of other items but not as sources.

Supported tags. Job items do not support the `actionHandler` tag. All other tags in the tag library are supported.

Scoring Models

For a scoring model, the repository item must reference a file configured for scoring. Valid types of files include:

- scenario (**.scn*)
- PASW Modeler stream (**.str*)
- Predictive Model Markup Language (PMML)
- Real Time predictive application definition

The following properties should be considered when working with scoring items:

Output. A scoring item produces HTML output.

Prompts. When processed, the item can prompt for values for parameters, a data file, a data provider definition, and a model name.

Item linking. Scoring items can be used as targets of other items but not as sources.

Supported tags. Scoring items do not support the `outputLocation` and `actionHandler` tags. All other tags in the tag library are supported.

Custom Dialogs

Note: This functionality requires PASW Statistics adapters in the PASW Collaboration and Deployment Services environment. For more information, see the PASW Statistics installation documentation.

For a custom web interface, the repository item must reference a dialog definition (**.spd*). The following properties should be considered when working with custom dialog items:

Output. A custom dialog item generates:

- a single output file (**.spw*) that must be targeted to a frame or window
- HTML that can be targeted to a frame/window or a DIV

Prompts. When processed, the item will prompt for values for any prompts defined in the dialog definition. The item can also prompt for data sets. However, any help for prompts defined in the *.spd* file is not used. The application should include its own help references.

Location restrictions. The output can be viewed in a frame, DIV, or a new window.

Item linking. Dialog items can be used as targets of other items but not as sources.

Supported tags. Dialog items do not support the `actionHandler` tag. All other tags in the tag library are supported.

The web deployment properties described for use in a URL referencing a custom dialog item can be specified in the tag library either as properties nested in the `repositoryItem` tag or using the `sourceLinkPrompt` tag. For more information, see the topic Custom Dialog Parameters in Chapter 3 on p. 18.

The `dataset.uri` and `dataset.table` properties should always be defined, with the latter applying to data provider definitions only. In contrast, the `javascript.url`, `javascript.name`, `stylesheet.url`, and `stylesheet.name` properties are all ignored. Values for those properties should be defined within the JSP itself.

Building an Application

Each JSP page in a custom application must define some standard directives to allow the tag library to be used and referenced properly. The first, the `page` directive, sets properties for the entire page itself. These properties include:

- *language*, the scripting language used by the page
- *contentType*, the MIME type and character set used for responses to clients
- *session*, whether or not the tag library stores information on the session

The second directive, `taglib`, indicates which tags will be used by the JSP page. Properties defined for this directive include:

- *uri*, the proper path to *pasw-taglib.tld*
- *prefix*, a scope for the tags

Note that previous versions of PASW Collaboration and Deployment Services used a *.tld* file named *reporting-taglib.tld*. Any existing JSP pages using that name should be updated to reference *pasw-taglib.tld*.

The following sample uses the page directive to define the content type as text/html using the UTF-8 character set, the scripting language as Java, and use of the session object as true. The taglib directive identifies the location of the reporting *.tld* file and specifies a prefix of *r* for all tags defined within.

```
<%@ page contentType="text/html;charset=utf-8"
  language="java" session="true" %>

<%@ taglib uri="/WEB-INF/tlds/pasw-taglib.tld" prefix="r" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  http://www.w3.org/TR/html4/loose.dtd">

<html>
<!–Rest of HTML / JSP goes here –>
</html>
```

To put your application into production you should plan on creating a web application archive (*.war*) file containing the *.jsp* files and deploy it as a separate web application on your application server. This is the preferred method.

For example, the structure of expanded sample reporting tag library application archive (*paswTagLib.war*) included in the default installation of PASW Collaboration and Deployment Services is as follows:

```
paswTagLib
│   index.html
│   setup.html
├─js
│  <JavaScript files> │
├─jsp
│  <Java Server Page files>
├─META-INF
│    MANIFEST.MF
│├─WEB-INF
│  │  web.xml
│  │  weblogic.xml
│  ├─lib
│  │  <Java archive files>
│  └─tlds
│       pasw-taglib.tld
│       reporting-taglib.tld
│└─xsl
     <Extensible Stylesheet Language files>
```

Note that the TLD (Tag Library descriptions) file and libraries (*.jar* files) are included in the deployed *.war* file. The TLD file is also referenced in the application descriptor file (*web.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <filter>
    <filter-name>Common Authentication Filter</filter-name>
    <filter-class>com.spss.report.taglib.filter.AuthFilter</filter-class>
  <init-param>
    <param-name>PARAMETER_ENCODING</param-name>
    <param-value>UTF-8</param-value>
    <description>Parameter Encoding</description>
  </init-param>
   <init-param>
      <param-name>SSO_ADAPTER_CLASS</param-name>
      <param-value>com.spss.er.sso.authenticator.SessionAuthenticatorImpl</param-value>
      <description>SSO Authenticator Impl class</description>
   </init-param>
  </filter>
  <filter-mapping>
    <filter-name>Common Authentication Filter</filter-name>
    <url-pattern>/reportingTaglib/*</url-pattern>
  </filter-mapping>
  <filter-mapping>
    <filter-name>Common Authentication Filter</filter-name>
    <url-pattern>/tagLib/*</url-pattern>
  </filter-mapping>
  <servlet>
    <servlet-name>ReportingTaglibServlet</servlet-name>
    <display-name>
      Servlet responsible for fulfilling all requests from
      reporting taglibs
    </display-name>
    <servlet-class>
      com.spss.report.taglib.servlet.ReportingTaglibServlet
    </servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>ReportingTaglibServlet</servlet-name>
    <url-pattern>/reportingTaglib/*</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>ReportingTaglibServlet</servlet-name>
    <url-pattern>/tagLib/*</url-pattern>
  </servlet-mapping>

  <!–Start : Taglib Node –>

  <taglib>
```

```
    <taglib-uri>/reporting-taglib.tld</taglib-uri>
    <taglib-location>/WEB-INF/tlds/pasw-taglib.tld</taglib-location>
  </taglib>

  <!–End : Taglib Node –>

  <!–start : Security-Constraint Node –>
  <!–End : Security-Constraint Node –>
</web-app>
```

The application descriptor also specifies that *ReportingTaglibServlet* servlet is mapped to the */taglib* and */reportingTagLib* URL patterns, and either URL would call the servlet. The servlet Java class is *com.spss.report.taglib.servlet.ReportingTaglibServlet*. Optional single sign-on functionality is enabled by a servlet filter *Common Authentication Filter* which uses *com.spss.report.taglib.filter.AuthFilter* class and is mapped to the servlet by URL. The filter is initialized with encoding and SSO adaptor class parameters.

For more information about *.war* files, see online resources such as *http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/WebComponents3.html*. Also see your application server's documentation for additional information and requirements.

Important! Application *.war* files that are not deployed by PASW Collaboration and Deployment Services installation scripts or Package Manager, such as tag library or custom applications, may need to have class loader order modified. For example, for reporting and scoring tag library applications on WebSphere, class loader order must be set to *Classes loaded with application class loader first* and *.war* class loader policy to *Single class loader for application*.

Implementation Details

Users creating custom *.jsp* pages should be aware of the following information.

- Each time the server stops, any *.jsp* that are placed directly in the *tmp/deploy* directory files are lost. To preserve your *.jsp* files, save backup copies to a local drive and copy them back to the server after each restart. Production applications should by packaged in .war files. For more information, see the topic Building an Application on p. 32.
- Internet Explorer 6.0 and Mozilla Firefox™ 1.5 are supported. Firefox has the following restrictions:
  - For reports containing images and/or charts, the *.mht* output format combines all the images/charts and HTML in a single file compatible with Internet Explorer only.
  - ShowCase Report Writer reports (*.rpt*) use special technology for charts. This technology is only compatible with Internet Explorer.
- The server pre-validates all tags to the extent possible and writes error messages to the HTTP response as they are identified. This provides the JSP developer assistance in resolving problems as a page is being created. For example, the following items are validated: verify all required credentials are defined; verify credentials; verify report parameters exist; verify column names exists for a report object; verify the linkage between items is logically sound.
- The tags require a repository server.

- The tag library supports links between prompts and items, between row clicks and target items, between reports and drill-down reports in the same frame, and between prompts/table rows and target items opened in separate windows.
- All linkage behavior is hidden from the user and is defined using `sourceLinkRepositoryItem` and/or `sourceLinkPrompt` JSP tags. The user is not required to understand any technology beyond JSP tag usage.
- All target items must be predefined with parameters to receive the passed parameters.
- For custom dialogs, the standard CSS defines default styles that are included automatically at the point the `repositoryItem` tag is used. To override those styles, include a custom style sheet after the `repositoryItem` tag. For example:

```
<r:repositoryItem name="sample" inputURI="spsscr:///myDialog.spd"
  ...more here...
</r:repositoryItem>

<link rel="stylesheet" type="text/css" href="MyStyles.css">
```

Public JavaScript API

The framework provides JavaScript functions for processing repository items, retrieving bookmarked report values, and retrieving cascading prompt values.

runRepositoryItem Function

The public JavaScript function provided by the framework for running an item is `runRepositoryItem`. It allows the developer to run an item by connecting this JavaScript to an event handler, and activate an item when using prompts. However, when running an item directly using this function, the normal prompt validation is bypassed. It is the application's responsibility to validate any parameters before invoking function.

The function accepts three arguments.

- a string corresponding to the name of the item to execute. The name must have been defined using the `name` attribute of the `repositoryItem` tag.
- an Array of data values to use as parameter values when running the item. The array has the following structure:

```
var thisVar = new Object();
thisVar.value = "param_value";
thisVar.columnName = "param_name";
var linkedData = new Array(thisVar);
```

- an optional parameter specifying a target location for the item output. This follows the same rules as the `location` attribute of the `repositoryItem` tag. It may be the ID of a DIV, the name of an IFRAME or FRAME, **NEW* or **windowName*. If omitted, the default location from the `repositoryItem` is used.

The data value array many be specified in one of the following ways:

- **null (or omitted)**. In this case, any necessary values are retrieved using any `sourceLinkPrompt` tags defined for the item.
- **the link data from actionHandler**. `actionHandler` tags define functions to call and the parameter values which are captured as part of the event. Those parameters can be passed directly to the `runRepositoryItem` API.
- **user defined**. The JavaScript calling the `runRepositoryItem` API can define the values of the array above as necessary. The *columnName* is the name of the column defined in the `sourceLinkVariable` tag. The value is the value to pass as the parameter.

getBookmarkedValues Function

The `getBookmarkedValues` function retrieves the values of cells that have been bookmarked in a PASW BIRT Report Designer report. The id attribute of the cell should be set to the bookmark value. This function can be used for linking complex PASW BIRT Report Designer reports involving crosstabs and nested tables.

The `getBookmarkedValues` function accepts two arguments:

- a parent node in the DOM of the PASW BIRT Report Designer report that the function needs to traverse to get the cell values matching the items specified in the bookmarks array
- an array of bookmarks defined in the PASW BIRT Report Designer report whose values are needed. For example, [“bookid1”,”bookid2”].

The function returns an array of data values to use as parameter values when running the report. The array has the following structure:

```
var thisVar = new Object();
  thisVar.value = "cell_value";
  thisVar.columnName = "bookmark";
```

The *columnName* is the name of the bookmark. The *value* is the value of the specified cell that is bookmarked.

retrievePromptValues

The `retrievePromptValues` function should be called when using parameters with custom controls, and supports both cascading and non-cascading prompts. Call this function in the body `onLoad` handler to load the initial values of the prompt (or the parent prompt in the case of a cascading prompt). Call this function in the `onChange` handler of the control used to define the cascading parameter. The function will make calls to the server and get the prompt values to fill the parameter controls with updated values depending on the parent parameter value selected.

The `retrievePromptValues` function accepts four arguments:

- a string denoting the name of the report containing the definitions of the cascading parameters. The name must have been defined as the `name` attribute of a `repositoryItem` tag.

- a string corresponding to the name of the parameter in the report. For cascading parameters, this string is the name of the cascading parameter group. The cascading group must be present in the report.
- a user-defined function that accepts an array of value and display text for the new options. The array can be null, in which case the function should clear the control. This function will be called by `retrieveCascadingPromptValues` to populate the parameter controls with new values.

```
function callback(options) {
  // logic to clear the control
  // logic to add value and display text to control
  for(var i = 0; i <options.length; i++) {
    control.value = options[i].value;
    Display Text for control = options[i].displayText;
  }
}
```

- an array of the selected preceding values present in the cascading group. This array is only needed for cascading parameters and should be omitted for a non-cascading parameter. The parameters must be in sequential order. To get the list of the parent cascading parameter, specify preceding values:

```
var precedingvals= new Array();
```

The preceding values array has the following structure. For example, to get the list of cities in MN:

```
precedingvals= new Array();
precedingvals[0]= "USA";
precedingVals[1]="MN";
```

PASW Tag Library Tag Reference

The various tags included in the PASW Tag Library are dependent on each other and, for validation purposes, need to know that the references are correctly met. The tags must also be defined in the correct sequence. The following sections describe each available tag in detail.

This tag library depends upon JSP 1.1.

credential Tag

The `credential` tag defines both a data source login credential and a repository login credential. The credential is referenced by name for all items and/or prompts defined on the page. It should be defined prior to any tags that may reference the credential. In normal use, it would be the first tag referenced in the JSP.

The `credential` tag can contain `properties` elements. For, example, in the case of J. D. Edwards (JDE) enabled data sources, the `credential` looks like the following:

```
<credential>
   <properties>
      <property name="JDE_LIBRARY_LIST_SELECTED" value="liblist_name"/>
   </properties>
```

```
</credential>
```

Tag Nesting:

- None

Expected Output:

- None. This tag provides authentication information. The tag does not produce output, but caches the credentials using the `name` attribute as a key for later use with a report or prompt tag.

Table 4-1
Attributes for the credential tag

Name	Required	Description
name	true	Either an internal name for the repository credential or the name of a data source used in a repository item object. This is used to link items and prompts to this credential and to satisfy any data source logins required by referenced items. ■ For repository credentials, this must match the name provided on the `repositoryCredentialName` attribute of the `repositoryItem` tag. ■ For database credentials, the name must match the data source name as it is referenced by the item using that data source. This name is used to store the credential values in a session variable. All credentials must have a unique name.
useSSO	false	Indicates if Single Sign On credential for Kerberos should be used. If this attribute is set to true, then the username, password, and provider attributes must not be specified. When using SSO, the Authentication Filter must be configured in the web.xml file. For more information, see the topic Building an Application on p. 32.

Name	Required	Description
credentialDefinitionName	false	The name of a credential defined as a resource in the repository. If this value is specified, the `username`, `password`, and `provider` attributes do not need to be defined as the credential resource includes this information.
provider	false	For repository credentials, this is the optional security provider name. Valid values include: ■ *Native* for the built-in provider ■ *AD/<domain>* for Active Directory, where *<domain>* corresponds to the DNS namespace ■ *ADL/<domain>* for Active Directory with local override, where *<domain>* corresponds to the DNS namespace ■ *iSeries* for iSeries ■ *devldapOpenLDAP* for OpenLDAP If not specified, the built-in repository security provider is used. This attribute is ignored for database credentials.
username	false	The user name to use for authentication.
password	false	The password for the specified username. The password is used internally by the tag library. It is NOT written to the JSP result.

Sample Usage:

The following sample specifies three credentials. This first is for accessing the repository with a specified username and password. The value of *Native* for *provider* indicates that the username/password pair for validation is defined in the native local security provider. The second credential employs single sign-on for the repository using the user's previously authenticated credentials. The third credential is for a data source named *Northwind*.

```
<r:credential name="repositoryCredential" provider="Native"
   username='admin' password='password'/>

<r:credential name="repositorySSO" useSSO="true" />

<r:credential name="Northwind" username='sa' password='sa'/>
```

repositoryItem Tag

The `repositoryItem` tag is the main tag for for defining repository item definitions that will be used by the application. The `repositoryItem` tag may reference reports (ShowCase reports, PASW BIRT Report Designer reports or visualization reports), jobs, scoring items, or SPD files. The repository items may be run directly, used to provide prompts, or ran programatically.

Any `sourceLinkPrompt` and `sourceLinkRepositoryItem` tags should be nested within the `repositoryItem` tag.

- Use a nested `sourceLinkRepositoryItem` tag if this item will be run when the user clicks on a different item.
- Use `sourceLinkPrompt` when the parameter values will come from prompts defined on the page or defined directly in the item.

You may optionally specify aditional properties that are specific to a type of repository item. The property names must be in lower case for them to work in the Firefox browser. These property values will be passed to the URL to run the repository item. The properties are specified as a nested XML block.

Tag Nesting:

- This tag may include one `sourceLinkRepositoryItem` and/or multiple `sourceLinkPrompt` and `outputLocation` tags.

Table 4-2
Attributes for the repositoryItem tag

Name	Required	Description
name	true	Defines a unique name for the item. The name can then be referenced by other tags or via the `runRepositoryItem()` JavaScript API.
inputURI	true	The item definition to be used to render the report output. This value must specify a URI that may be used to locate the item definition. The following URI schemes are supported: ■ **file**: References a specific file on the application server and/or a network file location ■ **spsscr**: References a file in the repository. This scheme allows files to be referenced by identifier or hierarchical path within the repository. Specific version markers can be specified. If no version or label is specified, the latest version is used. ■ **scoring**: References a model configuration from the repository. Scoring configurations are referenced

Name	Required	Description
		by name from the tag libraries. If a scoring configuration is renamed, the tag library reference must also be modified.
activate	true	Specifies when the item will be activated. Options include: ■ **ONDEMAND**: Runs the item when activated by a row click on a source report. ■ **ONLOAD**: Runs the item when the page initially loads. ■ **NONE**: Item does not run automatically. In this case, the item is used to provide prompts or prompt values. Regardless of the activate setting, any report may be run programatically by using the public JavaScript `runRepositoryItem()` API.
location	false	The destination for the output resulting from running the item. The usage varies slightly depending on what the target type is. ■ For DIV targets, the location should specify the ID of the DIV tag where the output is to be placed. ■ For IFRAME targets, the location must specify the name of the frame. ■ To open the output in a new window, specify a location of `*NEW`. ■ To direct the output to a named window, use an asterisk (`*`) followed by the window name. For example, `*MYWINDOW` will open a new window called *MYWINDOW* and will reuse that window on each activation of the link.
repositoryCredentialName	true	The name of the credential that should be used when accessing the item from the repository and running the item via the reporting service. The credential should have been previously defined using the `credential` tag.

Name	Required	Description
outputType	false	The type of output to generate. The supported output types vary by item type. Normally this will be either HTML or PNG but other options include: **PASW BIRT Report Designer Reports:** HTML, PDF, RPTDocument, PowerPoint, Word Document, HTMLC **Visualization Reports:** PNG, EMF, JPEG, HTML **ShowCase Query:** HTML, PDF, CSV, WK4, Excel 97/2003, xlsx **ShowCase Report Writer:** HTML, PDF, MHT If not specified, the output will default to HTML (or PNG for visualization reports). When a PASW BIRT Report Designer or ShowCase report is used as a link source, the `outputType` will be ignored and HTML generated since the other output types don't support linking. To display reports using the viewer, specify a type of *RPTDocument*. For this type, the target must be an iframe.
showTitle	false	Specifies whether the title bar of the BIRT Viewer is to be shown. Specify the value as either true or false. This setting only applies when the outputType is RPTDocument which displays in the Report Viewer. Default is true.
title	false	Specifies the title for BIRT Viewer. This setting only applies when the outputType is RPTDocument which displays in the Report Viewer. If not specified, the default title is displayed.
showToolBar	false	Specifies whether the ToolBar of the BIRT Viewer is to be shown. Specify the value as either true or false. This setting only applies when the outputType is RPTDocument which displays in the Report Viewer. Default is true.

Name	Required	Description
showNavigationBar	false	Specifies whether the navigation bar of the BIRT Viewer is to be shown. Specify the value as either true or false. This setting only applies when the outputType is RPTDocument which displays in the Report Viewer. Default is true.
width	false	This is the width of the output image. The width must be greater than 0 and specified in conjunction with the height. If not specified, the default width and height are used.
height	false	This is the height to use when output is an image. The width must also be specified or the setting will have no effect. The value must be greater than 0.

Sample Usage:

The following sample defines an item named *AllCountries* for a PASW BIRT Report Designer report stored in the repository.

```
<r:repositoryItem name="AllCountries"
  inputURI="spsscr:///SampleReports/BIRT/CountrySales.rptdesign"
  repositoryCredentialName="repositoryCredential"
  outputType="HTML" width="400" height="300"
  activate="ONLOAD" location="ReportDIV">
</r:repositoryItem>
```

To prompt for parameter values for an item, include a *sourceLinkPrompt* tag. The following sample retrieves a value for the parameter *ShipCountry* using the JavaScript function *getValue*.

```
<r:repositoryItem name="CountrySales"
  inputURI="spsscr:///SampleReports/BIRT/CountrySalesByCity.rptdesign"
  repositoryCredentialName="repositoryCredential"
  outputType="HTML" activate="ONDEMAND" location="ReportDIV">
  <r:sourceLinkPrompt targetNameParameter="ShipCountry"
    getValueJSFunction="getValue('IDFilter')"/>
</r:repositoryItem>
```

To run a second item in response to a user action, include a *sourceLinkRepositoryItem* tag. The following sample runs the visualization report *CityDetails* in response to an action in the source report *AllCountries*.

