IBM SPSS Collaboration and Deployment Services - Essentials for Python 4.2 Reference

IBM®

Preface

IBM® SPSS® Collaboration and Deployment Services is an enterprise-level application that enables widespread use and deployment of predictive analytics. IBM SPSS 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 IBM SPSS 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.

IBM® SPSS® Collaboration and Deployment Services - Essentials for Python provide a set of APIs for programmatic interaction with IBM SPSS Collaboration and Deployment Services. This manual documents the APIs and provides examples of their use for managing the repository, accessing objects, and automating processes.

About IBM Business Analytics

IBM Business Analytics software delivers complete, consistent and accurate information that decision-makers trust to improve business performance. A comprehensive portfolio of business intelligence, predictive analytics, financial performance and strategy management, and analytic applications provides clear, immediate and actionable insights into current performance and the ability to predict future outcomes. Combined with rich industry solutions, proven practices and professional services, organizations of every size can drive the highest productivity, confidently automate decisions and deliver better results.

As part of this portfolio, IBM SPSS Predictive Analytics software helps organizations predict future events and proactively act upon that insight to drive better business outcomes. Commercial, government and academic customers worldwide rely on IBM SPSS technology as a competitive advantage in attracting, retaining and growing customers, while reducing fraud and mitigating risk. By incorporating IBM SPSS software into their daily operations, organizations become predictive enterprises – able to direct and automate decisions to meet business goals and achieve measurable competitive advantage. For further information or to reach a representative visit *http://www.ibm.com/spss*.

Technical support

Technical support is available to maintenance customers. Customers may contact Technical Support for assistance in using IBM Corp. products or for installation help for one of the supported hardware environments. To reach Technical Support, see the IBM Corp. web site at *http://www.ibm.com/support*. Be prepared to identify yourself, your organization, and your support agreement when requesting assistance.

Contents

Chapter
1

IBM SPSS Collaboration and Deployment Services - Essentials for Python

Overview

IBM® SPSS® Collaboration and Deployment Services provides a scripting framework with a set of APIs that advanced users and administrators can use to write independent routines or batch jobs that combine a set of routines for working with repository objects and jobs. This can greatly simplify bulk tasks, including the following:

- Changing security permissions for a large group of users
- Labeling or removing a label from a large number of folders or files
- Uploading or downloading a large number of folders or files

The framework includes the ability to perform tasks from the command line, as well as a rich API for interacting with the IBM® SPSS® Collaboration and Deployment Services Repository within your own Python code.

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

Installation

The scripting framework can be installed on the Windows, UNIX, and IBM i platforms. The scripting platform is independent of the platform used by the repository accessed by the scripting facility. For example, a repository running on the Windows platform can be called by scripting functions running on the UNIX platform.

Installing on Windows

1. If Python is already installed on your system, uninstall it.

2. Insert the installation media.

3. Open the *PYTHON\Disk1\InstData\NoVM* directory of Disk 2 and double-click *install.exe*. Follow the screen instructions to complete the installation. Install to the default location. This installs the required Python, ZSI, and PyXML technologies.

4. Open the *PYTHON* directory on the installation media and extract the contents of *cads-scripting-1.0.zip* to a temporary directory.

5. Add IBM® SPSS® Collaboration and Deployment Services - Essentials for Python directory location to your PC's **Path** system environment variable.
6. At a command prompt, change the current directory to the folder where you extracted *cads-scripting-1.0.zip*. Type the following command and press Enter.

   ```
   python setup.py install
   ```

Installing on UNIX

1. If Python 2.4.3, ZSI 2.0 rc3, and PyXML 0.8.4 are not already installed on your system, install after downloading from their respective web sites before proceeding to step 2.
   - Python 2.4.3: *http://www.python.org/download/releases/2.4.3/*
   - ZSI 2.0 rc3: *http://sourceforge.net/projects/pywebsvcs*
   - PyXML 0.8.4: *http://sourceforge.net/project/showfiles.php?group_id=6473*
2. Insert Disk 2.
3. Open the *PYTHON* directory and extract the contents of *cads-scripting-1.0.tar.gz* to a temporary directory.
4. In the temporary directory, edit *setup.cfg*. Replace `<PythonInstallDir>` with IBM® SPSS® Collaboration and Deployment Services - Essentials for Python installation path. If no value is specified, the path will default to Python library, for example */usr/lib/python2.4*.