```
<r:repositoryItem name="CityDetails"
  inputURI="spsscr:///SampleReports/Vis/CitiesBarChart.viz"
  repositoryCredentialName="repositoryCredential"
  outputType="png" width="400" height="300"
  activate="ONDEMAND" location="SecondReportDIV">
  <r:sourceLinkRepositoryItem sourceReportName="AllCountries">
    <r:sourceLinkVariable columnName="ShipCountry" targetNameParameter="ShipCountry"/>
```

```
  </r:sourceLinkRepositoryItem>
</r:repositoryItem>
```

repositoryItemPrompt Tag

The `repositoryItemPrompt` tag generates the HTML for a prompt variable that is defined in the referenced item. The item that the prompt is referencing must be defined using the `repositoryItem` tag before this tag can be used. Use this when you want prompt controls such as those used in Deployment Portal to be used in your application.

This tag generates the HTML prompt controls in the location corresponding to where the tag is used. The tag must be associated with a particular parameter of an item to be useful. The association with parameters is done using the `sourceLinkPrompt` tag, where the `promptID` of the `sourceLinkPrompt` must match the `promptID` of this tag.

Tag Nesting:

- None

Expected Output:

- An HTML element that allows the user to select and/or type personal values depending on the `promptType`, which is selected as `parameterName`. The `repositoryItemPrompt` tag supports all parameters Deployment Portal supports. As a result, all types of prompts are supported and the appropriate HTML element is generated.

Table 4-3
Attributes for the repositoryItemPrompt tag

Name	Required	Description
promptId	false	A unique identifier that can be referenced from the `promptId` attribute of the `sourceLinkPrompt` tag.
repositoryItemName	true	A reference to the name of the item as defined in the `name` attribute of the `repositoryItem` tag.
parameterName	false	Name of the prompt variable as defined in the item.

Sample Usage:

The following sample prompts for a value for the *EmployeeID* parameter in the *Employees* report.

```
<repositoryItem name="Employees"
   inputURI="file:///d:/yourDS/ReportTaglib/Employees.dbq"
   repositoryCredentialName="localhost" activate="NONE" />

<repositoryItemPrompt promptId="EmployeeIdPrompt"
   repositoryItemName="Employees" parameterName="EmployeeID" />
```

report Tag

This tag is deprecated. Use the `repositoryItem` tag instead

reportPrompt Tag

This tag is deprecated. Use the `repositoryItemPrompt` tag instead

outputLocation Tag

This tag associates the generated output that exists in the repository with the location on the page where the output is displayed. When the item runs, the output is retrieved from the repository and displayed at the specified target location on the page.

This tag must always be nested within a `repositoryItem` tag.

Tag Nesting:

- None

Table 4-4
Attributes for the outputLocation tag

Name	Required	Description
outputId	false	This is the path to the output that exists in the repository. For custom dialogs, this attribute should be omitted. The output from running the syntax is automatically detected.
location	true	This attribute specifies where the output should be placed on the page. ■ For DIV targets, the location should specify the ID of the DIV tag where the output is to be placed. ■ For IFRAME targets, the location must specify the name of the frame. ■ To open the report output in a new window, specify a location of `*NEW`. ■ To direct the output to a named window, use an asterisk (`*`) followed by the window name. For example, `*MYREPORTS` will open a new window called *MYREPORTS* and will reuse that window on each activation of the link.

Name	Required	Description
		HTML outputs may target a DIV. All other outputs should target an IFRAME or a window.
partId	false	This is used to identify the specific part or item of the SPW archive output. **Note:** This functionality requires PASW Statistics adapters in the PASW Collaboration and Deployment Services environment. For more information, see the PASW Statistics installation documentation.

Sample Usage:

The following sample specifies an output location for a chart stored in the repository using the *ChartFRAME* IFRAME tag.

```
<outputLocation outputId="spsscr:///output/output_chart.png"
   location="ChartFRAME"/>
```

If the attribute values depend on parameter values, use the *sourceLinkPrompt* tag to define matches for the parameters. If a match is found, it is substituted for the parameter. For example, the following sample defines two *outputLocation* tags with filenames that depend on parameters.

```
<repositoryItem name= "Call_Center_Score"
   inputURI= "spsscr:///job/Call Center"
   repositoryCredentialName="localhost"
   activate="ONDEMAND"/>
   <outputLocation outputId ="spsscr:///output/output_tab_${JobParam1}.png"
      location="ChartFRAME"/>
   <outputLocation outputId="/output/output_chart_${JobParam2}.html"
      location="ReportDIV"/>
   <sourceLinkPrompt promptId="JobParam1" parameterValue="Jan" />
   <sourceLinkPrompt promptId="JobParam2"
      targetNameParameter="html_id_for_the_value" />
</repositoryItem>
```

For *JobParam1*, a value of *Jan* is substituted in the name, resulting in *output_chart_Jan.png* appearing at *ChartFRAME*.

For *JobParam2*, the value associated with the html control for the parameter is substituted in the name. If that value is *Illinois*, the file *output_tab_Illinois.html* appears at *ReportDIV*.

sourceLinkPrompt Tag

The `sourceLinkPrompt` tag associates the item parameters with the prompts providing their values. These could be user defined HTML elements, javaScript functions, prompts created using the `repositoryItemPrompt` tag, or directly specified values.

The `sourceLinkPrompt` tag must always be nested within a `repositoryItem` tag. When the item runs, the parameter values are retrieved using the `sourceLinkPrompt`s.

Tag Nesting:

- None

Validations Performed:

- None

Expected Output:

- None

Table 4-5
Attributes for the sourceLinkPrompt tag

Name	Required	Description
targetNameParameter	true	Name of the report parameter as it is defined in the report definition.
promptId	false	The `promptId` could be the ID of a `reportPrompt` tag or the name of an HTML control. When a prompt value is needed, the `reportPrompt` or the HTML control will be used to determine the prompt value. Either `promptId`, `parameterValue` or `getValueJSFunction` should be specified.
parameterValue	false	Specifies a value for the parameter instead of prompting for one. This should be specified when the application knows the parameter value when the JSP is being processed. In that case, the value can be specified directly using this attribute. If `parameterValue` is specified, then `promptId` and `getValueJSFunction` should not be used.

Name	Required	Description
getValueJSFunction	false	Identifies a function to call to retrieve the prompt value(s). The function should return either a single value or return an array of values. This attribute should include the function name, parentheses and any parameters as necessary. For example, for a function called *MyGetValues* that takes one parameter, set the attribute to `MyGetValues('myPrompt-tID')`.
validateJSFunction	false	Identifies a function to call to to provide validation of the prompt. The function should return `true` if the prompts are valid. This attribute should include the function name, parentheses and any parameters as necessary. For example, for a function called *MyValidate* that takes one parameter, set the attribute to `MyValidate('myPromptID')`.

Sample Usage:

The following sample prompts for two parameter values using `reportPrompt` tags. The `sourceLinkPrompt` tags for the *CountrySales* report use the identifiers for those prompts to supply their values to the report.

```
<r:repositoryItem name="CountrySales"
  reportDefinitionURI="spsscr:///SampleReports/BIRT/CountryCity_cascadingParameter.rptdesign"
  repositoryCredentialName="repositoryCredential"
  outputType="HTML" activate="ONDEMAND" location="ReportDIV">
  <r:sourceLinkPrompt targetNameParameter="ShipCountry" promptId="IDFilter"/>
  <r:sourceLinkPrompt targetNameParameter="ShipCity" promptId="IDFilter1"/>
</r:repositoryItem>

<table width="95%" cellspacing="1" bgcolor="black">
  <tr bgcolor="white">
    <r:repositoryItemPrompt promptId="IDFilter" repositoryItemName="CountrySales1"
      parameterName="ShipCountry"/>
  </tr>
  <tr bgcolor="white">
    <r:repositoryItemPrompt promptId="IDFilter1" repositoryItemName="CountrySales1"
      parameterName="ShipCity"/>
  </tr>
</table>
```

sourceLinkRepositoryItem Tag

The `sourceLinkRepositoryItem` tag identifies the source item and variables used to satisfy the item's defined parameters. Using this mechanism, when the source item is clicked, the parent item runs using the parameters defined within the nested `sourceLinkVariable` tags.

This tag must always be nested within a `repositoryItem` tag. It should contain one or more nested `sourceLinkVariable` tags.

Tag Nesting:

- The `sourceLinkRepositoryItem` tag contains one or more `sourceLinkVariable` tags that identify the source column and the target parameter names.

Expected Output:

- None

Table 4-6
Attributes for the sourceLinkRepositoryItem tag

Name	Required	Description
sourceName	true	Name of the `repositoryItem` that will serve as the source of the relationship
linkType	false	Determines what action on the source report will trigger the running of the current report. Currently there is only one supported `linkType`, *row*. For this type, when a row in the source report is clicked, the target report runs. In future releases, additional linkTypes may be added.

Sample Usage:

The following sample identifies *CityDetails* as the report to run in response to a user action in the *AllCountries* report.

```
<r:repositoryItem name="CityDetails"
  inputURI="spsscr:///SampleReports/BIRT/CountrySalesByCity.rptdesign"
  repositoryCredentialName="repositoryCredential"
  outputType="HTML" width="400" height="300"
  activate="ONDEMAND" location="SecondReportDIV">
  <r:sourceLinkRepositoryItem sourceReportName="AllCountries">
    <r:sourceLinkVariable columnName="ShipCountry"
      targetNameParameter="ShipCountry" />
  </r:sourceLinkRepositoryItem>
</r:repositoryItem>
```

sourceLinkReport Tag

This tag is deprecated. Use the `sourceLinkRepositoryItem` tag instead

sourceLinkVariable Tag

The `sourceLinkVariable` tag defines the mapping between the variable or column to use in the source item and the parameter as defined in the target item. This tag must always be nested under a `sourceLinkRepositoryItem` tag.

Tag Nesting:

- None

Validations Performed:

- None

Expected Output:

- None

Table 4-7
Attributes for the sourceLinkVariable tag

Name	Required	Description
columnName	true	For ShowCase reports, this attribute specifies the name of the column in the source report. For Visualization reports, this attribute contains the id of the `sourceVariable` or `derivedVariable` element of the Visualization specification. Currently only categorical variables are supported.
targetNameParameter	true	Name of the parameter in the target query

Sample Usage:

The following sample maps the *ShipCountry* variable in the *AllCountries* report to the *ShipCountry* parameter in the *CityDetails* report.

```
<r:repositoryItem name="CityDetails"
  inputURI="spsscr:///SampleReports/Vis/CitiesBarChart.viz"
  repositoryCredentialName="repositoryCredential"
  outputType="png" width="400" height="300"
  activate="ONDEMAND" location="SecondReportDIV">
  <r:sourceLinkRepositoryItem sourceName="AllCountries">
    <r:sourceLinkVariable columnName="ShipCountry" targetNameParameter="ShipCountry"/>
  </r:sourceLinkRepositoryItem>
</r:repositoryItem>
```

actionHandler Tag

Defines the action handlers that should be applied to the item. When action handlers are defined, the automatic linking setup using `sourceLinkRepositoryItem` no longer applies. The application builder is responsible for running any target items using the `runRepositoryItem` public Java script API.

Tag Nesting:

- Any data values that need to be passed as parameters to the JavaScript function should be defined using nested `actionParameter` tags.

Table 4-8
Attributes for the actionHandler tag

Name	Required	Description
event	true	The event name. Valid events include: ■ *onclick* ■ *onmouseover* ■ *onmouseout*
function	true	The name of the Java Script function to call when the event occurs. This should be the function name only, without () or any parameters.
partId	false	This is used to identify the specific part of the report that the actions should apply to.

Sample Usage:

The following `repositoryItem` tag defines three action handlers, one for each type of event that could occur. Each handler calls a unique JavaScript function that defines the subsequent processing.

```
<r:repositoryItem name="AllCountries"
   inputURI="spsscr:///SampleReports/BIRT/CountrySales.rptdesign"
   repositoryCredentialName="repositoryCredential"
   outputType="HTML"
   width="400" height="300"
   activate="ONLOAD" location="ReportDIV">
   <r:actionHandler event="onclick" function="myOnClick">
      <r:actionParameter name="ShipCountry"/>
   </r:actionHandler>
   <r:actionHandler event="onmouseover" function="myOnOver">
      <r:actionParameter name="ShipCountry"/>
   </r:actionHandler>
   <r:actionHandler event="onmouseout" function="myOnOut" />
</r:repositoryItem>
```

actionParameter Tag

There should be an `actionParameter` for each data value from the item that needs to be passed to the `actionHandler` JavaScript function. This tag must be nested within the `actionHandler` tag.

Tag Nesting:

- None

Table 4-9
Attributes for the actionParameter tag

Name	Required	Description
name	true	Name of the column or variable that defines which value from the report results should be passed to the function. ■ For visualization reports, the name is the `id` attribute of the `sourceVariable` or `derivedVariable` element. Currently only categorical variables are supported. ■ For ShowCase reports, this would be the column name.

Sample Usage:

The following sample defines an `actionParameter` named *ShipCountry* that gets passed to the JavaScript function *myOnClick* when the user clicks the report.

```
<r:actionHandler event="onclick" function="myOnClick">
   <r:actionParameter name="ShipCountry"/>
</r:actionHandler>
```

Tag Library Beans

The framework includes tag library beans that can be used together for a variety of purposes. For example, the beans can be used to retrieve a data set that can then be used to build custom HTML controls.

In order to use the beans, you must first declare references to them in the JSP. This is done through the import attribute of the page directive.

```
<%@ page contentType="text/html;charset=utf-8"
   language="java"
   session="true"
   import="java.util.Map"
   import="java.util.HashMap"
   import="com.spss.report.taglib.bean.ReportBean"
   import="com.spss.report.taglib.bean.Credential"
%>
```

The code samples for beans use the JavaServer Pages Standard Tag Library (JSTL) which should be included using the taglib directive.

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
```

For more information on JSTL, refer to the Sun documentation (*http://java.sun.com/products/jsp/jstl/*).

Credential Bean

The Credential bean defines the credentials that will be used by other beans. The code sample below creates two credentials and stores them in a HashMap. In the sample below, the *localhost* credential provides the logon information for the repository. The *ps4008* credential is for a SQL Server data source called ps4008 that is referenced by the report definition.

```
<%
Map credentialMap = new HashMap();
Credential repositoryCredential = new Credential("localhost","Native","admin","spss",null);
Credential datasourceCredential = new Credential("ps4008",null,"sa","sa",null);
credentialMap.put("localhost",repositoryCredential);
credentialMap.put("ps4008",datasourceCredential);
%>
```

ReportBean Bean

The ReportBean is used to retrieve the data for a data set that is defined in a report definition. The code below uses the previously created credentialMap to retrieve a data set. Visualization reports do not support this function.

```
<%–Creating JavaBeans –%>
<jsp:useBean id="report" class="com.spss.report.taglib.bean.ReportBean">
  <jsp:setProperty name="report" property="reportDefinitionURI"
    value="file:///d:/SPSS/ps4008/Test.dbq" />
  <jsp:setProperty name="report" property="repositoryCredentialName"
    value="localhost" />
  <jsp:setProperty name="report" property="host" value="localhost" />
  <jsp:setProperty name="report" property="port" value="8080" />
  <jsp:setProperty name="report" property="dataSetName"
    value="DataSet1" />
  <jsp:setProperty name="report" property="credentialMap"
    value="<%=credentialMap%>" />
</jsp:useBean>
```

The properties used in this code are:

- *reportDefinitionURI*. The location of the report
- *repositoryCredentialName*. The host name
- *port*.The port name

- *dataSetName*. For PASW BIRT Report Designer reports this is the name of the data set as defined in the report definition. This does not apply to ShowCase reports and should be omitted.
- *credentialMap*. A reference to a HashMap containing the credentials to use

The ReportBean can then be run to return the data set. The data can be used to generate a list control as shown in the code below.

```
<SELECT style="WIDTH :250 px" ID="EmployeeID_Prompt" NAME="EmployeeID_Prompt"
  TABINDEX="2">
<c:forEach var="row" items="${report.rows}">
  <c:forEach var="column" items="${row.columns}">
    <c:if test='${column.name == "EmployeeID"}'>

      <OPTION VALUE='<c:out value="${column.value}" />'>
        <c:out value="${column.value}" />
      </OPTION>
    </c:if>
  </c:forEach>
</c:forEach>
</SELECT>
```

SearchBean Bean

The SearchBean bean provides a query mechanism for locating content in the repository that meet specified criteria. For example, the bean can retrieve a list of data provider definition and PASW Statistics data file (*.sav*) sources in the repository that match a specified search criterion. The code below defines bean properties to query for all data provider definition and PASW Statistics data sources using the MIME types associated with those sources.

```
<jsp:useBean id="data_sources"
  class="com.spss.report.taglib.bean.SearchBean" scope="page">
  <jsp:setProperty name="data_sources" property="request" value="<%= request %>" />
  <jsp:setProperty name="data_sources" property="credentialName"
    value="AuthenticationCredential" />
  <jsp:setProperty name="data_sources" property="searchQuery"
    value="<%=  "('$$search/mimetype'='application/x-vnd.spss-spss-data' or "
      + "'$$search/mimetype'='application/x-vnd.spss-statistics-data' or "
      + "'$$search/mimetype'='application/x-vnd.spss-data-provider')" %>" />
</jsp:useBean>
```

The properties used in this code are:

- *request*. An `HttpServletRequest` object.
- *credentialName*. Credential needed to connect to repository. In this case, the value corresponds to the credential *AuthenticationCredential* defined using the `credential` tag.
- *searchQuery*. String denoting the search criterion. The structure of this string matches the syntax used for the dataset.search.criteria parameter for URL strings.

The SearchBean can then be run to return the matching data sources. The code below presents the name, modification date, version label, and author metadata for the data sources in a table.

```
<Table  border="0" height="100%" width="100%" cellpadding="0" cellspacing="0">
  <tr>
    <td align="center" bgcolor="#EEEEEE"  style="padding-top:5px; padding-bottom:5px;">
      Data Source
    </td>
    <td align="center" bgcolor="#EEEEEE"  style="padding-top:5px; padding-bottom:5px;">
      Modified Date
    </td>
    <td align="center" bgcolor="#EEEEEE"  style="padding-top:5px; padding-bottom:5px;">
      Version Label
    </td>
    <td align="center" bgcolor="#EEEEEE"  style="padding-top:5px; padding-bottom:5px;">
      Author
    </td>
  </tr>
  <c:forEach var="data_source" items="${data_sources.records}" varStatus="status"
    begin="0" end="3" step="1">
    <tr>
      <td align="center" bgcolor="#EEEEEE"  style="padding-top:5px; padding-bottom:5px;">
        <c:out value="${data_source.title}" />
      </td>
      <td align="center" bgcolor="#EEEEEE"  style="padding-top:5px; padding-bottom:5px;">
        <c:out value="${data_source.modifiedDate}" />
      </td>
      <td align="center" bgcolor="#EEEEEE"  style="padding-top:5px; padding-bottom:5px;">
        <c:out value="${data_source.versionLabel}" />
      </td>
      <td align="center" bgcolor="#EEEEEE"  style="padding-top:5px; padding-bottom:5px;">
        <c:out value="${data_source.author}" />
      </td>
    </tr>
  </c:forEach>
</Table>
```

PevMetaDataBean Bean

The PevMetaDataBean bean retrieves variable metadata from data provider definition and PASW Statistics data file (*.sav*) sources. The code below defines properties for the bean to query a *.sav* file.

```
<jsp:useBean id="variables"
  class="com.spss.report.taglib.bean.PevMetaDataBean" scope="page">
  <jsp:setProperty name="variables" property="request" value="<%= request %>" />
  <jsp:setProperty name="variables" property="datasetуri"
    value="spsscr:///sav_files/demo.sav" />
  <jsp:setProperty name="variables" property="credentialName"
    value="AuthenticationCredential" />
</jsp:useBean>
```

The properties used in this code are:

- *request*. An `HttpServletRequest` object.
- *datasetуri*. The URI for the data file or data provider definition containing the variables.
- *credentialName*. Credential needed to connect to repository. In this case, the value corresponds to the credential *AuthenticationCredential* defined using the `credential` tag.

The PevMetaDataBean can then be run to return the metadata for the variables in the dataset. The code below presents the metadata in a table.

```
<Table  border="0" height="100%" width="100%" cellpadding="0" cellspacing="0">
  <tr>
    <td align="center" bgcolor="#EEEEEE"  style="padding-top:5px;
      padding-bottom:5px;">
      Variable Name
    </td>
  </tr>
  <c:forEach var="group" items="${variables.variablesMetaData}" >
    <c:forEach var="v" items="${group.variableMetaData}" varStatus="status"
      begin="0" end="3" step="1">
      <tr>
        <td align="center" bgcolor="#EEEEEE"  style="padding-top:5px;
          padding-bottom:5px;">
          <c:out value="${status.count}" /> <c:out value="${v.name}" />
        </td>
      </tr>
    </c:forEach>
  </c:forEach>
</Table>
```

ScoringBean Bean

The ScoringBean bean retrieves a list of scoring configurations for a specified model that are able to respond to a scoring request. The `getScoringConfigurations` method of the bean accepts the following parameters:

- *credential*. Credentials for accessing the repository defined using the `Credential` bean.
- *modelLocationUri*. The URI for a model in the repository.

Alternatively, instead of supplying a `Credential` bean item, the following two parameters can be used for specifying credentials:

- *request*. An `HttpServletRequest` object.
- *credentialName*. Credential needed to connect to the repository defined using the `credential` tag.

The following code retrieves the scoring configurations for the model *KMeans.xml* that can respond to a scoring request using a credential defined using the `credential` tag:

```
<r:credential name="repositoryCredential" provider="Native"
  username='<%= request.getParameter("userid")%>'
```

```
  password='<%= request.getParameter("password")%>' />

<%
  String[] configurations = ScoringBean.getScoringConfigurations(request,
    "repositoryCredential", "spsscr:///Sample/KMeans.xml");
%>
```

The array returned by the bean can be used to populate a form from which a user can select a scoring configuration to use for subsequent scoring.

```
<form id="selectConfigurationForm" target="ScoringIframe" method="POST">
  <div style="display:none">
    <input name="userid" type="text" value="<%= request.getParameter("userid")%>"/>
    <input name="password" type="text" value="<%= request.getParameter("password")%>"/>
  </div>
  Select Scoring Configuration:
  <select name="selectedConfiguration" onchange="onSelectConfiguration(this)">
    <option></option>
    <%
      for (int i=0; i < configurations.length; i++)
      {
    %>
        <option value="<%= configurations[i].replaceAll("[ ]", "%20")%>">
          <%= configurations[i] %></option>
    <%
      }
    %>
  </select>
</form>
```

JavaServer Pages Samples

PASW Collaboration and Deployment Services includes a variety of JSP samples illustrating the use of the tag library. The samples are grouped into the following categories:

- **Reporting.** Using PASW BIRT Report Designer and visualization reports interactively, including running a second report in response to a selection. To access these samples, go to:

  http://<server-name>:<port>/reportTagLib/index.html

- **Scoring.** Generating scores for a predictive model configured for scoring. To access these samples, go to:

  http://<server-name>:<port>/scoringTagLib/index.html

- **PASW Statistics syntax.** Generating and executing PASW Statistics syntax, as well as working with the resulting output. To access these samples, go to:

  http://<server-name>:<port>/spssSyntaxTagLib/index.html

Note: This functionality requires PASW Statistics adapters in the PASW Collaboration and Deployment Services environment. For more information, see the PASW Statistics installation documentation.

If the URL for a set of samples fails to return an introduction page, the package or war file containing the samples may not be deployed to the repository server. Use the Package Manager tool to deploy the desired package or deploy the war file in accordance with the documentation for your application server. The war files to be deployed are under the *./components/paswTagLib/apps* and *./components/paswTagLib/scoring* directories of the repository installation.

On the introduction page for the samples, click View Source for any sample to examine its source code. To explore their functionality, you can run the samples from the page by clicking Run. However, successful execution requires:

- sample resources in a specific folder structure in the repository.
- valid credentials for accessing the resources referenced in the samples.

Instructions for configuring the environment for successful sample execution are available from the introduction page for the samples.

Chapter
5

Portal Integration

The PASW Collaboration and Deployment Services web services architecture provides the ability to integrate it with portal servers. This enables delivery of highly customized content through pluggable user interface components that utilize Web services to produce fragments of markup code that are aggregated into a portal page. Typically, a portal page is displayed as a collection of non-overlapping windows, where each window displays a segment of content. Some examples of portal applications are e-mail, weather reports, discussion forums, and news. Similarly, PASW Collaboration and Deployment Services portals can be used to deliver customized content, such as output of reports and analytical processing, charts, diagrams, etc.

PASW Collaboration and Deployment Services supports portal integration based on JSR 268 standard. JSR 268, proposed by Java Community Process group (*http://jcp.org*), enables interpretability for portlets between different Web portals. This specification defines a set of APIs for interaction between the portlet container and the portlet, addressing the areas of personalization, presentation and security. Implementation of JSR 168 include IBM Web Portal from WebSphere, Oracle Application Server Portal 10g, BEA WebLogic Portal, Vignette Portal, Sun Portal Server, and JBoss. PASW Collaboration and Deployment Services also supports portal integration with Microsoft SharePoint server using Web Parts.

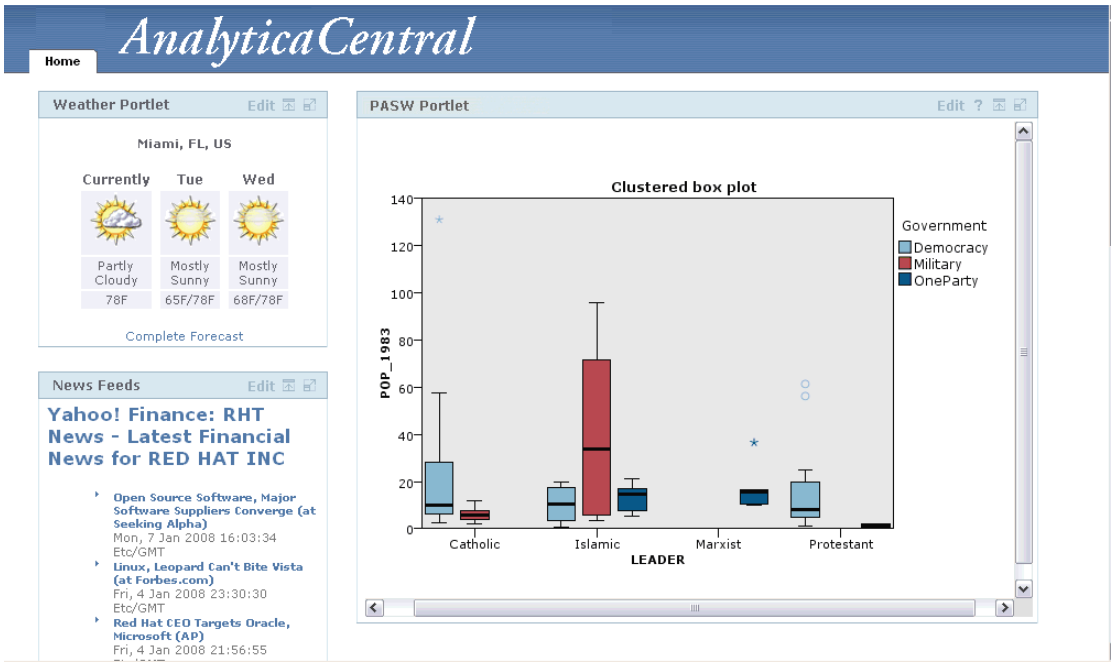
Officially supported portal environments include:

- WebSphere Portal Server 6.1
- Weblogic Portal Server 10.0
- JBoss Portal Server 2.6.1
- Sun Java Enterprise System 5
- Microsoft Sharepoint 2007 Server

PASW Collaboration and Deployment Services may also be integrated with other portal environments based on JSR 168 and J2SE 5.0.

PASW Collaboration and Deployment Services is packaged with a portlet and a Web Part which can be used to deliver repository content to portal users.

Figure 5-1
PASW Collaboration and Deployment Services portlet displaying repository content in a portal server window



The architecture of PASW Collaboration and Deployment Services also enables creation of custom JSR 168-compliant portlets and SharePoint Web Parts that use PASW Collaboration and Deployment Services Web services.

Installation

PASW Collaboration and Deployment Services portal components are distributed on the repository installation Disk 2 in */PORTLET* as *PASWPortlet.war* (portlet) and *PASWWebPart.wsp* (Web Part).

Portlet installation

- The procedure for installing *PASWPortlet.war* varies depending on the portal server type. Refer to portal server vendor documentation for details.

Web Part installation

SharePoint Web Part installation prerequisites include:

- Microsoft SharePoint 2007
- Microsoft Web Service Enhancement 2.0 (WSE 2.0 SP3)

To install the Web Part:

1. Copy *PASWWebPart.wsp* from the repository installation disk to a predefined location on the SharePoint host, for example, *c:\temp*.

2. From the */bin* directory of the SharePoint server installation run the following commands:

   stsadm -o addsolution -filename c:\tmp\paswwebpart.wsp
   stsadm -o deploysolution -name paswwebpart.wsp -immediate -allowgacdeployment -url http://<hostname>

3. Use SharePoint administration utilities to add the Web Part to the Web Part gallery and to subsequently deploy it. For more information, see Microsoft SharePoint documentation.

   Once the component has been installed, it must be configured to access a specific resource in repository. Component preferences must also be set up.

Configuration

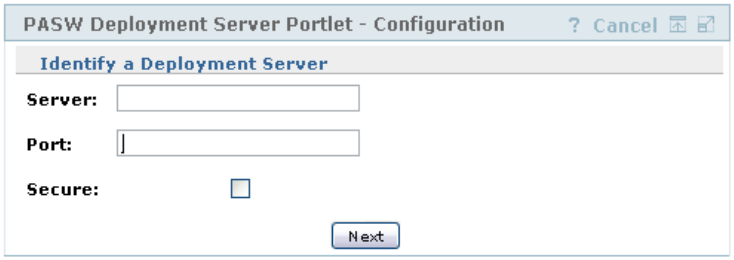
After the portal component has been installed and the portal page layout has been completed, you will be prompted to configure the component to access a repository resource. The general procedure for configuring portal access to PASW Collaboration and Deployment Services involves defining the repository server, specifying repository credentials, selecting the resource to be delivered to the portal, and if necessary, specifying data source credentials and default prompt values. You can also configure components' appearance and behavior by setting the preferences.

Configuring the portlet

Open the portlet configuration page. The page may open differently depending on the portal server type.

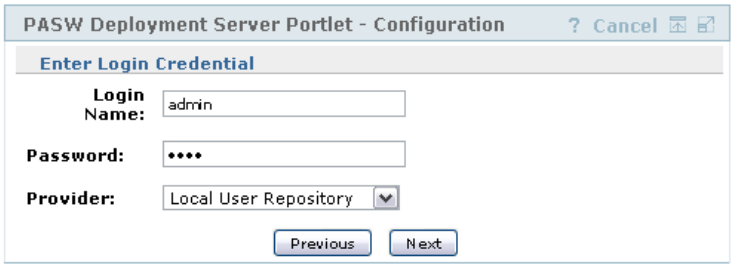
1. Specify the repository host and port and whether the server requires a secure connection.

Figure 5-2
Portlet configuration: specifying the repository



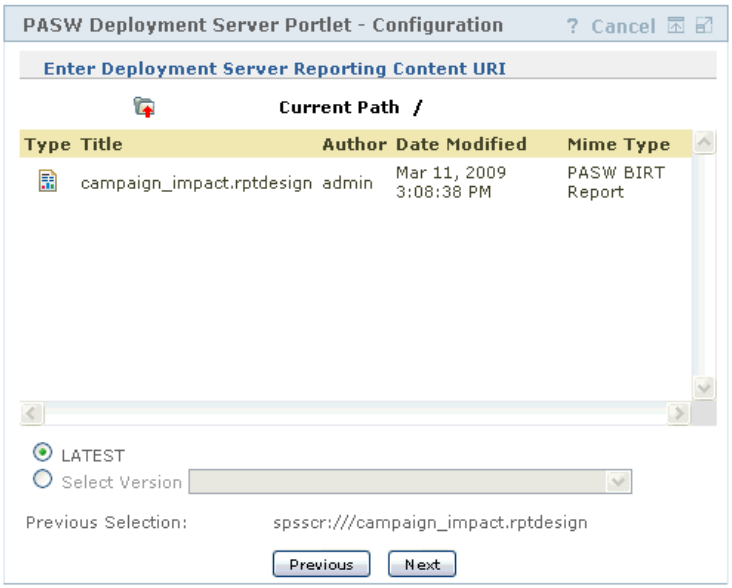
2. Specify the PASW Collaboration and Deployment Services user credentials and security provider for login authentication.

Figure 5-3
Portlet configuration: specifying repository credentials



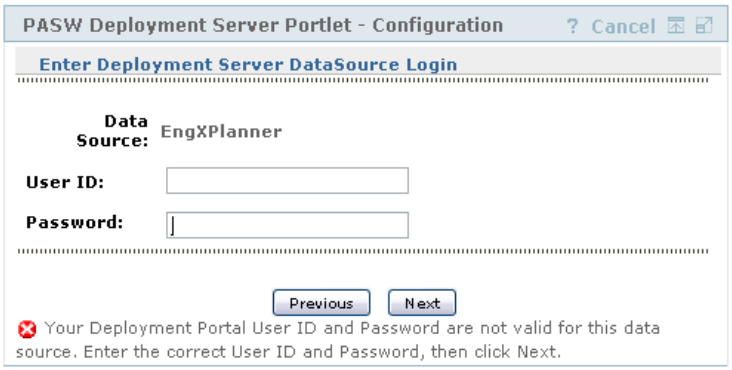
3. Select the repository resource to be delivered to the portal. Make sure the correct resource version is specified.

Figure 5-4
Portlet configuration: selecting the resource



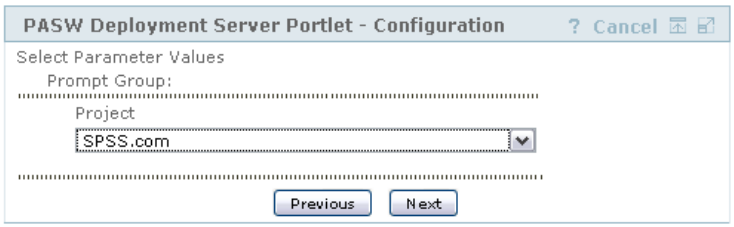
4. If necessary, specify the credentials for the data source referenced by the resource; for example if a report uses a database, database credentials must be provided. Note that depending on the resource, it may be necessary to specify credentials for multiple data sources

Figure 5-5
Portlet configuration: specifying data source credentials



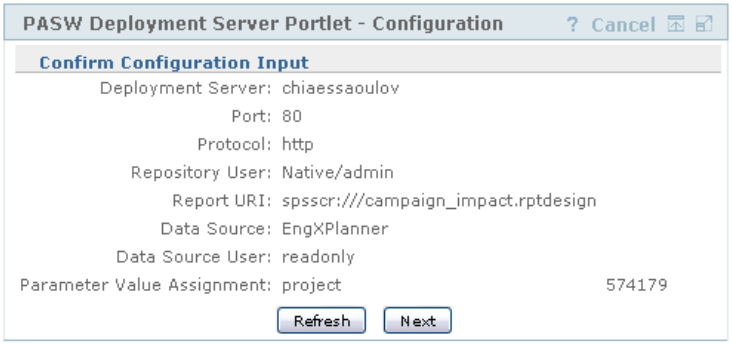
5. If the resource includes prompts (for example, a report may allow for a dynamic selection of values), specify the default prompt settings.

Figure 5-6
Portlet configuration: setting default prompt values



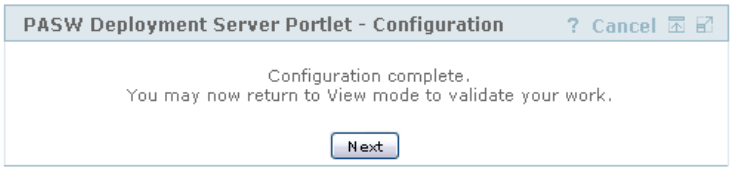
6. Verify that configuration information is correct. To start over, click Refresh.

Figure 5-7
Portlet configuration: confirmation page



7. Click Next to proceed to viewing the resource.

Figure 5-8
Portlet configuration: completion message



Portlet settings can been edited after the initial configuration: for example, it can be pointed to a different repository resource if necessary.

- Certain aspects of the appearance and behavior of the portlet are set through its preferences. The following preferences are available:

Preference	Description
expiration-cache	The expiration period for the portlet cache, i.e., the time in seconds after which the portlet output expires. -1 indicates that the output never expires. The default value is 600.
log-messages	Specifies whether portlet messages will be appended to the portal server log file. The default value is NO.
reenter-dsLogin	Specifies whether the user must provide the data source credentials for the portlet instance every time she logs into the portal. The default value is NO.
reenter-parameter	Specifies whether the user must reenter the prompt values for the portlet instance every time she logs into the portal. The default value is NO.
refresh-parameter	Specifies whether the user can enter different parameter values and re-display the content based on those values. The default value is NO.
use-single-sign-on	Specifies whether the portlet will be used with single sign-on. The default value is NO.
validate-input-parameter	Enables user input parameter validation in order to protect against cross-site scripting attacks. The default value is YES.
window-height	The height of the portlet window (pixels). The default value is 750.
window-title	Descriptive name for the portlet instance.
window-width	The width of the portlet window (percent). The default value is 100%.

Preferences are set with portal server administration facilities and the way they are accessed will differ depending on the server type.

Configuring the Web Part

Web Part configuration involves the same basic steps as the portlet configuration: setting up access to the repository resource and configuration option. Note that the number of displayed items in the repository tree (when you select the resource) is controlled by an additional configuration option.

Single Sign-on

PASW Collaboration and Deployment Services allows single sign-on access, and special configuration of the portal server may be required to enable it for the portlet or Web part. The procedures for enabling single sign-on will be different depending on the portal server. For example, JBoss portal configuration is as follows:

- *<JBoss installation folder>/bin/run.bat* file must be modified to include Java arguments for Kerberos-based single sign-on as in the example below.

```
set SSO_OPTS=-Djava.security.krb5.realm=SSOREALM.COM
-Djava.security.krb5.kdc=kdchost.ssorealm.com
-Djavax.security.auth.useSubjectCredsOnly=false

set JAVA_OPTS=%JAVA_OPTS% -Dprogram.name=%PROGNAME% %SSO_OPTS%

set JAVA_OPTS=%JAVA_OPTS% -Xms128m -Xmx512m -XX:PermSize=64m -XX:MaxPermSize=256m
```

- The following section must be added to *<JBoss installation folder>/server/default/conf/login-config.xml*.

```
<application-policy name="com.sun.security.jgss.initiate">
        <authentication>
          <login-module code = "com.sun.security.auth.module.Krb5LoginModule" flag = "required">
          <module-option name="useTicketCache">true</module-option>
      <module-option name="debug">false</module-option>
          </login-module>
        </authentication>
  </application-policy>
```

Known Issues

- When PASW Collaboration and Deployment Services portlet is used with JBoss portal, the repository tree view may not expand. In order to correct the problem, modify the *<JBoss installation folder>/bin/run.bat* (*run.sh* on UNIX) to increase the new generation and permanent generation size by adding the following arguments to JAVA_OPTS:

```
-XX:MaxNewSize=256m -XX:MaxPermSize=256m
```

- Cookie settings in the Safari browser may prevent some repository artifacts from being displayed in the PASW Collaboration and Deployment Services portlet without first prompting for credentials. The browser cookie policy should be set to *Always* instead of *Only from sites I visit* to avoid repeated requests for credentials.

Chapter 6

Scripting

PASW Collaboration and Deployment Services provides a scripting framework with a set of Content Repository and Process Management APIs that advanced users and administrators can use to write independent routines or batch jobs that combine a set of routines. This can greatly simplify bulk tasks such as changing security permissions for a large group of users, labeling or removing a label from a large number of folders/files, or uploading/downloading a large number of folders/files. The framework includes the ability to perform tasks from the command line, as well as a rich API for interacting with PASW Collaboration and Deployment Services within your own Python code.

For general information about Python, a dynamic object-oriented programming language, see the Python site (*http://www.python.org*).

The scripting framework can be used on Windows, Unix, and iSeries platforms. For instructions on installing the framework, consult the repository installation document for the desired platform.

Command Line Scripting

The Python file *CADSTool.py* can be used from the command line to manipulate resources stored within the repository. The general syntax used for calling PASW Collaboration and Deployment Services scripting operations from the command line is:

```
python CADSTool.py <Operation> <Keywords>
```

Where:

- `<Operation>` designates the function to invoke
- `<Keywords>` defines keyword/value pairs used as input parameters to the function

Global Keywords

Table 6-1 lists the keywords supported by all PASW Collaboration and Deployment Services scripting functions. The second column lists any optional, shortened versions of the keywords. Note that keywords are case sensitive.

Table 6-1
Global Keywords

Keyword	Optional Short Version	Usage
`--user`	`-u`	The user name to connect to the repository server
`--password`	`-p`	The password to connect to the repository server
`--host`	`-q`	The host/server name where the repository is installed

Keyword	Optional Short Version	Usage
`--port`	`-o`	The repository server port number
`--useDefault`	`-z`	Indicates that user, password, host, and port need to be read from the *Authorization.properties* file
`-h`		The scripting module help information

Repository Connections

You must specify the repository user ID, password, host, and port at the end of every command. There are two ways to provide this connection information:

- Using keywords, such as
  `--user <user> --password <password> --host <host> --port <port>`
- Through the *Authorization.properties* file, where the command contains a `--useDefault` parameter (or the short version `-z`). This retrieves the connection information from the *Authorization.properties* file, which is located at *<Scripting folder>\Lib\site-packages\config\Authorization.properties*. Use a simple text editor to modify the following values in the file to match the settings of your repository:

```
# Authorization Information
user=admin
password=spss
host=yourhost
port=80
```

Parameters passed through the command line always have precedence.

- If `--user` and `--password` are provided via the command line and the `--useDefault` or `-z` parameter is also provided, the user and password from the command line are used, with the host and port retrieved from the *Authorization.properties* file.
- If all the configurable information (`--user <user> --password <password> --host <host> --port <port>`) is provided via the command line, but the `--useDefault` or `-z` parameter is also used, the `--useDefault` is ignored and only the command line information is used.

For all APIs described in this guide, the syntax and examples use the `-z` parameter in an effort to use the minimum number of required parameters.

Content Repository Functions

This section outlines the Python command line usage of scripts for repository functions. Every operation contains detailed syntax information, an example, and expected messages.

Keywords

Table 6-2 lists the keywords supported for repository functions. The second column lists any optional, shortened versions of the keywords.

Important: Keywords are case sensitive.

Table 6-2
Keywords for repository APIs

Keyword	Optional Short Version	Usage
`--source`	`-s`	The source file/folder path
`--target`	`-t`	The target folder path
`--version`	`-v`	The version of a file
`--principal`	`-r`	The user who needs to be granted permission
`--permission`	`-n`	The permission type (such as read, write, modify, delete)
`--label`	`-l`	The label to assign to a version of a file
`--criteria`	`-c`	The search criteria for searching metadata attributes of files/folders
`--author`	`-a`	The author name for a file/folder
`--description`	`-d`	The description for a file/folder
`--title`	`-i`	The title for a file/folder
`--expirationDate`	`-q`	The expiration date for a file/folder
`--expirationStartDate`		The expiration start date for a file/folder
`--expirationEndDate`		The expiration end date for a file/folder
`--keyword`	`-k`	The keyword for a file/folder
`--cascade`	`-x`	Indicates that security settings for a folder should propagate to subfolders and files
`--provider`	`-f`	The security provider used to retrieve the users/principals
`--createVersion`	`-b`	Indicates that a new version of a file is to be created
`--contentLanguage`	`-g`	The content language for a file/folder
`--topic`		The topic(s) assigned to a file/folder. You can enter multiple values such a `--topic "topic1;topic2"`
`--modifiedBy`		The user who modified a file/folder
`--mimeType`		The mime type of a file
`--createdBy`		The user who created a file/folder
`--submittedHierarchy`		Indicates whether to search the Submitted Jobs folder
`--propertyName`		The name of a custom property
`--customProperty`		The name/value pair of a custom property to be updated
`--propertyName`		The name of the custom property to retrieve valid values for

For all operations that accept label and version information, the user should either specify a label or a version, but not both. If no version or label is specified for a given folder/file, the latest version is used.

Operations

The following sections list all repository scripting operations supported for PASW Collaboration and Deployment Services.

advanceSearch

Searches for files and folders in the repository, based on various parameters. Note that currently `expirationStartDate` and `expirationEndDate` do not work when used in conjunction with other search fields (such as title, author, etc).

Syntax

```
python CADSTool.py advanceSearch --author <author>
    --title <title> --description <description>
    --createdBy <createdBy> --modifiedBy <modifiedBy>
    --keyword <keyword> --label <label>
--topic <topic>
--expirationStartDate <expirationStartDate>
--expirationEndDate <expirationEndDate>
--submittedHierarchy -z
```

Where:

- `<author>` is the name of the author.
- `<title>` is the title of the file/folder.
- `<description>` is the description of the file/folder.
- `<createdBy>` is the name of the user who created the file/folder.
- `<modifiedBy>` is the name of the user who modified the file/folder.
- `<keyword>` is the keyword associated with the file/folder.
- `<label>` is the label for the version marker.
- `<topic>` is the topic associated with the file/folder.
- `<expirationStartDate>` is the expiration start date of the file/folder. The date format is `YYYY-MM-DDThh:mm:ss.sTZD` (for example, `1997-07-16T19:20:30.45+01:00`), where:
  `YYYY` = four-digit year
  `MM` = two-digit month (`01` is January, etc.)
  `DD` = two-digit day of month (`01` through `31`)
  `hh` = two-digit hour (`00` through `23`, no am/pm)
  `mm` = two-digit minute (`00` through `59`)
  `ss` = two-digit second (`00` through `59`)
  `s` = digits representing a decimal fraction of a second, with a valid range of `0` to `999`
  `TZD` = time zone designator (`Z` or `+hh:mm` or `-hh:mm`)
- `<expirationEndDate>` is the expiration end date of the file/folder. The date format is `YYYY-MM-DDThh:mm:ss.sTZD`.
- `<submittedHierarchy>` indicates the file/folder is in the Submitted Jobs folder.

All parameters are optional.

Example

```
python CADSTool.py advanceSearch --author "admin" --title "demo" --label "Label 1"
--useDefault -z
```

Messages

The following messages may display when using this API:

- When the API completes successfully, a list of all files and folders matching the search criteria is displayed. This typically includes the file names with their fully qualified path and versions.
- `Error searching files and folders`

applySecurity

Sets the security access control list (ACL) for a file/folder in the repository.

Note: For all security related operations, three web services are involved: Directory Information Service, Provider Information Service, and repository Service. For more information about the web services, see the web services documentation included with PASW Collaboration and Deployment Services.

Syntax

```
python CADSTool.py applySecurity --source "<source>" --principal "<principal>"
--permission "<permission>" --provider "<provider>" --cascade -z
```

Where:

- `<source>` is the fully qualified repository path of the file/folder to apply the security ACL to. This is a required parameter.
- `<principal>` is the user (such as *admin*) to apply to the specified file/folder as part of the ACL. This is a required parameter.
- `<permission>` is the type of permission to apply to the specified file/folder (such as read, write, modify, delete, or owner). This is a required parameter.
- `<provider>` is the security provider to use for retrieving information about the users (principals). This is an optional parameter.
- `--cascade` is used when setting security on a folder, to propagate the security settings to all files and subfolders within the specified folder. This is an optional parameter.

Examples

- The following example applies security to a file/folder:

```
python CADSTool.py applySecurity --source "/Temp Folder/Temp.txt" --principal Joe
--permission modify_acl --provider Native -z
```

- The following example applies security to a folder and all its files and subfolders:

```
python CADSTool.py applySecurity --source "/Temp Folder/" --principal Joe
--permission modify_acl --provider Native --cascade -z
```

Messages

The following messages may display when using this API:

- `<permission> permission set successfully for <source>.`

- <source> No such file/folder exists. Please try again.
- <permission> Invalid permission type, Please try again.
- <source> Error setting security ACL.

cascadeSecurity

Propagates a folder's security settings to all files and subfolders within the folder.

Syntax

```
python CADSTool.py cascadeSecurity --source "<source>" -z
```

Where:

- <source> is the fully qualified path of the folder in the repository. This is a required parameter.

Example

```
python CADSTool.py cascadeSecurity --source "/Temp Folder/" -z
```

Messages

The following messages may display when using this API:

- Security ACL cascaded successfully for <source>.
- <source> No such folder exists. Please try again.
- <source> Error cascading security ACL.

copyResource

Copies a file or folder to another folder in the repository. A renaming feature is provided for this API, where the specified file/folder can be renamed when it is copied. The cases described at the beginning of moveResource on p. 81 also apply to this copyResource API.

Syntax

```
python CADSTool.py copyResource --source "<source>" --target "<target>" -z
```

Where:

- <source> is the fully qualified Content Repository path of the file/folder to copy. This is a required parameter.
- <target> is the fully qualified repository path where the file/folder is to be copied. This is a required parameter.

Examples

- The following example copies a file:

```
python CADSTool.py copyResource --source "/Temp Folder/Temp.txt" --target
```

```
"/Sample Folder" -z
```

- The following example copies and renames a file:

```
python CADSTool.py copyResource --source "/Temp Folder/Temp.txt" --target
"/Sample Folder/New.txt" -z
```

Messages

The following messages may display when using this API:

- `<source> copied successfully.`
- `<source> No such file/folder exists.  Please try again.`
- `<target> No such folder exists.  Please try again.`
- `<source> Error copying file/folder.`

createFolder

Creates a new folder at a specified location in the repository.

Syntax

```
python CADSTool.py createFolder --source "<source>" -z
```

Where:

- `<source>` is the fully qualified path of the new folder to create. This is a required parameter. Based on the provided path, the new folder is created, including any subfolders.

Example

- The following example creates *Temp Folder* if it does not already exist.

```
python CADSTool.py createFolder --source "/Temp Folder/Sample Folder" -z
```

Messages

The following messages may display when using this API:

- `<source> Folder created successfully.`
- `<source> No such folder exists.  Please try again.`
- `<folder> Folder already exists.  Please try again.`
- `<source> Error creating folder.`

deleteFile

Deletes a file from the repository, including all its versions.

Syntax

```
python CADSTool.py deleteFile --source "<source>" --submittedHierarchy -z
```

Where:

- <source> is the fully qualified repository path of the file to delete. This is a required parameter.
- --submittedHierarchy deletes a file from the Submitted Jobs folder. This is an optional parameter.

Example

- The following example deletes a file from the repository, including all its versions:

```
python CADSTool.py deleteFile --source "/Temp Folder/Temp.txt" -z
```

- The following example deletes a file from the Submitted Jobs folder, including all its versions:

```
python CADSTool.py deleteFile --source "Submitted Jobs/admin/
2007-05-21.14.10.22.422-test.dbq/test.dbq.html" --submittedHierarchy -z
```

Messages

The following messages may display when using this API:

- <source> deleted successfully.
- <source> No such file exists. Please try again.
- <source> Error deleting file.

deleteFileVersion

Deletes a specific version of a file from the repository.

Syntax

```
python CADSTool.py deleteFileVersion --source "<source>" --version "<version>"
--label "<label>" --submittedHierarchy -z
```

Where:

- <source> is the fully qualified repository path of the file to delete. This is a required parameter.
- <version> is the specific version of the file to delete. This is an optional parameter.
- <label> is the label of the file to delete. This is an optional parameter.
- --submittedHierarchy deletes a specific version of a file from the Submitted Jobs folder. This is an optional parameter.

Examples

- The following example deletes a specific version of a file:

```
python CADSTool.py deleteFileVersion --source "/Temp Folder/Temp.txt" --version
"0:2006-08-25 21:15:49.453" -z
```

- The following example deletes a file with a specific label:

```
python CADSTool.py deleteFileVersion --source "/Temp Folder/Temp.txt" --label
```

```
"version1" -z
```

- The following example deletes a file with a specific label from the Submitted Jobs folder:

```
python CADSTool.py deleteFileVersion --source "Submitted Jobs/admin/
2007-05-21.14.10.22.422-test.dbq/test.dbq.html" --label "LATEST" -z
```

Messages

The following messages may display when using this API:

- `<source> deleted successfully.`
- `<source> No such file exists.  Please try again.`
- `<source> Error deleting file.`

deleteFolder

`deleteFolder` deletes a folder from the repository, including all its contents.

Syntax

```
python CADSTool.py deleteFolder --source <source> --submittedHierarchy -z
```

Where:

- `<source>` is the fully qualified repository path of the folder to delete. This is a required parameter.
- `--submittedHierarchy` deletes a specific version of the folder from the Submitted Jobs folder. This is an optional parameter.

Examples

- The following example deletes a folder:

```
python CADSTool.py deleteFolder --source "/Temp Folder/" -z
```

- The following example deletes a folder from the Submitted Jobs folder:

```
python CADSTool.py deleteFolder --source "Submitted Jobs/admin/
2007-05-21.14.10.22.422-test.dbq/" --submittedHierarchy -z
```

Messages

The following messages may display when using this API:

- `<source> deleted successfully.`
- `<source> No such folder exists.  Please try again.`
- `<source> Error deleting folder.`

downloadFile

Downloads a specific version of a file from the repository onto the local file system.

Syntax

```
python CADSTool.py downloadFile --source "<source>" --version "<version>" --label
"<label>" --target "<target>" -z
```

Where:

- <source> is the fully qualified repository path or Object URI of the folder containing the file to download. The Object URI can be obtained by viewing the properties of a folder in Deployment Manager. This is a required parameter.
- <version> is the version of the file to download. This is an optional parameter.
- <label> is the label of the file to be downloaded. This is an optional parameter.
- <target> is the fully qualified path (on the local file system) where the file is to be downloaded.

Examples

- The following example downloads the latest version of the file:

```
python CADSTool.py  downloadFile --source "/Temp Folder/Temp.txt" --target "C:\Temp\" -z
```

- The following example downloads a specific version of the file using a version marker:

```
python CADSTool.py  downloadFile --source "/Temp Folder/Temp.txt" --version
"0:2006-08-25 21:15:49.453" --target "C:\Temp\" -z
```

- The following example downloads a labeled version of the file:

```
python CADSTool.py  downloadFile --source "/Temp Folder/Temp.txt" --label "version 1"
--target "C:\Temp\" -z
```

Messages

The following messages may display when using this API:

- <source> File downloaded successfully.
- <source> No such file exists. Please try again.
- <target> No such folder exists. Please try again.
- <source> Error downloading File.

export

Starts an export from the Content Repository, allowing you to select which files and folders to export, and saving the **.pes* export file to the local file system.

Syntax

```
python CADSTool.py export --source "<source>"  --target "<target>"  -z
```

Where:

- `<source>` is the fully qualified repository path of the folder to export. This is a required parameter.
- `<target>` is the fully qualified path (on the local file system) for the *.*pes* export file to create. This is a required parameter.

Example

```
python CADSTool.py export --source "/Temp Folder/" --target "C:\Demo\Temp.pes" -z
```

Messages

The following messages may display when using this API:

- `<source> exported successfully.`
- `<source> No such folder exists.  Please try again.`
- `<source> Error exporting folder.`

getAllVersions

Retrieves a list of all versions of a file in the repository.

Syntax

```
python CADSTool.py  getAllVersions --source "<source>" --submittedHierarchy -z
```

Where:

- `<source>` is the fully qualified repository path of the file to retrieve versions for. This is a required parameter.
- `--submittedHierarchy` retrieves versions from the Submitted Jobs folder. This is an optional parameter.

Examples

- The following example retrieves all versions of a specified file:

```
python CADSTool.py  getAllVersions --source "/Temp Folder/Temp.txt" -z
```

- The following example retrieves all versions of a specified file from the Submitted Jobs folder:

```
python CADSTool.py  getAllVersions --source "Submitted Jobs/admin/
2007-05-21.14.10.22.422-test.dbq/test.dbq.html" --submittedHierarchy" -z
```

Messages

The following messages may display when using this API:

- `<source> No such file exists.  Please try again.`

- <source> Error retrieving file versions.
- When the process completes successfully, the information for every file version is displayed, including version marker and label information.

getAccessControlList

Retrieves the security access control list (ACL) for a specified file/folder in the Content Repository.

Syntax

```
python CADSTool.py getAccessControlList --source "<source>" -z
```

Where:

- <source> is the fully qualified path of the file/folder. This is a required parameter.

Example

```
python CADSTool.py getAccessControlList --source "/Temp Folder/Temp.txt" -z
```

Messages

The following messages may display when using this API:

- <source> No such file/folder exists. Please try again.
- Error retrieving security details for <source>.

getChildren

Retrieves the list of all files and folders in a specified folder of the repository.

Syntax

```
python CADSTool.py getChildren --source "<source>" -z
```

Where:

- <source> is the fully qualified path of the folder. This is a required parameter.

Example

```
python CADSTool.py getChildren --source "/Temp Folder" -z
```

Messages

The following messages may display when using this API:

- When the command completes successfully, it lists all contents of the specified folder.
- <source> No such folder exists. Please try again.
- <source> Error getting resources.

getCustomPropertyValue

Retrieves the valid values accepted by a specified custom property.

Syntax

```
python CADSTool.py getCustomPropertyValue --propertyName
"<propertyName>" --propertyName "<propertyName>" -z
```

Where:

- `<propertyName>` is the name of the custom property. This is an optional parameter.

Example

```
python CADSTool.py getCustomPropertyValue --propertyName "Custom Number" -z
```

Messages

The following messages may display when using this API:

- `<propertyName> takes values as <valid values>`
- `Error retrieving property details for <propertyName>.`

getMetadata

Retrieves the metadata attributes of a file or folder in the repository.

Syntax

```
python CADSTool.py getMetadata --source "<source>" --version "<version>" --label
"<label>" --submittedHierarchy -z
```

Where:

- `<source>` is the fully qualified repository path of the file/folder to retrieve metadata for. For folders, the version/label attributes are ignored. This is a required parameter.
- `<version>` is the version of the file/folder to retrieve metadata for. This is an optional parameter.
- `<label>` is the label of the file/folder to retrieve metadata for. This is an optional parameter.
- `--submittedHierarchy` retrieves metadata from the Submitted Jobs folder. This is an optional parameter.

Examples

- The following example retrieves metadata for a specific version of a file/folder:

```
python CADSTool.py getMetadata --source "/Temp Folder/Temp.txt" --version
"1:2006-08-25 21:15:49.453" -z
```

- The following example retrieves metadata for a labeled version of a file/folder:

```
python CADSTool.py getMetadata --source "/Temp Folder/Temp.txt" --label "version 1" -z
```

- The following example retrieves metadata for a labeled version of a file/folder in the Submitted Jobs folder:

```
python CADSTool.py getMetadata --source "Submitted Jobs/admin/
2007-05-21.14.10.22.422-test.dbq/test.dbq.html" --label "LATEST" --submittedHierarchy -z
```

Messages

The following messages may display when using this API:

- `<source> No such file exists.  Please try again.`
- `<source> Error retrieving file metadata.`
- When the process completes successfully, all metadata information for the specified file/folder is displayed, including any custom metadata properties.

import

Imports an existing **.pes* export file from the local file system to the repository.

Syntax

```
python CADSTool.py import --source "<source>" --target "<target>" -z
```

Where:

- `<source>` is the fully qualified path (on the local file system) of the **.pes* export file to import to the repository. This is a required parameter.
- `<target>` is the fully qualified repository path to import the **.pes* export file to. This is a required parameter.

Example

```
python CADSTool.py import --source "C:\Demo\Sample.pes" --target "/Temp Folder/" -z
```

Messages

The following messages may display when using this API:

- `<source> imported successfully.`
- `<source> No such file exists.  Please try again.`
- `<target> No such folder exists.  Please try again.`
- `<source> Error importing folder.`

moveResource

Moves a file or folder to another folder in the repository. A renaming feature is provided for this API, where the specified file/folder can be renamed when it is moved. The following cases describe the behavior of the renaming feature:

If the source is */Temp Folder/Temp.txt* and the target is */Demo Folder*:

- **Case 1**: If folder *Demo Folder* exists, *Temp.txt* is moved to *Demo Folder*.
- **Case 2**: If folder *Demo Folder* does not exist, *Temp.txt* is moved to " / " and renamed to *Demo Folder*.

If the source is */Temp Folder/Temp.txt* and the target is */Demo Folder/Abc.dat*:

- **Case 1**: If folder *Demo Folder* exists, *Temp.txt* is moved to *Demo Folder* and renamed to *Abc.dat*.
- **Case 2**: If folder *Demo Folder* does not exist, an error is displayed.

Syntax

```
python CADSTool.py moveResource --source "<source>" --target "<target>" -z
```

Where:

- `<source>` is the fully qualified repository path of the file/folder to move. This is a required parameter.
- `<target>` is the fully qualified repository path where the file/folder is to be moved. This is a required parameter.

Examples

- The following example moves a file:

```
python CADSTool.py moveResource --source "/Temp Folder/Temp.txt" --target
"/Sample Folder" -z
```

- The following example moves a folder:

```
python CADSTool.py moveResource --source "/Temp Folder/" --target "/Sample Folder -z
```

- The following example moves and renames a file:

```
python CADSTool.py moveResource --source "/Temp Folder/Temp.txt" --target
"/Sample Folder/New.txt" -z
```

Messages

The following messages may display when using this API:

- `<source> moved successfully.`
- `<source> No such file/folder exists.  Please try again.`
- `<target> No such folder exists.  Please try again.`
- `<source> Error moving file/folder.`

removeLabel

Removes a label from a file in the repository.

Syntax

```
python CADSTool.py removeLabel --source "<source>" --label "<label>" -z
```

Where:

- `<source>` is the fully qualified path of the file in the repository. This is a required parameter.
- `<label>` is the label name to remove from the specified file. This is a required parameter.

Example

```
python CADSTool.py removeLabel --source "/Temp Folder/Temp.txt" --label "version 1" -z
```

Messages

The following messages may display when using this API:

- `Label removed successfully for <source>.`
- `<source> No such folder exists.  Please try again.`
- `<source> Error deleting label.`
- `<label> No such label exists.  Please try again.`

removeSecurity

Removes the security access control list (ACL) from a specified file or folder in the repository.

Syntax

```
python CADSTool.py removeSecurity --source "<source>" --principal "<principal>"
--provider "<provider>" --cascade -z
```

Where:

- `<source>` is the fully qualified path of the file/folder to remove security from. This is a required parameter.
- `<principal>` is the user/principal (such as *admin*) to remove security from for the specified file/folder. This is a required parameter.
- `<provider>` is the security provider to use for retrieving information about the users (principals). This is an optional parameter.
- `--cascade` is used when removing security from a folder, to remove the security settings from all files and subfolders within the specified folder. This is an optional parameter.

Example

```
python CADSTool.py removeSecurity --source "/Temp Folder/Temp.txt" --principal Joe
--provider Native --cascade -z
```

Messages

The following messages may display when using this API:

- `<source> All the security ACL removed successfully.`
- `<source> No such folder exists.  Please try again.`
- `<source> Error deleting security ACL.`

search

Searches for files and folders in the repository. The results are a list of files/folders matching the search criteria, and their versions.

Note: The `search` API accepts a search string as an input parameter. The parameter value is used to search the metadata information of files and folders in the repository. The API returns a list of all files and folders with matching criteria.

Syntax

```
python CADSTool.py search --criteria "<criteria>" -z
```

Where:

- `<criteria>` is the search string used to search metadata for all files and folders in the Content Repository. This is a required parameter.

Example

```
python CADSTool.py search --criteria "Age" -z
```

Python Example

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")
     resourceSpecifierList = pesImpl.search
(
    "age",
 )
```

Messages

The following messages may display when using this API:

- When the search completes successfully, a list of all files and folders matching the search criteria are displayed. This typically includes the file names with their fully qualified path and versions.
- `<criteria> No file or folder matches the search criteria.`
- `Error searching files and folders.`

setLabel

Applies a label to a version of a file in the repository. If the file is already labeled, the original label is removed and replaced with the new label.

Syntax

```
python CADSTool.py setLabel --source "<source>" --version "<version>" --label
"<label>" -z
```

Where:

- `<source>` is the fully qualified path of the file in the repository. This is a required parameter.
- `<version>` is the version of the file to apply the label to. This is a required parameter.
- `<label>` is the label name to apply to the specified version of the file. This is a required parameter.

Example

```
python CADSTool.py setLabel --source "/Temp Folder/Temp.txt" --version
"1:2006-08-25 21:15:49.453" --label "versionNo3" -z
```

Messages

The following messages may display when using this API:

- `Label set successfully for <source>.`
- `<source> No such folder exists.  Please try again.`
- `<source> Error setting label.`

setMetadata

Applies metadata properties to files and folders in the repository. Table 6-3 lists the metadata properties and whether they can be applied to files and/or folders.

Table 6-3
Metadata properties and resource types

Metadata Property	Resource Type
Author	File
Description	File/Folder
Title	File/Folder
Expiration Date	File/Folder
Keyword	File
Topics	File
Custom Metadata	File/Folder

Syntax

```
python CADSTool.py setMetadata --source "<source>" --version "<version>" --label
"<label>" --author "<author>" --title "<title>" --description "<description>"
--expirationDate "<expirationDate>" --topic "<topic>" --keyword "<keyword>"
--customProperty "<customProperty>" -z
```

Where:

- <source> is the fully qualified repository path of the file or folder to set metadata on. This is a required parameter.
- <author> is the author of the file/folder. This is an optional parameter.
- <title> is the title of the file/folder. This is an optional parameter.
- <description> is the description of the file/folder. This is an optional parameter.
- <expirationDate> is the expiration date of the file/folder. This is an optional parameter. The date format is YYYY-MM-DDThh:mm:ss.sTZD (for example, 1997-07-16T19:20:30.45+01:00), where:
  YYYY = four-digit year
  MM = two-digit month (01 is January, etc.)
  DD = two-digit day of month (01 through 31)
  hh = two-digit hour (00 through 23, no am/pm)
  mm = two-digit minute (00 through 59)
  ss = two-digit second (00 through 59)
  s = digits representing a decimal fraction of a second, with a valid range of 0 to 999
  TZD = time zone designator (Z or +hh:mm or -hh:mm)
- <keyword> is the keyword for the file/folder. This is an optional parameter.
- <version> is the specific version of the file/folder to apply metadata on. This is an optional parameter.
- <label> is the labeled version of the file/folder to apply metadata on. This is an optional parameter.
- <topic> is the topic to apply to the file/folder. This is an optional parameter.
- <customProperty> is the custom property values to apply to the file/folder. This is an optional parameter. The values are specified as <customProperty>=<value>. To apply more than one custom property, use a semicolon (;) as a separator (<customProperty>=<value>;<customProperty>=<value>). Separate multi-select property values with the | operator (<customProperty>=opt1|opt2;<customProperty>=value).

Note: At least one optional parameter must be provided to use the setMetadata API.

Example

```
python CADSTool.py setMetadata --source "/Temp Folder/Temp.txt" --version
"0:2006-08-25 21:15:49.453" -label "version1" --author "Joe" --title "Title1.txt"
--description "Test File" --topic "topic1;topic2" --expirationDate "21-08-06"
--keyword "age" --customProperty "multi=hi|hello|bye;Complexity Degree=Simple" -z
```

Messages

The following messages may display when using this API:

- <source> Metadata set successfully.
- <source> No such file/folder exists. Please try again.
- <source> Error setting metadata.

uploadFile

`uploadFile` saves a file to the Content Repository from the local file system, with the option of creating a new version of the file if it already exists.

Syntax

```
python CADSTool.py  uploadFile --source "<source>" --target
"<target>" --createVersion -z
```

Where:

- `<source>` is the fully qualified path (on the local file system) of the file to upload. This is a required parameter.
- `<target>` is the fully qualified path of the folder in the repository where the file is to be uploaded. This is a required parameter.
- `--createVersion` indicates that the specified file already exits and a new version should be created. This is an optional parameter.

Examples

- In the following example, the `<target>` is a fully qualified path for *Temp Folder*:

```
python CADSTool.py  uploadFile --source "C:\Temp\Temp.txt" --target "/Temp Folder" -z
```

- If *Temp.txt* already exists in the*/Temp Folder*, use the `--createVersion` parameter:

```
python CADSTool.py  uploadFile --source "C:\Temp\Temp.txt" --target "/Temp Folder"
--createVersion -z
```

Messages

The following messages may display when using this API:

- `<source> File uploaded successfully.`
- `<source> No such file exists.  Please try again.`
- `<target> No such folder exists.  Please try again.`
- `<source> Error Uploading File.`

Process Management Functions

This section outlines the Python command line usage of scripts for process management functions. Every API contains detailed syntax information, an example, and expected messages.

Keywords

Table 6-4 lists the keywords supported for Process Management APIs. The second column lists any optional, shortened version of keywords provided. The table only lists keywords specific to Process Management APIs. For additional keywords that apply to both Process Management APIs and repository APIs, see Table 6-1 and Table 6-2.

Table 6-4
Keywords for Process Management APIs

Keyword	Optional Short Version	Usage
`--source`	`-s`	The source job, including the path
`--target`	`-t`	The target folder path
`--notification`	`-j`	Indicates that the job will run with notifications
`--async`	`-m`	Indicates that the job will run asynchronously
`--execId`	`-y`	The execution Id for the job
`--jobStepName`	`-q`	The job step name
`--log`		Indicates that logs should not be deleted. If used in conjunction with `--target`, logs are stored in a location specified by `--target`. Otherwise, logs are displayed inline.

Operations

The following sections list all Process Management scripting APIs supported for PASW Collaboration and Deployment Services. The syntax and examples shown contain the minimum number of required parameters.

executeJob

Runs a job synchronously or asynchronously based on the parameters passed. In the case of a synchronous run, the API does not return until the job completes. In the case of an asynchronous run, the API returns after the job starts.

Syntax

```
python CADSTool.py  executeJob --source "<source>" --notification --async -z
```

Where:

- `<source>` is the fully qualified path of the job in the repository. This is a required parameter.
- `--notification` is used to run the job with notifications. This is an optional parameter.
- `--async` is used to run the job asynchronously. This is an optional parameter.

Examples

- The following example runs the job synchronously without notifications:

```
python CADSTool.py  executeJob --source "/Temp Folder/Temp Job" -z
```

- The following example runs the job synchronously with notifications:

```
python CADSTool.py  executeJob --source "/Temp Folder/Temp Job" --notification -z
```

- The following example runs the job asynchronously without notifications:

```
python CADSTool.py  executeJob --source "/Temp Folder/Temp Job" --async -z
```

- The following example runs the job asynchronously with notifications:

```
python CADSTool.py  executeJob --source "/Temp Folder/Temp Job" --async --notification -z
```

Messages

The following messages may display when using this API:

- `<source> Job executed successfully.  Job execution Id is <execId>.`
- `<source> No such job exists.  Please try again.`
- `<source> Error executing job.`

getJobExecutionDetails

Lists run details for a specific job, including any job steps and iterations.

Syntax

```
python CADSTool.py  getJobExecutionDetails --execId "<execID>" --log  --target
"<target>" -z
```

Where:

- `--execId` is the execution Id of the job. This is a required parameter.
- `--log` indicates that the job log should be displayed inline. If the `--log` parameter is not included, any log generated by a job step run is not displayed. This is an optional parameter.
- `<target>` is the location (on the local file system) to store the logs. This is an optional parameter, and is only used in conjunction with the `--log` parameter.

Examples

- The following example lists the details of a specific job run:

```
python CADSTool.py  getJobExecutionDetails --execId "0a58c3710016a7860000010d1a6a87
b48400" -z
```

- The following example lists the details of a specific job run, with the log displayed inline:

```
python CADSTool.py  getJobExecutionDetails --execId "0a58c3710016a7860000010d1a6a87
b48400" --log -z
```

- The following example lists the details of a specific job run, with the job logs stored in a specific location:

```
python CADSTool.py  getJobExecutionDetails --execId "0a58c3710016a7860000010d1a6a87
b48400" --log --target "c:\logs" -z
```

Messages

The following messages may display when using this API:

- For a successful run, all run details are listed for the job, job steps, and job iterations. Logs are displayed inline or saved to a specified location on the local file system.
- `<execId> No such execution exists.  Please try again.`
- `<execId> Error displaying details of a job execution.`
- `--target cannot be used without --log parameter`

getJobExecutionList

Lists current runs and completed runs for a specific job, for all versions of the job.

Syntax

```
python CADSTool.py  getJobExecutionList --source "<source>" -z
```

Where:

- `<source>` is the fully qualified path of the job in the repository. This is a required parameter.

Example

```
python CADSTool.py  getJobExecutionList --source "/Temp Folder/Temp Job" -z
```

Messages

The following messages may display when using this API:

- For a successful run of the specified job, all run details such as execution Id, job name, job execution status, and job execution start and end time are listed.
- `<source> No such job exists.  Please try again.`
- `<source> Error displaying execution list for a job.`

API Reference

The PASW Collaboration and Deployment Services scripting framework allows interaction with repository objects directly within Python scripts. Within your Python code, import the `PESImpl` class from the `pes.api.PESImpl` module. Create a `PESImpl` object using the connection information for the repository to which to connect.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
```

Specific functions can then be accessed using the `pesImpl` object.

Content Repository APIs

This section outlines the `PESImpl` functions used for working with resources stored in the repository. Every function contains detailed syntax information, an example, and expected messages.

APIs

The following sections list all Content Repository scripting APIs supported for PASW Collaboration and Deployment Services.

Notes:

- For all APIs with optional parameters `Label` and `Version`, use either `Label` or `Version`, but not both. If no `Version` or `Label` is specified for a given file/folder, the latest version is used.
- For all APIs described in this guide that require a path to files/folders in the Content Repository, either the path or the Object URI can be used. The Object URI can be obtained by viewing the object properties in Deployment Manager.
- For methods requiring input of source or target repository or file system paths that contain non-Latin Unicode characters, the strings must be specified as Unicode objects, for example:

```
identificationSpecifier = pesImpl.uploadFile
(source=u'C:\Analytics\La Peña.txt',
 target=u'/La Peña')
```

advanceSearch Method

Searches for files and folders in the Content Repository, based on various parameters passed as input. Note that currently `expirationStartDate` and `expirationEndDate` do not work when used in conjunction with other search fields (such as title, author, etc).

You can search on the following items:

- Author
- Description
- Title
- Created By
- Modified By
- Expiration Start Date
- Expiration End Date
- Mime Type
- Label
- Keyword
- Topics

The following sections describe Python API usage.

Method Signature:

```
PageResult advanceSearch (criteriaDict, submittedHierarchy
) throws InsufficientParameterException
```

Input Parameters:

Table 6-5 lists the input parameters for the `advanceSearch` API.

Table 6-5
Input parameters for advanceSearch API

Field	Required?	Type	Example Value	Description
`criteriaDict`	Yes	Dictionary	`{ "author":"admin", "title":"search", "label":"label 1", }`	The dictionary contains the key/value of pair against which the search will be done. The acceptable key values are: • `author` • `title` • `description` • `createdBy` • `modifiedBy` • `expirationStartDate` • `expirationEndDate` • `mimeType` • `label` • `keyword` • `topic`
`submittedHierarchy`	No	Boolean	`True` or `False`	Indicates whether the file/folder is in the Submitted Jobs folder

Information Returned:

Table 6-6 lists the information returned by the `advanceSearch` API.

Table 6-6
Information returned by advanceSearch API

Type	Description
`PageResult`	See PageResult on p. 123.

Exceptions:

Table 6-7 lists possible exceptions returned by the `advanceSearch` API.

Table 6-7
Possible exceptions for advanceSearch API

Type	Description
`InsufficientParameterException`	This exception is displayed if the mandatory parameters are not specified.

Code Snippet:

The `advanceSearch` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `advanceSearch` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

# create a dictionary containing all search items.
criteriaDict = {
                'author':'admin',
                'title': 'search'
                'label' : 'label 1'
                'createdBy': 'admin',
                'modifiedBy': 'Mike'

            }
# here the files will be looked for the criteria "Age"
pageResult = pesImpl.advanceSearch(criteriaDict)

rows = pageResult.getRows()
for row in rows:
    print row.getAuthor()
    print row.getTitle()
    for childRow in row.getChildRow():
        print childRow.getVersionMarker()
        print childRow.getUri()
```

applySecurity Method

Sets the security ACL for a file/folder in the Content Repository. The following sections describe Python API usage.

Method Signature:

```
Boolean applySecurity (source, principal, permission, provider, cascade) throws
ResourceNotFoundException, InsufficientParameterException, IllegalParameterException
```

Input Parameters:

Table 6-8 lists the input parameters for the `applySecurity` API.

Table 6-8
Input parameters for applySecurity API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000 010dcee0eaa28219"`	The fully qualified path or Object URI of the file or folder in the Content Repository
`principal`	Yes	String	`admin`	The user (such as *admin*) to apply to the specified file/folder as part of the ACL
`permission`	Yes	String	`READ, WRITE, DELETE, MODIFY_ACL, OR OWNER`	The type of permission to apply to the specified file/folder

Field	Required?	Type	Example Value	Description
provider	No	String	Native	The security provider to use for applying security to users (such as *Native*)
cascade	No	Boolean	True or False	Propagates the security settings to all files and subfolders within the specified folder

Information Returned:

Table 6-9 lists the information returned by the applySecurity API.

Table 6-9
Information returned by applySecurity API

Type	Description
Boolean	True or False based on whether the API runs successfully.

Exceptions:

Table 6-10 lists possible exceptions returned by the applySecurity API.

Table 6-10
Possible exceptions for applySecurity API

Type	Description
ResourceNotFoundException	This exception is displayed if the source file does not exist.
InsufficientParameterException	This exception is displayed if any of the required parameters are not specified.
IllegalParameterException	This exception is displayed if the specified user or security provider name is incorrect.

Code Snippet:

The applySecurity API creates an object of class PESImpl by passing the user name, password, host, and port. The applySecurity API can be called on the instance of the PESImpl object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")
bSuccess = pesImpl.applySecurity
(
source="/Temp Folder/Temp.txt",
principal="Joe",
permission="modify_acl",
provider="Native"
)

# One can also specify the cascade flag to cascade the
# security permissions. By default cascade flag is
# false
bSuccess = pesImpl.applySecurity
(
```

```
source="/Temp Folder/",
principal="Joe",
permission="modify_acl",
provider="Native"
cascade="True"
```

cascadeSecurity Method

Propagates a folder's security settings to all files and subfolders within the folder. The following sections describe Python API usage.

Method Signature:

```
Boolean cascadeSecurity (source) throws ResourceNotFoundException,
InsufficientParameterException
```

Input Parameters:

Table 6-11 lists the input parameters for the `cascadeSecurity` API.

Table 6-11
Input parameters for cascadeSecurity API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder"` or `"0a58c3670016a7860000 010dcee0eaa28219"`	The fully qualified path or Object URI of the folder in the Content Repository

Information Returned:

Table 6-12 lists the information returned by the `cascadeSecurity` API.

Table 6-12
Information returned by cascadeSecurity API

Type	Description
`Boolean`	`True` or `False` based on whether the API runs successfully.

Exceptions:

Table 6-13 lists possible exceptions returned by the `cascadeSecurity` API.

Table 6-13
Possible exceptions for cascadeSecurity API

Type	Description
`ResourceNotFoundException`	This exception is displayed if the source folder does not exist.
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.

Code Snippet:

The `cascadeSecurity` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `cascadeSecurity` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

# here the source value is a fully qualified path
bSuccess = pesImpl.cascadeSecurity(source="/Temp Folder")

# alternatively here the source value is a ResourceID
bSuccess = pesImpl.cascadeSecurity ("0a58c3670016a7860000010dcee0eaa28219")
```

copyResource Method

Copies a file or folder to another folder in the Content Repository. The specified file/folder can be renamed when it is copied. The cases described at the beginning of moveResource Method on p. 112 also apply to this `copyResource` API. The following sections describe Python API usage.

Method Signature:

```
uri copyResource (source, target) throws ResourceNotFoundException,
InsufficientParameterException
```

Input Parameters:

Table 6-14 lists the input parameters for the `copyResource` API.

Table 6-14
Input parameters for copyResource API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000 010dcee0eaa28219"`	The fully qualified path or Object URI of the file or folder in the Content Repository
`target`	Yes	String	`"/New Folder"` or `"/New Folder/abc.dat"`	The fully qualified path or Object URI of the folder to copy the file to. A new file name can also be provided for renaming the specified file/folder when it is copied.

Information Returned:

Table 6-15 lists the information returned by the `copyResource` API.

Table 6-15
Information returned by copyResource API

Type	Description
`uri`	URI of the copied file/folder

Exceptions:

Table 6-16 lists possible exceptions returned by the `copyResource` API.

Table 6-16
Possible exceptions for copyResource API

Type	Description
`ResourceNotFoundException`	This exception is displayed if the source file or target folder does not exist.
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.

Code Snippet:

The `copyResource` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `copyResource` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

uri = pesImpl.copyResource
(
source="/Temp/Temp.txt",
target="/New Folder"
)

# following code snippet prints the uri.
print uri
```

createFolder Method

Creates a new folder at a specified location in the Content Repository. The following sections describe Python API usage.

Method Signature:

```
uri createFolder (source) throws InsufficientParameterException,
ResourceAlreadyExistsException
```

Input Parameters:

Table 6-17 lists the input parameters for the `createFolder` API.

Table 6-17
Input parameters for createFolder API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder/Sample Folder"`	The folder(s) to create in the Content Repository

Information Returned:

Table 6-18 lists the information returned by the `createFolder` API.

Table 6-18
Information returned by createFolder API

Type	Description
`uri`	URI of the folder created

Exceptions:

Table 6-19 lists possible exceptions returned by the `createFolder` API.

Table 6-19
Possible exceptions for createFolder API

Type	Description
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.
`ResourceAlreadyExistsException`	This exception is displayed if the specified folder already exists in the Content Repository.

Code Snippet:

The `createFolder` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `createFolder` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

# here creating a folder at the specified location
uri = pesImpl.createFolder
(
source="/Temp Folder/Sample Folder"
)
# following code snippet prints the uri
print uri
```

deleteFile Method

Deletes a file from the Content Repository, including all its versions. The following sections describe Python API usage.

Method Signature:

```
Boolean deleteFile (source, submittedHierarchy) throws ResourceNotFoundException,
InsufficientParameterException, IllegalParameterException
```

Input Parameters:

Table 6-20 lists the input parameters for the `deleteFile` API.

Table 6-20
Input parameters for deleteFile API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000 010dcee0eaa28219"`	The fully qualified path or Object URI of the file in the Content Repository
`sub-mit-ted-Hier-archy`	No	Boolean	`True` or `False`	Indicates whether the file is in the Submitted Jobs folder

Information Returned:

Table 6-21 lists the information returned by the `deleteFile` API.

Table 6-21
Information returned by deleteFile API

Type	Description
`Boolean`	`True` or `False` based on whether the API runs successfully.

Exceptions:

Table 6-22 lists possible exceptions returned by the `deleteFile` API.

Table 6-22
Possible exceptions for deleteFile API

Type	Description
`ResourceNotFoundException`	This exception is displayed if the source file or target folder does not exist.
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.
`IllegalParameterException`	This exception is displayed if the specified resource to delete is a folder.

Code Snippet:

The `deleteFile` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `deleteFile` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

bSuccess = pesImpl.deleteFile(source="/Temp/Temp.txt")
```

deleteFileVersion Method

Deletes a specific version of a file from the Content Repository. The following sections describe Python API usage.

Method Signature:

```
Boolean deleteFileVersion (source,version,label, submittedHierarchy) throws
ResourceNotFoundException, InsufficientParameterException, IllegalParameterException
```

Input Parameters:

Table 6-23 lists the input parameters for the `deleteFileVersion` API.

Table 6-23
Input parameters for deleteFileVersion API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000010dcee0eaa28219"`	The fully qualified path or Object URI of the file in the Content Repository
`version`	No. Either version or label must be specified.	String	`"0:2006-08-25 21:15:49.453"`	The specific version of the file to delete
`label`	No. Either version or label must be specified.	String	`"Version 1"`	The specific labeled version of the file to delete
`submittedHierarchy`	No	Boolean	`True` or `False`	Indicates whether the file is in the Submitted Jobs folder

Information Returned:

Table 6-24 lists the information returned by the `deleteFileVersion` API.

Table 6-24
Information returned by deleteFileVersion API

Type	Description
`Boolean`	`True` or `False` based on whether the API runs successfully.

Exceptions:

Table 6-25 lists possible exceptions returned by the `deleteFileVersion` API.

Table 6-25
Possible exceptions for deleteFileVersion API

Type	Description
`ResourceNotFoundException`	This exception is displayed if the source file or target folder does not exist.

Type	Description
InsufficientParameterException	This exception is displayed if any of the required parameters are not specified.
IllegalParameterException	This exception is displayed if the specified resource to delete is a folder.

Code Snippet:

The deleteFileVersion API creates an object of class PESImpl by passing the user name, password, host, and port. The deleteFileVersion API can be called on the instance of the PESImpl object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

# here delete the file by specifying version value
bSuccess = pesImpl.deleteFileVersion
(
source="/Temp/Temp.txt",
version="1:2006-08-25 21:15:49.453"
)

# here delete the file by specifying label value
bSuccess = pesImpl.deleteFileVersion
(
source="/Temp/Temp.txt",
label="version 1"
)
```

deleteFolder Method

Deletes a folder and all its contents from the Content Repository. The following sections describe Python API usage.

Method Signature:

```
Boolean deleteFolder (source, submittedHierarchy) throws ResourceNotFoundException,
IllegalParameterException, InsufficientParameterException
```

Input Parameters:

Table 6-26 lists the input parameters for the deleteFolder API.

Table 6-26
Input parameters for deleteFolderAPI

Field	Required?	Type	Example Value	Description
source	Yes	String	"/Temp Folder" or "0a58c3670016a7860000010dcee0eaa28219"	The fully qualified path or Object URI of the folder in the Content Repository
submittedHierarchy	No	Boolean	True or False	Indicates whether the folder is in the Submitted Jobs folder

Information Returned:

Table 6-27 lists the information returned by the `deleteFolder` API.

Table 6-27
Information returned by deleteFolder API

Type	Description
`Boolean`	`True` or `False` based on whether the API runs successfully.

Exceptions:

Table 6-28 lists possible exceptions returned by the `deleteFolder` API.

Table 6-28
Possible exceptions for deleteFolder API

Type	Description
`ResourceNotFoundException`	This exception is displayed if the folder does not exist.
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.
`IllegalParameterException`	This exception is displayed if the specified resource to delete is not a folder.

Code Snippet:

The `deleteFolder` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `deleteFolder` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

bSuccess = pesImpl.deleteFolder(source="/Temp Folder")
```

downloadFile Method

Downloads a specific version of a file from the Content Repository onto the local file system. The following sections describe Python API usage.

Method Signature:

```
resourceSpecifier downloadFile (source, target, version, label) throws
ResourceNotFoundException, InsufficientParameterException
```

Input Parameters:

Table 6-29 lists the input parameters for the `downloadFile` API.

Table 6-29
Input parameters for downloadFile API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000 010dcee0eaa28219"`	The fully qualified Content Repository path or Object URI of the file to download
`target`	Yes	String	`"C:\Temp"`	The fully qualified path (on the local file system) of the folder to download the file to
`version`	No. Either version or label can be specified.	String	`"0:2006-08-25 21:15:49.453"`	The specific version of the file to download
`label`	No. Either version or label can be specified.	String	`"Version 2"`	The specific labeled version of the file to download

Information Returned:

Table 6-30 lists the information returned by the `downloadFile` API.

Table 6-30
Information returned by downloadFile API

Type	Description
`Resource`	See Resource on p. 122.

Exceptions:

Table 6-31 lists possible exceptions returned by the `downloadFile` API.

Table 6-31
Possible exceptions for downloadFile API

Type	Description
`ResourceNotFoundException`	This exception is thrown if the source file or target folder does not exist.
`InsufficientParameterException`	This exception is thrown if any of the required parameters are not specified.

Code Snippet:

The `downloadFile` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `downloadFile` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")
# here we specify the version
resource =  pesImpl.downloadFile
(
source="/Temp Folder/Temp.txt",
target="c:/Temp",
version="0:2006-08-25 21:15:49.453",
)
```

```
     # Or alternatively a label can be specified.
resource = pesImpl.downloadFile
(
source="/Temp Folder/Temp.txt",
target="c:/Temp",
label="label 1"
)
# Or alternatively here the location of the file to
# download is passed as a ResourceID
resource = pesImpl.downloadFile
(
source="0a58c3670016a7860000010dcee0eaa28219",
target="c:/Temp",
label="label 1"
)
# Following snippet prints Author, Title and
# ResourceID
print resource.getAuthor()
print resource.getTitle()
print resource.getResourceID()
```

exportResource Method

Starts an export from the Content Repository, allowing the user to select which files and folders to export, and saving the **.pes* export file to the local file system. The following sections describe Python API usage.

Method Signature:

```
Boolean exportResource (source, target) throws ResourceNotFoundException,
InsufficientParameterException
```

Input Parameters:

Table 6-32 lists the input parameters for the `exportResource` API.

Table 6-32
Input parameters for exportResource API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder"` or `"0a58c3670016a78 60000010dcee0eaa2 8219"`	The fully qualified Content Repository path or Object URI of the folder to export
`target`	Yes	String	`"C:\Temp\backup.pes"`	The fully qualified path (on the local file system) and **.pes* file name to export the folder to

Information Returned:

Table 6-33 lists the information returned by the `exportResource` API.

Table 6-33
Information returned by exportResource API

Type	Description
`Boolean`	`True` or `False` based on whether the API runs successfully.

Exceptions:

Table 6-34 lists possible exceptions returned by the `exportResource` API.

Table 6-34
Possible exceptions for exportResource API

Type	Description
`ResourceNotFoundException`	This exception is displayed if the source file or target folder does not exist.
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.
`IllegalParameterException`	This exception is displayed if the specified target is a folder. Only a **.pes* file is allowed.

Code Snippet:

The `exportResource` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `exportResource` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

bSuccess = pesImpl.exportResource
(
source="/Temp Folder",
target="C:\Temp\backup.pes"
)
```

getAccessControlList Method

Retrieves the security ACL for the specified file/folder in the Content Repository. The following sections describe Python API usage.

Method Signature:

```
list getAccessControlList(source, submittedHierarchy) throws ResourceNotFoundException,
InsufficientParameterException
```

Input Parameters:

Table 6-35 lists the input parameters for the `getAccessControlList` API.

Table 6-35
Input parameters for getAccessControlList API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000010dcee0eaa28219"`	The fully qualified path or Object URI of the file/folder in the Content Repository
`submittedHierarchy`	No	Boolean	`True` or `False`	Indicates whether the file/folder is in the Submitted Jobs folder

Information Returned:

Table 6-36 lists the information returned by the `getAccessControlList` API.

Table 6-36
Information returned by getAccessControlList API

Type	Description
`Dictionary`	A dictionary is displayed containing the user name(s) and the associated permission. For example: `{"admin":"MODIFY_ACL","Joe":"DELETE"}`

Exceptions:

Table 6-37 lists possible exceptions returned by the `getAccessControlList` API.

Table 6-37
Possible exceptions for getAccessControlList API

Type	Description
`ResourceNotFoundException`	This exception is displayed if the source file or target folder does not exist.
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.

Code Snippet:

The `getAccessControlList` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `getAccessControlList` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")
aclDic = pesImpl. getAccessControlList
(
source = ""/Temp Folder/Temp.txt"
)
    # this will display all the dictionary contents
     print aclDic
```

getAllVersions Method

Retrieves a list of all versions of a file in the Content Repository. The following sections describe Python API usage.

Method Signature:

```
resourceList getAllVersions (source, submittedHierarchy) throws
ResourceNotFoundException, InsufficientParameterException, IllegalParameterException
```

Input Parameters:

Table 6-38 lists the input parameters for the `getAllVersions` API.

Table 6-38
Input parameters for getAllVersions API

Field	Required?	Type	Example Value	Description
source	Yes	String	"/Temp Folder/Temp.txt" or "0a58c3670016a7860000010dcee0eaa28219"	The fully qualified path or Object URI of the file in the Content Repository
submittedHierarchy	No	Boolean	True or False	Indicates whether the file is in the Submitted Jobs folder

Information Returned:

Table 6-39 lists the information returned by the getAllVersions API.

Table 6-39
Information returned by getAllVersions API

Type	Description
resourceList	A list of resource objects. See Resource on p. 122.

Exceptions:

Table 6-40 lists possible exceptions returned by the getAllVersions API.

Table 6-40
Possible exceptions for getAllVersions API

Type	Description
ResourceNotFoundException	This exception is displayed if the source file does not exist.
InsufficientParameterException	This exception is displayed if any of the required parameters are not specified.
IllegalParameterException	This exception is displayed if the specified source is a folder.

Code Snippet:

The getAllVersions API creates an object of class PESImpl by passing the user name, password, host, and port. The getAllVersions API can be called on the instance of the PESImpl object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")
# here the source value is a fully qualified path
resourceList = pesImpl.getAllVersions(source="/Temp Folder/Temp.txt")

# following code snippet iterates over the list of
# resources and prints author, title and resourceID

For resource in resourceList:
        print resource.getAuthor()
print resource.getTitle()
print resource.getResourceID()


# alternatively here the source value is a ResourceID
resourceList = pesImpl.getAllVersions(source="0a58c3670016a7860000010dcee0eaa28219")
```

```
# following code snippet iterates over the list of
# resources and prints author, title and resourceID

For resource in resourceList:
        print resource.getAuthor()
        print resource.getTitle()
        print resource.getResourceID()
```

getChildren Method

Retrieves a list of all files and folders within a specified Content Repository folder. The following sections describe Python API usage.

Method Signature:

```
ResourceList getChildren (source, submittedHierarchy) throws ResourceNotFoundException,
InsufficientParameterException
```

Input Parameters:

Table 6-41 lists the input parameters for the `getChildren` API.

Table 6-41
Input parameters for getChildren API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder"` or `"0a58c3670016a7860f00010dcee0eaa28219"`	The fully qualified path or Object URI of the folder in the Content Repository
`submittedHierarchy`	No	Boolean	`True` or `False`	Indicates whether the folder is in the Submitted Jobs folder

Information Returned:

Table 6-42 lists the information returned by the `getChildren` API.

Table 6-42
Information returned by getChildren API

Type	Description
`ResourceList`	See Resource on p. 122.

Exceptions:

Table 6-43 lists possible exceptions returned by the `getChildren` API.

Table 6-43
Possible exceptions for getChildren API

Type	Description
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.
`ResourceNotFoundException`	This exception is displayed if the folder does not exist.

Code Snippet:

The `getChildren` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `getChildren` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")
# here the source value is a fully qualified path
resourceList = pesImpl.getChildren(source="/Temp Folder")

# alternatively here the source value is a ResourceID
resourceList = pesImpl.getChildren(source="0a58c3670016a7860000010dcee0eaa28219")

# following code snippet iterates over the list of
# resources and prints author, title and resourceID
for resource in resourceList:
        print resource.getAuthor()
print resource.getTitle()
print resource.getResourceID()
```

getCustomPropertyValue Method

Retrieves the valid values accepted by a specified custom property. The following sections describe Python API usage.

Method Signature:

```
list getCustomPropertyValue(propertyName) throws ResourceNotFoundException,
InsufficientParameterException
```

Input Parameters:

Table 6-44 lists the input parameters for the `getCustomPropertyValue` API.

Table 6-44
Input parameters for getCustomPropertyValue API

Field	Required?	Type	Example Value	Description
`propertyName`	Yes	String	`"FreeForm"`	The name of the custom property

Information Returned:

Table 6-45 lists the information returned by the `getCustomPropertyValue` API.

Table 6-45
Information returned by getCustomPropertyValue API

Type	Description
list	Returns a list of valid values the custom property accepts. If the property requires a selection (for example, single select or multi-select), the list contains all valid values for the selection. If it is a free-form property, the list contains the type of data the property accepts (for example, `String`, `Date`, or `Number`).

Exceptions:

Table 6-46 lists possible exceptions returned by the `getCustomPropertyValue` API.

Table 6-46
Possible exceptions for getCustomPropertyValue API

Type	Description
`ResourceNotFoundException`	This exception is displayed if the source file or target folder does not exist.
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.

Code Snippet:

The `getCustomPropertyValue` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `getCustomPropertyValue` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")
list = pesImpl.getCustomPropertyValue
(
propertyName = "FreeForm"
)
```

getMetadata Method

Retrieves the metadata attributes of a file or folder in the Content Repository, including any custom properties and topic information. The following sections describe Python API usage.

Method Signature:

```
Resource getMetadata (source, version, label, submittedHierarchy) throws
ResourceNotFoundException, InsufficientParameterException
```

Input Parameters:

Table 6-47 lists the input parameters for the `getMetadata` API.

Table 6-47
Input parameters for getMetadata API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000010dcee0eaa28219"`	The fully qualified path or Object URI of the file or folder in the Content Repository
`version`	No. Either version or label can be specified.	String	`"0:2006-08-25 21:15:49.453"`	The specific version of the file or folder

Field	Required?	Type	Example Value	Description
label	No. Either version or label can be specified.	String	"Version 1"	The specific labeled version of the file or folder
submittedHierarchy	No	Boolean	True or False	Indicates whether the file is in the Submitted Jobs folder

Information Returned:

Table 6-48 lists the information returned by the getMetadata API.

Table 6-48
Information returned by getMetadata API

Type	Description
Resource	See Resource on p. 122.

Exceptions:

Table 6-49 lists possible exceptions returned by the getMetadata API.

Table 6-49
Possible exceptions for getMetadata API

Type	Description
ResourceNotFoundException	This exception is displayed if the source file or folder does not exist.
InsufficientParameterException	This exception is displayed if any of the required parameters are not specified.

Code Snippet:

The getMetadata API creates an object of class PESImpl by passing the user name, password, host, and port. The getMetadata API can be called on the instance of the PESImpl object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

# This will retrieve the metadata of the latest
# version of the specified.

resource = pesImpl.getMetadata(source="/Temp Folder/Temp.txt")
# Following snippet prints Author, Title and
# ResourceID
 print resource.getAuthor()
 print resource.getTitle()
 print resource.getResourceID()

 # following statement will print all associated custom
 # metadata
 print resource.getCustomMetadata()

# following statement prints all the topics defined on
# resource
 print resource.getTopicList()

# Here we specify version
```

```
resource = pesImpl.getMetadata(source="/Temp Folder/Temp.txt",version="1:2006-08-25
21:15:49.453")

# We can also specify label to get metadata of a file

Resource = pesImpl.getMetadata(source="/Temp Folder/Temp.txt",label="label 1")
```

importResource Method

Imports an existing **.pes* export file from the local file system to the Content Repository. The following sections describe Python API usage.

Method Signature:

```
Boolean importResource (source, target) throws ResourceNotFoundException,
InsufficientParameterException
```

Input Parameters:

Table 6-50 lists the input parameters for the `importResource` API.

Table 6-50
Input parameters for importResource API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"C:\Temp\New.pes"`	The fully qualified path (on the local file system) of the **.pes* file to import
`target`	Yes	String	`"/Temp Folder"` or `"0a58c3670016a7860000 010dcee0eaa28219"`	The fully qualified Content Repository path or Object URI of the folder to import to

Information Returned:

Table 6-51 lists the information returned by the `importResource` API.

Table 6-51
Information returned by importResource API

Type	Description
`Boolean`	`True` or `False` based on whether the API runs successfully.

Exceptions:

Table 6-52 lists possible exceptions returned by the `importResource` API.

Table 6-52
Possible exceptions for importResource API

Type	Description
`ResourceNotFoundException`	This exception is displayed if the source file or target folder does not exist.
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.

Code Snippet:

The `importResource` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `importResource` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

bSuccess = pesImpl.importResource
(
source="C:\Temp\New.pes",
target="/Temp Folder"
)
```

moveResource Method

Moves a file or folder to another folder in the Content Repository. The specified file/folder can be renamed when it is moved. The following cases describe the behavior of the renaming feature:

If the source is */Temp Folder/Temp.txt* and the target is */Demo Folder*:

- **Case 1**: If folder *Demo Folder* exists, *Temp.txt* is moved to *Demo Folder*.
- **Case 2**: If folder *Demo Folder* does not exist, *Temp.txt* is moved to " / " and renamed to *Demo Folder*.

If the source is */Temp Folder/Temp.txt* and the target is */Demo Folder/Abc.dat*:

- **Case 1**: If folder *Demo Folder* exists, *Temp.txt* is moved to *Demo Folder* and renamed to *Abc.dat*.
- **Case 2**: If folder *Demo Folder* does not exist, an error is displayed.

The following sections describe Python API usage.

Method Signature:

```
Boolean moveResource (source, target) throws
ResourceNotFoundException, InsufficientParameterException
```

Input Parameters:

Table 6-53 lists the input parameters for the `moveResource` API.

Table 6-53
Input parameters for moveResource API

Field	Required?	Type	Example Value	Description
source	Yes	String	"/Temp Folder/Temp.txt" or "0a58c3670016a7860000010dcee0eaa28219"	The fully qualified path or Object URI of the file or folder in the Content Repository
target	Yes	String	"/New Folder" or "/New Folder/abc.dat"	The fully qualified path or Object URI of the folder to move the file to. A new file name can also be provided for renaming the specified file/folder when it is moved.

Information Returned:

Table 6-54 lists the information returned by the moveResource API.

Table 6-54
Information returned by moveResource API

Type	Description
Boolean	True or False based on whether the API runs successfully.

Exceptions:

Table 6-55 lists possible exceptions returned by the moveResource API.

Table 6-55
Possible exceptions for move Resource API

Type	Description
ResourceNotFoundException	This exception is displayed if the source file or target folder does not exist.
InsufficientParameterException	This exception is displayed if any of the required parameters are not specified.

Code Snippet:

The moveResource API creates an object of class PESImpl by passing the user name, password, host, and port. The moveResource API can be called on the instance of the PESImpl object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

# here moving a File to the specified location
bSuccess = pesImpl.moveResource
(
source="/Temp Folder/Temp.txt",
target="/New Folder"
)

# here moving a Folder to the specified location
bSuccess = pesImpl.moveResource
(
source="/Temp Folder",
```

```
target="/New Folder"
)
```

removeLabel Method

Removes a label from a file in the Content Repository. The following sections describe Python API usage.

Method Signature:

```
uri removeLabel (source, label) throws ResourceNotFoundException,
InsufficientParameterException
```

Input Parameters:

Table 6-56 lists the input parameters for the `removeLabel` API.

Table 6-56
Input parameters for removeLabel API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000 010dcee0eaa28219"`	The fully qualified path or Object URI of the file in the Content Repository
`label`	Yes	String	`"Version 1"`	The label name to remove

Information Returned:

Table 6-57 lists the information returned by the `removeLabel` API.

Table 6-57
Information returned by removeLabel API

Type	Description
`uri`	URI of the updated file

Exceptions:

Table 6-58 lists possible exceptions returned by the `removeLabel` API.

Table 6-58
Possible exceptions for removeLabel API

Type	Description
`ResourceNotFoundException`	This exception is displayed if the source file does not exist.
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.

Code Snippet:

The `removeLabel` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `removeLabel` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

# here label "Version 1" will be removed from "/Temp
# Folder/Temp.txt" file

uri = pesImpl.removeLabel(source="/Temp Folder/Temp.txt", label=
" Version 1")
# following code snippet prints the uri.
print uri
```

removeSecurity Method

Removes the security ACL from a specified file or folder in the Content Repository. The following sections describe Python API usage.

Method Signature:

```
Boolean removeSecurity (source, principal, provider, cascade) throws
ResourceNotFoundException, InsufficientParameterException, IllegalParameterException
```

Input Parameters:

Table 6-59 lists the input parameters for the `removeSecurity` API.

Table 6-59
Input parameters for removeSecurity API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000010dcee0eaa28219"`	The fully qualified path or Object URI of the file or folder in the Content Repository
`principal`	Yes	String	`admin`	The user (such as *admin*) to remove from the specified file/folder
`provider`	No	String	`Native`	The security provider (such as *Native*) to use for obtaining the information about users
`cascade`	No	Boolean	`True` or `False`	Propagates the security settings to all files and subfolders within the specified folder

Information Returned:

Table 6-60 lists the information returned by the `removeSecurity` API.

Table 6-60
Information returned by removeSecurity API

Type	Description
`Boolean`	`True` or `False` based on whether the API runs successfully.

Exceptions:

Table 6-61 lists possible exceptions returned by the `removeSecurity` API.

Table 6-61
Possible exceptions for removeSecurity API

Type	Description
`ResourceNotFoundException`	This exception is displayed if the source file or target folder does not exist.
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.
`IllegalParameterException`	This exception is displayed if the specified user or security provider name is incorrect.

Code Snippet:

The `removeSecurity` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `removeSecurity` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")
bSuccess = pesImpl.removeSecurity
(
source="/Temp Folder/Temp.txt",
principal="Joe",
provider="Native"
)
# One can also specify the cascade flag to cascade the
# security permissions. By default cascade flag is
# false
bSuccess = pesImpl.removeSecurity
(
source="/Temp Folder/Temp.txt",
principal="Joe",
provider="Native",
cascade="True"
)
```

search Method

Searches for files in the Content Repository and displays a list of files that match the search criteria, and their versions. The following sections describe Python API usage.

Method Signature:

```
PageResult search (criteria) throws InsufficientParameterException
```

Input Parameters:

Table 6-62 lists the input parameters for the `search` API.

Table 6-62
Input parameters for search API

Field	Required?	Type	Example Value	Description
`criteria`	Yes	String	`"Age"`	The value used to search file metadata

Information Returned:

Table 6-63 lists the information returned by the `search` API.

Table 6-63
Information returned by search API

Type	Description
`PageResult`	Returns the list of files based on the search criteria used. See PageResult on p. 123..

Exceptions:

Table 6-64 lists possible exceptions returned by the `search` API.

Table 6-64
Possible exceptions for search API

Type	Description
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.

Code Snippet:

The `search` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `search` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

# here the files will be looked for the criteria "Age"
pageResult = pesImpl.search(criteria="Age")

# following code snippet iterates over the list of
# resources and prints author, title and resourceID

rows = pageResultList.getRows()
for row in rows:
        print row.getAuthor()
        print row.getTitle()
        print row.getResourceID()
```

setLabel Method

Applies a label to a version of a file in the Content Repository. If the file is already labeled, the original label is replaced with the new label. The following sections describe Python API usage.

Method Signature:

```
uri setLabel (source, version, label) throws
ResourceNotFoundException, InsufficientParameterException
```

Input Parameters:

Table 6-65 lists the input parameters for the `setLabel` API.

Table 6-65
Input Parameters for setLabel API

Field	Required?	Type	Example Value	Description
source	Yes	String	"/Temp Folder/Temp.txt" or "0a58c3670016a7860000 010dcee0eaa28219"	The fully qualified path or Object URI of the file in the Content Repository
version	Yes	String	"0:2006-08-25 21:15:49.453"	The specific version of the file
label	Yes	String	"Version 1"	The label to apply to the file

Information Returned:

Table 6-66 lists the information returned by the `setLabel` API.

Table 6-66
Information returned by setLabel API

Type	Description
uri	URI of the updated file

Exceptions:

Table 6-67 lists possible exceptions returned by the `setLabel` API.

Table 6-67
Possible exceptions for setLabel API

Type	Description
ResourceNotFoundException	This exception is displayed if the source file or version does not exist.
InsufficientParameterException	This exception is displayed if any of the required parameters are not specified.

Code Snippet:

The `setLabel` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `setLabel` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

# here label "Version 1" will be set to "/Temp
# Folder/Temp.txt" file

uri = pesImpl.setLabel(source="/Temp Folder/Temp.txt", version=
"1:2006-08-25 21:15:49.453", label=" Version 1")

# following code snippet prints the uri.
print uri
```

setMetadata Method

Applies metadata properties to files and folders in the Content Repository. Table 6-68 lists the metadata properties and whether they can be applied to files and/or folders.

Table 6-68
Keywords for Content Repository APIs

Metadata Property	Resource Type
Author	File
Description	File/Folder
Title	File/Folder
Expiration Date	File/Folder
Keyword	File
Topics	File
Custom Metadata	File/Folder

The following sections describe Python API usage.

Method Signature:

```
uri setMetadata (source, version, label, props) throws
ResourceNotFoundException, InsufficientParameterException
```

Input Parameters:

Table 6-69 lists the input parameters for the `setMetadata` API.

Table 6-69
Input parameters for setMetadata API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000 010dcee0eaa28219"`	The fully qualified path or Object URI of the file/folder in the Content Repository
`version`	No. Either version or label can be specified.	String	`"0:2006-08-25 21:15:49.453"`	The specific version of the file to be downloaded
`label`	No. Either version or label can be specified.	String	`"Label 1"`	The label of the specific version
`props`	Yes	Dictionary	`{ 'author':'admin', 'title':'newTitle', 'description','desc', 'topic':[a,b], 'customProperty': { 'language':'hindi|english', 'FreeForm': 'abcd' } }`	Contains all the metadata to be set, in the Dictionary with the metadata name as keys. As shown in the Example Value column, it takes the list as a value from `topic` and Dictionary for `customProperty`. For the rest of the metadata it takes string.

Information Returned:

Table 6-70 lists the information returned by the `setMetadata` API.

Table 6-70
Information returned by setMetadata API

Type	Description
`uri`	URI of the file/folder for which metadata was set

Exceptions:

Table 6-71 lists possible exceptions returned by the `setMetadata` API.

Table 6-71
Possible exceptions for setMetadata API

Type	Description
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.
`ResourceNotFoundException`	This exception is displayed if the source file/folder does not exist.

Code Snippet:

The `setMetadata` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `setMetadata` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

# dictionary containing customProperty values.
customPropertyDic = {'freeform':'demo', 'complexity:'medium',
                'language':'hindi|english'
                }
# dictionary containing all the metadata.
propertyDic = {'author':"authorName", 'title':"title",
                  'description':"description",
               'keyword':"keyword",
                  'topic':['topic 1','topic 2'],
                       'customProperty':customProperyDic
                     }


uri = pesImpl.setMetadata
(
source="/Temp Folder/Temp.txt",
version="0:2006-08-25:15:49.453",
props=propertyDic
)
     # you can use label instead of version
uri = pesImpl.setMetadata
(
source="/Temp Folder/Temp.txt",
                          label="label 1",
props=propertyDic
)

# following code snippet prints the uri
print uri
```

uploadFile Method

Saves a file to the Content Repository from the local file system, with the option of creating a new version of the file if it already exists. The following sections describe Python API usage.

Method Signature:

```
IdentificationSpecifier uploadFile (source, target, versionFlag) throws
ResourceNotFoundException, ResourceAlreadyExistsException,
InsufficientParameterException
```

Input Parameters:

Table 6-72 lists input parameters for the `uploadFile` API.

Table 6-72
Input parameters for uploadFile API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"C:\Temp\Temp.txt"`	The fully qualified path (on the local file system) of the file to upload
`target`	Yes	String	`"/Temp Folder`	The fully qualified path of the destination folder in the Content Repository where the file is to be uploaded
`versionFlag`	No	Boolean	`True` or `False`	If the specified file already exists, a new version of the file is created

Information Returned:

Table 6-73 lists the information returned by the `uploadFile` API.

Table 6-73
Information returned by uploadFile API

Type	Description
`uri`	URI of the uploaded file

Exceptions:

Table 6-74 lists possible exceptions returned by the `uploadFile` API.

Table 6-74
Possible exceptions for uploadFile API

Type	Description
`ResourceNotFoundException`	This exception is displayed if the source file or target folder does not exist.
`ResourceAlreadyExistsException`	This exception is displayed if a file/folder with the same name as the source file exists in the target folder and the `createVersion` parameter is not specified.
`InsufficientParameterException`	This exception is displayed if any of the required parameters are not specified.

Code Snippet:

The `uploadFile` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `uploadFile` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

uri = pesImpl.uploadFile
(
source="C:\Temp\Temp.txt",
target="/Temp Folder",
versionFlag=True
)
print uri
```

Wrapper Classes

The classes in this section are wrappers for objects returned from the PASW Collaboration and Deployment Services web services. The wrappers provide an easier interface for displaying the data.

Resource

The `Resource` class acts as a simplified wrapper to the Content Repository object `ResourceSpecifer.Resource`, which is returned through various API calls. This allows users to retrieve object-specific data through an easier interface. Along with metadata information, this class captures any custom metadata information associated with the specified object in the Content Repository. Table 6-75 lists all methods available in the `Resource` class.

Table 6-75
Resource class methods

Method Name	Description
`getAccessControlList`	Returns a dictionary of an object's security permissions. It contains the user name as a key and only the highest permission given to the user. For example: If user *Joe* has *delete* permission on *resource X*, then `getAccessControlList` of the resource object representing *X* will return `{'Joe':'DELETE'}` and not all three permissions (read, write, delete) from the web service call.
`getOwner`	Returns the name of the *owner* of the object as a string
`getAuthor`	Returns the name of the *author* of the object as a string
`getContentSize`	Returns the *size* of the object
`getCreatedBy`	Returns the *name of the user* who created the object as a string
`getCreationDate`	Returns the *creation date* of the object as a datetime object
`getDescription`	Returns the *description* of the object as a list
`getDescriptionLanguage`	Returns the *language* of the object as a list
`getExpirationDate`	Returns the *expiration date* of the object as a datetime object
`isExpired`	Indicates whether the specified object has expired or not
`getMimeType`	Returns the *file type* of the object as a string
`getModificationDate`	Returns the *last modified date* of the object as a datetime object
`getObjectCreationDate`	Returns the *object creation date* of the object as a datetime object

Method Name	Description
`getObjectLastModifiedBy`	Returns the *user* who last modified the object as a string
`getObjectLastModifiedDate`	Returns the *object last modified date* of the object as a datetime object
`getResourceID`	Returns the *Object URI* of the object as a string
`getResourcePath`	Returns the *path* of the specified object as a string
`getTitle`	Returns the *Title* for the object as a string
`getTopicList`	Returns a *list of Topics* for the object
`getVersionMarker`	Returns the *version* of the object as a string.
`getVersionLabel`	Returns the *label* of the object as a string
`getCustomMetadata`	Returns any *custom properties* associated with the object as a dictionary
`getKeywordList`	Returns a list of *keywords* associated with the object

IdentificationSpecifier

This class acts as a simplified wrapper to the Content Repository object `IdentificationSpecifier`, which is returned through various API calls. This allows users to retrieve identification-specific data through an easier interface. Table 6-76 lists all methods available in the `IdentificationSpecifier` class.

Table 6-76
IdentificationSpecifier class methods

Method Name	Description
`getIdentifier`	Returns the *identifier value* of an object as a string
`getVersionMarker`	Returns the *version* of an object as a string
`getVersionLabel`	Returns the *label* applied to an object as a string

PageResult

This class is used as a simplified wrapper to the `PageResult` object, which is returned from the `queryExecution` API. This allows users to retrieve data specific to a job run through an easier interface. Table 6-77 lists all methods available in the `PageResult` class.

Table 6-77
PageResult class methods

Method Name	Description
`getRows`	Returns a *list of row objects*, which is a wrapper around the Process Management `Row` object

SearchResult

This class acts as a simplified wrapper to the `Row` object in `PageResult`, which is returned from the Search web service. The `SearchResult` class differs from the `Row` wrapper class, though the purpose of both classes is the same. Table 6-78 lists all methods available in the `SearchResult` class.

Table 6-78
SearchResult class methods

Method Name	Description
`getTitle`	Returns the *name* of the file/folder
`getAuthor`	Returns the *author* of the file/folder
`getMimeType`	Returns the *mime type* of the file
`getObjectLastModifiedBy`	Returns the *user* who last modified the file/folder
`getModified`	Returns the *date and time* the file/folder was last modified
`getFolderPath`	Returns the *location* of the file/folder
`getFolder`	Returns the *name* of parent folder of the file/folder
`getParentURI`	Returns the *Object URI* of the parent
`getTopic`	Returns the *topics* associated with the file/folder
`getChildRow`	Returns the *list* of `SearchChildRow` objects (see the following section for more information)

SearchChildRow

This class acts as a simplified wrapper to the `ChildRow` object. It provides simplified access to various fields in the `ChildRow` object. Table 6-79 lists all methods available in the `SearchChildRow` class.

Table 6-79
SearchChildRow class methods

Method Name	Description
`getExpirationDate`	Returns the *expiration date* of the file/folder
`getKeyword`	Returns the *keywords* associated with the version of the file/folder
`getVersionLabel`	Returns the *version label* of the file/folder
`getDescription`	Returns the *description* of the file/folder
`getLanguage`	Returns the *language*
`getVersionCreationDate`	Returns *date and time* the file/folder was created
`getVersionMarker`	Returns the *version marker* of the file/folder
`getUri`	Returns the *Object URI* of the file/folder

Process Management APIs

This section outlines the `PESImpl` functions used for working with jobs stored in the repository. Every function contains detailed syntax information, an example, and expected messages.

APIs

The following sections list all Process Management scripting APIs supported for PASW Collaboration and Deployment Services.

Note: For all APIs described in this guide that require a path to files/folders in the Content Repository, either the path or the Object URI can be used. The Object URI can be obtained by viewing the object properties in Deployment Manager.

cancelJob Method

Cancels a running job. The following sections describe Python API usage.

Method Signature:

```
bSuccess =  cancelJob (executionId) throws
RemoteException, DatabaseException, SchedulingException, AuthorizationException
```

Input Parameters:

Table 6-80 lists the input parameters for the `cancelJob` API.

Table 6-80
Input parameters for cancelJob API

Field	Required?	Type	Example Value	Description
`executionId`	Yes	String	`0a58c33d002ce90800 00010e0ccf7b01800e`	Execution ID for the job

Information Returned:

Table 6-81 lists the information returned by the `cancelJob` API.

Table 6-81
Information returned by cancelJob API

Type	Description
`bSuccess`	Returns a message when the job is cancelled

Code Snippet:

The `cancelJob` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `cancelJob` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

bSuccess = pesImpl.cancelJob(executionId='0a58c33d002ce9080000010e0ccf7b01800e')
```

executeJob Method

Runs an Deployment Manager job synchronously or asynchronously based on the parameters passed. In the case of a synchronous run, the API does not return until the job completes. In the case of an asynchronous run, the API returns after the job starts. The following sections describe Python API usage.

Method Signature:

```
executionID  executeJob(source,notification,asynchronous) throws RemoteException,
SchedulingException
```

Input Parameters:

Table 6-82 lists the input parameters for the `executeJob` API.

Table 6-82
Input parameters for executeJob API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"C:\Temp\Temp.txt"`	The fully qualified path (on the local file system) of the file to upload
`notification`	No	Boolean	`True` or `False`	Indicates whether the job runs with or without notifications. Default is `False`.
`asynchronous`	No	Boolean	`True` or `False`	Indicates whether the job runs asynchronously. Default is `False`.

Information Returned:

Table 6-83 lists the information returned by the `executeJob` API.

Table 6-83
Information returned by executeJob API

Type	Description
`executionID`	String containing the execution ID (for example, `0a58c33d002ce9080000010daaf94ae88100`)

Code Snippet:

The `executeJob` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `executeJob` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

# To run the job asynchronously and without notification.
executionId =  pmImpl.executeJob(source='/Test', asynchronous=True)

# To run the job synchronously and without notification.
    executionId =  pmImpl.executeJob(source='/Test')

# To run the job asynchronously and with notifications.
executionId =  pmImpl.executeJob(source='/Test', notification = True, asynchronous=True)
```

getJobExecutionDetails Method

Lists the run details for a specific job, including any job steps and iterations. The following sections describe Python API usage.

Method Signature:

```
executionDetailsDict getJobExecutionDetails(executionId,log,target) throws RemoteException,
ObjectNotFoundException, DatabaseException, SchedulingException, AuthorizationException
```

Input Parameters:

Table 6-84 lists the input parameters for the `getJobExecutionDetails` API.

Table 6-84
Input parameters for getJobExecutionDetails API

Field	Required?	Type	Example Value	Description
`execu-tionId`	Yes	String	`0a58c33d002ce9080000 010e0ccf7b01800e`	The execution Id of the job
`log`	No	Boolean	`True` or `False`	Indicates whether the job log is displayed inline
`target`	No	String	`"c:\logs"`	The location (on the local file system) to store the logs. Only used in conjunction with the `--log` parameter.

Information Returned:

Table 6-85 lists the information returned by the `getJobExecutionDetails` API.

Table 6-85
Information returned by getJobExecutionDetails API

Type	Description

`executionDetailsDict`	Dictionary containing a list of run details for the job, job step, and job step iterations. The dictionary is structured as follows:
	
	`{`
	`"job":[executionDetails], "jobStep":[executionDetails],`
	`"jobStepIteration":[executionDetails]`
	`}`
	`The executionDetails object contains details like eventUuid, eventName, executionSuccess, executionState, startDateTime, endDateTime,log, artifactLocation etc.`
	`Methods that can be used to access above information is get_attribute_eventUuid(), get_attribute_eventName(),get_attribute_executionSuccess(), get_attribute_executionState(),get_attribute_startDateTime(), get_attribute_endDateTime(),get_element_log(), get_element_artifactLocation()`
	
	Methods that can be used to access the above information are:
	
	`get_attribute_eventUuid(),get_attribute_eventName(), get_attribute_executionSuccess(), get_attribute_execution State(), get_attribute_startDateTime(),get_attribute_endD ateTime(), get_element_log(),get_element_artifactLocatio n()`

Code Snippet:

The `getJobExecutionDetails` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `getJobExecutionDetails` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl

pesImpl = PESImpl("admin", "spss", "localhost", "8080")

# Note that even though log is false, we do get log
# details if any in execution Details object.
executionDetailsDict = pesImpl. getJobExecutionDetails
(executionId=" 0a58c35a50bd8291000001144ef7935b8088")

# stores the logs generated at the target location.
executionDetailsDict = pesImpl. getJobExecutionDetails (executionId="
0a58c35a50bd8291000001144ef7935b8088",log=True,
target="c:\logs")
```

getJobExecutionList Method

Lists the runs for a specific job, including any currently running jobs and completed jobs, for all versions of the job. The following sections describe Python API usage.

Method Signature:

```
PageResult getJobExecutionList(source)
```

Input Parameters:

Table 6-86 lists the input parameters for the `getJobExecutionList` API.

Table 6-86
Input parameters for getJobExecutionList API

Field	Required?	Type	Example Value	Description
`source`	Yes	String	`"/testJob"`	The fully qualified path of the job in the Content Repository.

Information Returned:

Table 6-87 lists the information returned by the `getJobExecutionList` API.

Table 6-87
Information returned by getJobExecutionList API

Type	Description
`PageResult`	See PageResult on p. 123..

Code Snippet:

The `getJobExecutionList` API creates an object of class `PESImpl` by passing the user name, password, host, and port. The `getJobExecutionList` API can be called on the instance of the `PESImpl` object.

```
from pes.api.PESImpl import PESImpl
```

```
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
pageResult = pesImpl.getJobExecutionList("\testJob")
rows = pageResult.getRows()
if rows:
    for row in rows:
        print row.getPath()
        print row.getObjId()
        print row.getEventObjId()
        print row.getVersionMarker()
        print row.getEventStartDateTime()
        print row.getEventEndDateTime()
```

Wrapper Classes

The classes in this section are wrappers for objects returned from the PASW Collaboration and Deployment Services web services. The wrappers provide an easier interface for displaying the data.

PageResult

This class acts as a simplified wrapper to the Process Management object `PageResult`, which is returned from the `queryExecution` API. This allows users to retrieve job execution specific data through an easier interface. Table 6-88 lists all methods available in the `PageResult` class.

Table 6-88
PageResult class methods

Method Name	Description
`getRows`	Returns a list of the row object, which is a wrapper around the Process Management `Row` object

Row

This class acts as a simplified wrapper to the Process Management object `Row`, which is returned from the `queryExecution` API. Table 6-89 lists all methods available in the `Row` class.

Table 6-89
Row class methods

Method Name	Description
`getObjId`	Returns the execution ID of the job
`getPath`	Returns the path of the job
`getVersionMarker`	Returns the version marker of the job that was run
`getVersionLabel`	Returns the version label of the job that was run
`getEventObjId`	Returns the event ID of the job that was run
`getEventState`	Returns the state of the running job
`getEventCompletionCode`	Returns the completion code of the job
`getEventStartDateTime`	Returns the start date and time of the job
`getEventEndDateTime`	Returns the end date and time of the job
`getQueuedDateTime`	Returns the queued date and time of the job

jobExecutionDetails

This class is returned from the `getJobExecutionDetails` API. It stores the run details for a job and stores a list of `jobStepExecution` objects. This class contains the `ExecutionDetails` object, to which it delegates all of its method calls. Table 6-90 lists all methods available in the `jobExecutionDetails` class.

Table 6-90
jobExecutionDetails class methods

Method Name	Description
`getJobStepDetails`	Returns a list of `jobStepExecutionDetails` objects
`getArtifactLocation`	Returns a list of job artifact locations
`getCompletionCode`	Returns the completion code of the job
`getEndDateTime`	Returns the end date and time of the job
`getEventName`	Returns the event name of the job
`getEventUUID`	Returns the event ID of the job
`getExecutionState`	Returns the run state of the job
`getExecutionSuccess`	Returns success or failure status of the job
`getExecutionWarning`	Indicates whether there were any warnings
`getLog`	Returns the log (as string) generated
`getNotificationEnabled`	Indicates whether e-mail notifications are enabled or not
`getQueuedDateTime`	Returns the queued date and time of the job
`getStartDateTime`	Returns the start date and time of the job
`getUserName`	Returns the name of the user who ran the job
`getUUID`	Returns the execution ID of the job

jobStepExecutionDetails

This class stores the run details for a job step and stores a list of `jobStepChildExecutionDetails` objects. This class contains the `ExecutionDetails` object, to which it delegates all of its method calls.Table 6-91 lists all methods available in the `jobStepExecutionDetails` class.

Table 6-91
jobStepExecutionDetails class methods

Method Name	Description
`getJobStepChldExecution-List`	Returns a list of `jobStepChildExecutionDetails` objects
`getArtifactLocation`	Returns a list of job artifact locations
`getCompletionCode`	Returns the completion code of the job
`getEndDateTime`	Returns the end date and time of the job
`getEventName`	Returns the event name of the job

Method Name	Description
`getEventUUID`	Returns the event ID of the job
`getExecutionState`	Returns the run state of the job
`getExecutionSuccess`	Returns success or failure status of the job
`getExecutionWarning`	Indicates whether there were any warnings
`getLog`	Returns the log (as string) generated
`getNotificationEnabled`	Indicates whether e-mail notifications are enabled or not
`getQueuedDateTime`	Returns the queued date and time of the job
`getStartDateTime`	Returns the start date and time of the job
`getUserName`	Returns the name of the user who ran the job
`getUUID`	Returns the execution ID of the job

jobStepChildExecutionDetails

This class stores the run details for job step iterations (for iterative jobs). This class contains the `ExecutionDetails` object, to which it delegates all of its method calls.Table 6-92 lists all methods available in the `jobStepChildExecutionDetails` class.

Table 6-92
jobStepChildExecutionDetails class methods

Method Name	Description
`getArtifactLocation`	Returns a list of job artifact locations
`getCompletionCode`	Returns the completion code of the job
`getEndDateTime`	Returns the end date and time of the job
`getEventName`	Returns the event name of the job
`getEventUUID`	Returns the event ID of the job
`getExecutionState`	Returns the run state of the job
`getExecutionSuccess`	Returns success or failure status of the job
`getExecutionWarning`	Indicates whether there were any warnings
`getLog`	Returns the log (as string) generated
`getNotificationEnabled`	Indicates whether e-mail notifications are enabled
`getQueuedDateTime`	Returns the queued date and time of the job
`getStartDateTime`	Returns the start date and time of the job
`getUserName`	Returns the name of the user who ran the job
`getUUID`	Returns the execution ID of the job

Chapter
7

HTML Archive

An HTML report typically involves a number of HTML files displaying a variety of referenced images using style sheets to control the appearance of the output. Due to the number of files involved, managing and sharing this output can be a challenge. If one file is missing or incorrectly referenced, the pages do not display correctly.

The HTML Archive, or HTMLC, format addresses the issue of managing numerous intra-linked files by placing all associated HTML artifacts into a single, cross-browser archive file. The repository includes a viewer enabling a variety of client applications to display the contents of the archive. When accessing an HTMLC file stored in the repository, relative cross-references within the archive are silently replaced with full paths that reference the archive file. This allows links to files within the archive to resolve completely and display correctly.

File Structure

An HTMLC archive file contains:

- a primary HTML file at the root of the archive. When rendering an HTMLC archive, the viewer uses the first file with an *.html* extension at the archive root as the primary file.
- secondary files referenced by the primary file, such as cascading style sheets, images, javascript, or other HTML files. Secondary files can exist in any folder within the archive.

All references to files within the archive should use relative paths.

Creating HTMLC Files

HTMLC files can be created in PASW BIRT Report Designer when working with report designs stored in the repository. However, custom HTMLC files can also be created using a file archiver such as the Java Archive tool or WinZip. To manually create an HTMLC file:

1. Create the structure for the files in the file system.
2. Create an archive containing those files and folders, specifying an extension of *.htmlc* for the output file.

The files in the archive may be created manually or automatically. In PASW Statistics, for example, you can export the results of an analysis as HTML. The resulting HTML and image files can be archived as an HTMLC file. Alternatively, you can use an HTML editor to manually create pages to be archived.

Custom HTMLC File Example

For this example, consider the folder structure shown in the "Archive Files" figure.

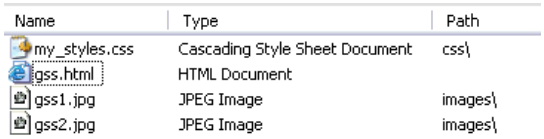
Figure 7-1
Archive Files

The HTML file *gss.html* references images contained in the *images* folder and uses styles contained in a cascading stylesheet in the *css* folder. Using the Java Archive tool, the following command creates an HTMLC file named *custom.HTMLC* containing the files.

```
jar -cvfM custom.HTMLC gss.htm images css
```

The contents of the resulting archive appear in "HTMLC Archive" figure.

Figure 7-2
HTMLC Archive



Storing this single archive in the repository allows the *gss.html* page to be displayed in repository clients, such as the Deployment Portal or Deployment Manager, with its referenced graphics using the defined styles.

Chapter
8

Customization Example

The Model Management page of Deployment Portal provides the ability to monitor the ongoing performance of models deployed to repository as scenario files. These scenarios are associated with PASW Collaboration and Deployment Services jobs that can be executed on demand or scheduled. Scenario files are created with the PASW Modeler application, and they use streams for underlying analytical processing. Model evaluation and champion challenger jobs in PASW Collaboration and Deployment Services are set up and executed using Deployment Manager, and Deployment Portal is used only to view the results. The information displayed as tabs on the Model Management page can include a listing of the best and worst performing models, trends of model performance, champion models, and a listing of all deployed scenarios. The options on the Configuration tab can be used to specify display parameters and show or hide individual tabs.

For information on using the Model Management page, see the Deployment Portal help system.

The user interface mainly consists of a single Java server page (JSP), *MMDMaster.jsp*. The interface components rendered on the page are either reports from PASW BIRT Report Designer or Visualization reports. These reports are rendered using the PASW Tag Library. The page itself is integrated into Deployment Portal using the Tab Extension framework.

PASW Tag Library

The PASW Tag Library provides support for running the PASW BIRT Report Designer and Visualization reports that generate the bulk of the content on the Model Management page. The tag library also supports interactivity between reports, allowing a source report to invoke a target report. The source report passes parameters to the target report for processing.

Report Definitions

The Report definitions used by the Model Management page are stored in the following directory within the repository installation:

<installation-directory>\components\peb-mmd\reports

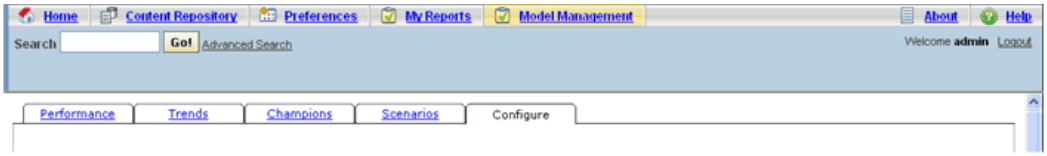
To examine the reports, open the PASW BIRT Report Designer reports in PASW BIRT Report Designer. The visualization reports can be opened using the PASW Viz Designer, or a text or XML editor.

The reports are provided for reference purposes, and should not be directly modified. Any modification of the reports will not be supported by SPSS Inc.. However, you may copy the reports and modify the copies as desired.

Running PASW BIRT Report Designer Reports

The Model Management page involves four master reports that are displayed in four tabs. These tabs are displayed in the "Model Management tabs" figure.

Figure 8-1
Model Management tabs



Each of the tabs corresponds to a master PASW BIRT Report Designer report. When the tab loads, the master PASW BIRT Report Designer report associated with the tab runs using the PASW Tag Library framework.

On the master JavaServer Page, there is one tag for each PASW BIRT Report Designer report. The example below shows the tag used for the Performance Tab.

```
<!–The tag that represents the report –>
<pasw-taglib:repositoryItem
    name="Performance_Tab_Report_Tag"
    inputURI="<<path of the Performance Report>>"
    repositoryCredentialName="localhost"
    activate="ONLOAD"
    location="Performance_Tab_Report_Output"
    outputType="HTML">
  <pasw-taglib:sourceLinkPrompt
      targetNameParameter="LeastPerformingScenarios"
      parameterValue="<<localized text>>" />

<!–
  <<<< Few other parameters which represent localized text>>>>
–>

  <!–This value comes from the User defined prompts –>
  <pasw-taglib:sourceLinkPrompt
      promptId="Performance_Tab_NumberOfPerformers_Prompt"
      targetNameParameter="NumberOfPerformers"/>
  <!–The value of this parameter is specified in the tag itself –>
  <pasw-taglib:sourceLinkPrompt
      targetNameParameter="RunsFromDate"
      parameterValue="<%=scenariosFrom%>"/>
  <pasw-taglib:sourceLinkPrompt
      targetNameParameter="RunsToDate"
      parameterValue="<%=scenariosTo%>"/>
</pasw-taglib:repositoryItem>
```

repositoryItem Tag

The PASW BIRT Report Designer report information is specified in the repositoryItem tag.

```
<pasw-taglib:repositoryItem
    name="Performance_Tab_Report_Tag"
    inputURI="<<path of the Performance Report>>"
    repositoryCredentialName="localhost"
    activate="ONLOAD"
    location="Performance_Tab_Report_Output"
    outputType="HTML">np
```

The repositoryItem tag has following attributes:

- *name*. The repositoryItem tag should be uniquely identified using the *name* attribute. The runRepositoryItem public API uses this name to render the report. For the Performance tab, the name is *Performance_Tab_Report_Tag*.
- *inputURI*. This attribute specifies the location of the PASW BIRT Report Designer report. For the Model Management page, all of the PASW BIRT Report Designer reports are picked up from the server's file system in the *peb-mmd* directory of the repository installation. The URI specified must be a valid URI.
- *activate*. This parameter determines when the report is run. For the Performance tab, the value is *ONLOAD*, indicating the report will run when the page loads. A value of *ONDEMAND* indicates that the user is responsible to initiate running of the report by calling the runReport public API provided by the Reporting tag library. For more information, see the topic Javascript API on p. 138.
- *location*. This specifies the location in which the report is to be rendered. This attribute corresponds to the *id* of the HTML element, which can be either a DIV or an IFRAME. For Model Management, the report location always points to a DIV.
- *outputType*. This specifies the format in which the report is to be rendered using the PASW Tag Library. The output format specified must be one which is supported by the PASW BIRT Report Designer Report Engine. For the PASW BIRT Report Designer reports used in Model Management, the output type is always *HTML*.

sourceLinkPrompt Tag

The sourceLinkPrompt tag specifies the linking of prompts to the report. In other words, this tag specifies how the report acquires the prompt values while running.

There are two ways in which the prompt values are specified for Model Management. The first method is using the *parameterValue* attribute, such as:

```
<pasw-taglib:sourceLinkPrompt
    targetNameParameter="RunsFromDate"
    parameterValue="<%=scenariosFrom%>"/>
```

Here, the name of the prompt is *RunsFromDate*, which is defined in the PASW BIRT Report Designer report. The value for this prompt is specified in the *parameterValue* attribute. The value passed in this attribute is directly passed to the report.

The second method of specifying prompt values is by linking a user prompt to the report parameter. For example:

```
<pasw-taglib:sourceLinkPrompt
    promptId="Performance_Tab_NumberOfPerformers_Prompt"
```

```
      targetNameParameter="NumberOfPerformers"/>

<input type="hidden" id="Performance_Tab_NumberOfPerformers_Prompt"
  name="Performance_Tab_NumberOfPerformers_Prompt"
  value="<%=userProfile.getPerformanceSize()%>"/>
```

Here the *promptId* attribute points to the *id* defined by the hidden HTML input tag. In this case, the value specified in the hidden field *Performance_Tab_NumberOfPerformers_Prompt* would be passed as the prompt value for the report parameter *NumberOfPerformers* while running the report.

Credentials

The reports comprising the Model Management page query the database underlying the repository for their content. Consequently, the reports need a data source corresponding to that database. This data source, *MMDDataSource*, is created within the repository when the user initially loads the Model Management page and is used whenever access to the repository database is required by any of the tags.

To access the *MMDDataSource* data source, the reports must specify valid credentials. The credential tag within the JavaServer pages allows the definition of these credentials.

```
<pasw-taglib:credential
      name="MMDDataSource"
      username="<<some db user name>>"
      password="<< password for the user>>"/>
```

The credentials for this data source are collected via *Login.jsp* before displaying the page and correspond to the username and password for the database underlying the repository. When valid credentials are obtained, the credentials are cached for the duration of the session and are used to run the PASW BIRT Report Designer reports. The Model Management PASW BIRT Report Designer reports are defined such that the name of the datasource is *MMDDataSource*.

In addition to the data source credentials, the Model Management reports also require credentials for the user executing the report.

```
<pasw-taglib:credential
      name="localhost"
      provider="<< some provider id >>"
      username="<< name of some CR user >>"
      password="<<password of the user >>" />
```

These credentials have the name *localhost*. Given that the repository may be configured to allow multiple security providers, the *provider* attribute is required.

The repositoryItem tag requires valid repository user credentials, which are specified in the *repositoryCredentialName* attribute of the tag. For Model Management, the value for this attribute is *localhost*, which corresponds to the user name, provider, and password of the user who has logged into Deployment Portal.

Running Visualization Reports

The methodology to run visualization reports is identical to that used for PASW BIRT Report Designer reports. However, there are a few differences in the usage.

- Visualization reports use a value of *ONDEMAND* for the *activate* attribute of the repositoryItem tag instead of the *ONLOAD* value used by PASW BIRT Report Designer reports.
- Parameters required for the visualization reports are passed by the master PASW BIRT Report Designer reports. For more information, see the topic Visualization Report Interactivity on p. 138.

Javascript API

The tag library has a framework built using JavaScript methods. These JavaScript methods provide both a sound validation framework and a handle to the user to run the reports on demand.

In order to run the reports on demand, the tag library provides a public API. This public API is available in the *reportTagLibPublicAPI.js* file within the *pasw-taglib.war*. The JavaScript file contains the following API:

```
function runRepositoryItem( reportName, linkData, targetId )
```

For Model Management, this function is used to invoke the child reports for the master report.

For example, when the Scenario tab is visible, the Scenario report data is displayed. When the user clicks on the link for a scenario in the master report, the JavaScript method showDetails is called. This JavaScript method is embedded within the PASW BIRT Report Designer report and indirectly calls the runRepositoryItem method to run two reports. One report is the *Scenario details* PASW BIRT Report Designer report and the other is the *Scenario Comparison* visualization report.

If the *linkData* in the API call is null, the report runs with the data available within the JavaServer page supplied using the various PASW Tag Library tags. Just before calling runRepositoryItem, the Javascript code stores the parameter values to the html hidden control. The tag library framework picks up these values and passes them as parameters to the report being run.

The *targetID* fields correspond to the individual DIV ids where the report is to be rendered.

Visualization Report Interactivity

The *Performance vs. Scenario* graph generated by the visualization report for the Champions Tab supports interactivity. Whenever the user clicks on a bar in the graph, the details of the corresponding scenario are displayed in an adjacent area. The reports use the actionHandler and actionParameter tags to achieve this functionality.

Using the actionHandler tag is not necessarily required for visualization reports. Typically, the sourceLinkRepositoryItem tag would work just as well for visualization reports. However, in the case of the Model Management page, the visualization chart can occur multiple times on the page. The application needs special logic to be able to expand detail rows and to run the target reports with specific output locations. The actionHandler tags offer that additional level of control.

The section of the page that renders the *Performance vs. Scenario* Visualization report follows:

```
<pasw-taglib:repositoryItem
name="Champions_Scenario_Index_Report"
  inputURI="ChampionsScenarioIndex.viz"
  repositoryCredentialName="localhost"
  activate="ONDEMAND"
  outputType="HTML"
  location="championsTabVisReport">
  <pasw-taglib:actionHandler event="onclick" function="selectCCScenario">
    <pasw-taglib:actionParameter name="filename" />
    <pasw-taglib:actionParameter name="filepath" />
    <pasw-taglib:actionParameter name="ccid" />
    <pasw-taglib:actionParameter name="equivalencekey" />
  </pasw-taglib:actionHandler>
</pasw-taglib:repositoryItem>
```

The repositoryItem tag gives details about the bar chart to be rendered. The nested actionHandler tag indicates that the JavaScript function selectCCScenario should be called whenever the *onClick* event occurs for the bars. The actionParameter tags nested within the actionHandler tag indicate that *filename*, *filepath*, *ccid*, and *equivalencekey* will be passed to the selectCCScerario function.

Each of these fields is defined within the visualization report XML. The definition for the *filename* variable is shown below:

```
<sourceVariable
categorical="true"
  id="filename"
  source="delimitedFileSource_430"
  sourceName="ct_filename">
```

This tag indicates that the column defined as *ct_filename* within the dataset will be used as *filename* by this report.

The JavaScript function selectCCScerario receives the id of the report on which the event occurred and an array of the parameter values. Internally, it calls runReport for dependent child reports and passes them the value array. For more information, see the topic Javascript API on p. 138.

URL Fragments

The Model Management page displays some repository artifacts in an I-FRAME. These artifacts are the outputs generated by certain job runs.

An artifact is loaded by setting the source of the I-FRAME to the URL having the following format:

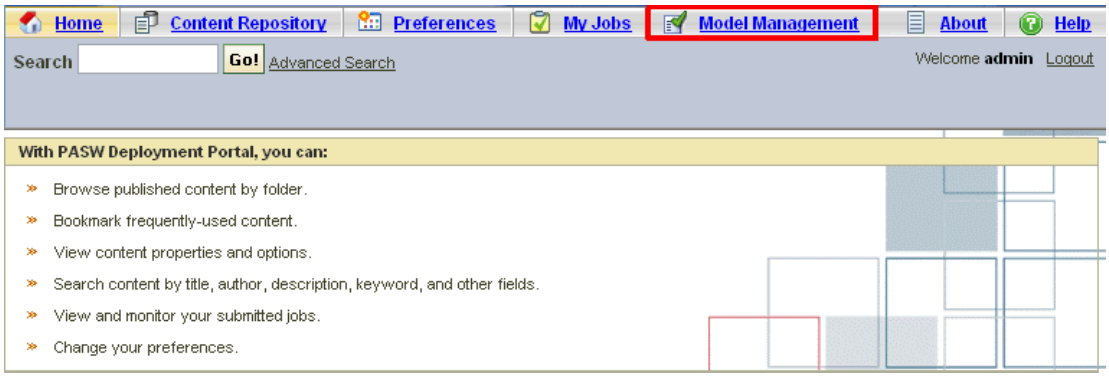
```
http://<servername>:<port>/peb/view?id=<artifact resource id>
```

For more information, see the topic URL Parameters in Chapter 3 on p. 7.

Tab Extension Framework

The navigation tabs of Deployment Portal can be expanded to include custom entries using the Tab Extension framework. The Model Management functionality uses this framework to add an entry point into the Model Management page.

Figure 8-2
Model Management extension



Deployment Portal reads extension files present in the following directory:

<installation-directory>\components\peb\extensions

These files are scanned to find all instances of the *peb-extension* elements. These elements will be individually rendered in the interface, provided the user credentials include any required actions. Any custom application must provide:

- Extension XML file or an entry in an existing extension XML for the application
- Appropriate entries in the localized text (*.tx*) file

The Model Management functionality is contained within the *peb-mmd.package* file in the *staging* directory of the repository installation. The package includes the file *mmd_extension.xml* in the *peb/extensions* directory. This XML file controls the appearance and functionality of the Model Management tab.

```
<file-viewer>
  <peb-extension>
    <tab-id>pebMmdTab</tab-id>
    <tab-key>mmd/pebMmdTabTitle</tab-key>
    <tab-url>
      /peb-mmd/controller?actionName=LoginToMMDAction
    </tab-url>
    <tab-icon>/image2?file=someIcon.gif</tab-icon>
    <tab-position>2</tab-position>
    <tab-security>
      <capability>RunReport</capability>
      <capability>ViewModelManagementDashboard</capability>
    </tab-security>
  </peb-extension>
</file-viewer>
```

Elements defined within this file include:

- *tab-id*. This should be the unique id for the tab. In this case it is *pebMmdTab*.
- *tab-key*. The key references the text appearing on the new tab. For localization purposes, Model Management isolates any localized text in XML files having *.tx* extensions. The key identifies the element in the localization file containing the text to be displayed. In this case, the *mmd/pebMmdTabTitle* key corresponds to the text *Model Management*.
- *tab-url*. This URL is invoked when the user clicks the tab. The URL can be either fully qualified (starting with a slash '/' character) or relative to the Deployment Portal application. In the latter case, the context is assumed to be *peb*. The link must point to a valid URI, with the URI location specified being the responsibility of the custom application. For Model Management, the link includes a reference to the war file *peb-mmd.war*.
- *tab-security*. This tag identifies the actions required to access the tab. If the current user does not have these actions, the tab will not be displayed in the header JSP. Model Management requires the *RunReport* and *ViewModelManagementDashboard* actions.

Index

actionHandler tag, 52, 138
actionParameter tag, 53, 138
actions, 1
activate attribute
  report tag, 136
  repositoryItem tag, 42
advanceSearch, 70, 90
allowDownload parameter
  in URL queries, 24
allowPivoting parameter
  in URL queries, 23
allowPrinterFriendly parameter
  in URL queries, 24
applySecurity, 71, 92
authentication, 5

batch_type parameter
  in URL queries, 18
BIRT reports
  JSP samples, 58

cancelJob, 125
cascadeSecurity, 72, 94
class loader
  for custom applications, 35
  order, 35
  policy, 35
columnName attribute
  sourceLinkVariable tag, 51
contentType attribute
  of page directive, 32
cookies, 66
copyResource, 72, 95
createFolder, 73, 96
creating
  HTMLC files, 133
Credential bean, 54
credential tag, 38, 137
credentialDefinitionName attribute
  credential tag, 40
credentials, 1
custom dialogs, 1, 32
customizing
  Deployment Portal, 3

dataset parameter
  in URL queries, 16
dataset_label parameter
  in URL queries, 17
dataset.prompt parameter
  in URL queries, 19
dataset_rowlimit parameter
  in URL queries, 17
dataset.search.criteria parameter
  in URL queries, 19
dataset.table parameter
  in URL queries, 19
dataset.uri parameter
  in URL queries, 18
dbcredential_datasourcename parameter
  in URL queries, 14
dbpwd_datasourcename parameter
  in URL queries, 14
dbuser_datasourcename parameter
  in URL queries, 14
deleteFile, 73, 97
deleteFileVersion, 74, 98
deleteFolder, 75, 100
Deployment Portal
  customizing, 3
downloadFile, 75, 101

emf files, 12
event attribute
  actionHandler tag, 52
Excel files, 12
executeJob, 87, 126
export, 76
exportResource, 103

format parameter
  in URL queries, 12
fragment parameter
  in URL queries, 13
function attribute
  actionHandler tag, 52

getAccessControlList, 78, 104
getAllVersions, 77, 105
getBookmarkedValues function, 37
getChildren, 78, 107
getCustomPropertyValue, 79, 108
getJobExecutionDetails, 88, 127
getJobExecutionList, 89, 128
getMetadata, 79, 109
getValueJSFunction attribute
  sourceLinkPrompt tag, 49

height attribute
  repositoryItem tag, 44

height parameter
  in URL queries, 15
HTMLC files, 132
  creating, 133
  structure, 132

id parameter
  in URL queries, 8
IdentificationSpecifier class, 123
import, 80
importResource, 111
inputURI attribute
  repositoryItem tag, 42

javascript.name parameter
  in URL queries, 22
javascript.url parameter
  in URL queries, 21
jobExecutionDetails class, 130
jobs, 31
jobStepChildExecutionDetails class, 131
jobStepExecutionDetails class, 130
jpeg files, 12
JSP samples
  accessing, 58
JSR 168, 60

language attribute
  of page directive, 32
linkType attribute
  sourceLinkRepositoryItem tag, 50
localhost credentials, 137
location attribute
  outputLocation tag, 47
  report tag, 136
  repositoryItem tag, 42
Lotus files, 11

MMDDataSource, 137
moveResource, 81, 112

name attribute
  actionParameter tag, 53
  credential tag, 39
  repositoryItem tag, 41, 136

output
  for custom dialogs, 32
  for jobs, 31
  for reports, 30
  for scoring models, 31
output.filename parameter
  in URL queries, 23
output.format parameter
  in URL queries, 22

outputId attribute
  outputLocation tag, 46
outputLocation tag, 46
outputType attribute
  report tag, 136
  repositoryItem tag, 43
outputtype parameter
  in URL queries, 11

page directive, 32
PageResult class, 91, 117, 123, 128–129
parameterName attribute
  repositoryItemPrompt tag, 45
parameterValue attribute
  sourceLinkPrompt tag, 48, 136
partId attribute
  actionHandler tag, 52
  outputLocation tag, 47
partId parameter
  in URL queries, 11
password attribute
  credential tag, 40
password parameter
  in URL queries, 9
PASW Statistics custom dialogs, 1
PASW Statistics Data File Driver Service, 1
PASW Statistics server, 1
PDF files, 12
PevMetaDataBean bean, 56
png files, 12
portal, 60
  single sign-on, 66
portlet, 60
postscript files, 12
PowerPoint files, 12
prefix attribute
  of taglib directive, 32
prepackaged portlets, 60
promptId attribute
  repositoryItemPrompt tag, 45
  sourceLinkPrompt tag, 48, 136
prompts
  for custom dialogs, 32
  for jobs, 31
  for reports, 30
  for scoring models, 31
promptstate parameter
  in URL queries, 10
provider attribute
  credential tag, 40
provider parameter
  in URL queries, 9

removeLabel, 82, 114
removeSecurity, 82, 115
report tag, 46

ReportBean bean, 54
reportDefinitionURI attribute
  report tag, 136
reportPrompt tag, 46
reports, 30
repository items, 30
  custom dialogs, 32
  jobs, 31
  reports, 30
  scoring models, 31
repositoryCredentialName attribute
  repositoryItem tag, 42, 137
repositoryItem tag, 41, 135
repositoryItemName attribute
  repositoryItemPrompt tag, 45
repositoryItemPrompt tag, 45
Resource class, 102, 106–107, 110, 122
retrievePromptValues function, 37
Row class, 129
runRepositoryItem, 138
runRepositoryItem function, 36

Safari browser, 66
scoring models, 31
scoring_configuration parameter
  in URL queries, 17
ScoringBean bean, 57
search, 83, 116
SearchBean bean, 55
SearchChildRow class, 124
SearchResult class, 123
session attribute
  of page directive, 32
setLabel, 84, 117
setMetadata, 84, 118
showLogs parameter
  in URL queries, 24
showNavigationBar attribute
  repositoryItem tag, 44
showOutline parameter
  in URL queries, 23
showTitle attribute
  repositoryItem tag, 43
showToolBar attribute
  repositoryItem tag, 43
single sign-on, 66
sourceLinkPrompt tag, 47, 136
sourceLinkReport tag, 50
sourceLinkRepositoryItem tag, 50
sourceLinkVariable tag, 51
sourceName attribute
  sourceLinkRepositoryItem tag, 50
statistics.server parameter
  in URL queries, 25
statistics.server.credential parameter
  in URL queries, 25
stylesheet.name parameter
  in URL queries, 21
stylesheet.url parameter
  in URL queries, 21

taglib directive, 32
targetNameParameter attribute
  sourceLinkPrompt tag, 48
  sourceLinkVariable tag, 51
title attribute
  repositoryItem tag, 43

uploadFile, 86, 121
uri attribute
  of taglib directive, 32
URL parameters
  example, 139
username attribute
  credential tag, 40
username parameter
  in URL queries, 9
useSSO attribute
  credential tag, 39

validate.method parameter
  in URL queries, 22
validateJSFunction attribute
  sourceLinkPrompt tag, 49
var_variable parameter
  in URL queries, 16
variable parameters
  in URL queries, 16
variable.display parameter
  in URL queries, 20
variable.sort parameter
  in URL queries, 20
version parameter
  in URL queries, 8
visualization reports
  JSP samples, 58
visualiztion reports
  interactivity, 138

waitstate parameter
  in URL queries, 10
war file, 32
Web Part, 60
WebSphere, 35
width attribute
  repositoryItem tag, 44
width parameter
  in URL queries, 15
Word files, 12