   ```
   [install]
   install-base = <PythonInstallDir>
   install-data = <PythonInstallDir>
   install-purelib = <PythonInstallDir>
   install-scripts = <PythonInstallDir>
   install_headers = <PythonInstallDir>
   ```

5. At a command prompt, change the current directory to the folder where you extracted *cads-scripting-1.0.tar.gz*. Execute the following command:

   ```
   python setup.py install
   ```

   Note that when extracting tar bundles, some tar utilities may display a message similar to the following:

   ```
   tar: A lone zero block at ####
   ```

   The value of *####* corresponds to some integer value. These messages are only informational and do not indicate a failure in the extraction.

Installing on IBM i

1. Log into your IBM i system using a Telnet terminal.
2. Insert Disk 1.
3. Start QShell with the following command

   ```
   QSH
   ```

4. Change the directory to */qopt/Server/IBMi/Python*.
5. Copy the content of the directory to a temporary location.
6. Run the installation script by executing the following command:

   ```
   ./PyInst.scr
   ```

   Python is installed as */QOpenSys/usr/local/bin/python2.4* and IBM® SPSS® Collaboration and Deployment Services - Essentials for Python is installed in */QOpenSys/usr/local/lib/python2.4/site-packages*.

Chapter

2

Command line scripting

The Python file *CADSTool.py* can be used from the command line to manipulate resources stored within the IBM® SPSS® Collaboration and Deployment Services Repository. The general syntax used for calling IBM® SPSS® 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 2-1 lists the keywords supported by all IBM® SPSS® Collaboration and Deployment Services scripting functions. The second column lists any optional, shortened versions of the keywords. Note that keywords are case sensitive.

Table 2-1
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
`--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
`--ssl`		Indicates that the repository server uses the secure sockets layer (SSL) protocol to encrypt communications. Before using this keyword, the repository server must be configured for SSL. For more information, see the administrator documentation.
`-h`		The scripting module help information

Repository connections

You must specify the IBM® SPSS® Collaboration and Deployment Services Repository user ID, password, host, and port at the end of every command. The following methods can be used to provide this connection information:

- Using keywords, such as the following:

```
--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. For example, 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. Alternatively, if the user, password, host, and port are all 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 here, the syntax and examples use the `-z` parameter in an effort to use the minimum number of required parameters.

Content repository scripting

Content repository scripting offers the ability to work with repository resources, such as files and folders. This area includes the following functionality:

- Creating and deleting folders
- Uploading and downloading files
- Exporting and importing folders
- Managing labels, security, and metadata

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

Important: Keywords are case sensitive.

Table 2-2
Keywords for repository APIs

Keyword	**Optional Short Version**	**Usage**
`--source`	`-s`	The source file or 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 or folders
`--author`	`-a`	The author name for a file or folder
`--description`	`-d`	The description for a file or folder
`--title`	`-i`	The title for a file or folder
`--expirationDate`	`-q`	The expiration date for a file or folder
`--expirationStartDate`		The expiration start date for a file or folder
`--expirationEndDate`		The expiration end date for a file or folder
`--keyword`	`-k`	The keyword for a file or 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 principals
`--createVersion`	`-b`	Indicates that a new version of a file is to be created
`--contentLanguage`	`-g`	The content language for a file or folder
`--topic`		The topics assigned to a file or folder. You can enter multiple values such a `--topic "topic1;topic2"`
`--modifiedBy`		The user who modified a file or folder
`--mimeType`		The mime type of a file
`--createdBy`		The user who created a file or 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 file, the latest version is used.

Operations

The following sections list all repository scripting operations supported for IBM® SPSS® Collaboration and Deployment Services.

The advanceSearch operation

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 --label "Production" --keyword "Quarterly"
--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`

The applySecurity operation

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

Syntax

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

Where:

- <*source*> is the fully qualified IBM® SPSS® Collaboration and Deployment Services Repository path of the file or 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 or folder as part of the ACL. This is a required parameter.
- <*permission*> is the type of permission to apply to the specified file or 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 folder:

```
python CADSTool.py applySecurity --source "/Projects" --principal "icrod"
--permission "READ" --provider "Native" -z
```

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

```
python CADSTool.py applySecurity --source "/Projects/" --principal "icrod"
--permission "READ" --provider "Native" --cascade -z
```

Messages

The following messages may display when using this API:

- `<permission> permission set successfully for <source>.`
- `<source> No such file or folder exists.  Please try again.`
- `<permission> Invalid permission type, Please try again.`
- `<source> Error setting security ACL.`

The cascadeSecurity operation

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

Syntax

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

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

Example

```
python CADSTool.py cascadeSecurity --source "/Projects" -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.`

The copyResource operation

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 The moveResource operation on p. 17 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 "/Demo/Drafts/MyReport.rptdesign" --target
"/Projects" -z
```

The following example copies and renames a file:

```
python CADSTool.py copyResource --source "/Demo/Drafts/MyReport.rptdesign" --target
"/Projects/Report.rptdesign" -z
```

Messages

The following messages may display when using this API:

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

The createFolder operation

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

Syntax

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

The value of <*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 *Drafts* if it does not already exist.

```
python CADSTool.py createFolder --source "/Demo/Drafts" -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.`

The deleteFile operation

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 "/Demo/Drafts/MyReport.rptdesign" -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.`

The deleteFileVersion operation

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 "/Demo/Drafts/MyReport.rptdesign" --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 "/Demo/Drafts/MyReport.rptdesign" --label
"Test" -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 "Test" -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.`

The deleteFolder operation

`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 "/Demo/Drafts" -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.`

The downloadFile operation

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 IBM® SPSS® Collaboration and Deployment Services 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 "/Demo/Drafts/MyReport.rptdesign"
--target "C:/Demo/Shared/" -z
```

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

```
python CADSTool.py  downloadFile --source "/Demo/Drafts/MyReport.rptdesign" --version
"0:2006-08-25 21:15:49.453" --target "C:/Demo/Shared/" -z
```

The following example downloads a labeled version of the file:

```
python CADSTool.py  downloadFile --source "/Demo/Drafts/MyReport.rptdesign" --label "Production"
--target "C:/Demo/Shared/" -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.`

The export operation

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 "/Projects/" --target "C:\Demo\drafts.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.`

The getAccessControlList operation

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

Syntax

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

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

Example

```
python CADSTool.py getAccessControlList --source "/Projects/MyReport.rptdesign" -z
```

Messages

The following messages may display when using this API:

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

The getAllVersions operation

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 "/Demo/Drafts/MyReport.rptdesign" -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.

The getChildren operation

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

Syntax

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

The value of <*source*> is the fully qualified path of the folder. This is a required parameter.

Example

```
python CADSTool.py getChildren --source "/Demo/Drafts" -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.`

The getCustomPropertyValue operation

Retrieves the valid values accepted by a specified custom property.

Syntax

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

The value of <*propertyName*> is the name of the custom property. This is an optional parameter.

Example

```
python CADSTool.py getCustomPropertyValue --propertyName "Language" -z
```

Messages

The following messages may display when using this API:

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

The getMetadata operation

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 or 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 or folder to retrieve metadata for. This is an optional parameter.
- <*label*> is the label of the file or 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 folder:

```
python CADSTool.py getMetadata --source "/Demo/Drafts" -z
```

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

```
python CADSTool.py getMetadata --source "/Demo/Drafts/MyReport.rptdesign" --label "Test" -z
```

The following example retrieves metadata for a labeled version of a file 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 or folder is displayed, including any custom metadata properties.

The import operation

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\drafts.pes" --target "/Demo/Drafts/" -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.`

The moveResource operation

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 "/Demo/Drafts/MyReport.rptdesign" --target
"/Approved" -z
```

The following example moves a folder:

```
python CADSTool.py moveResource --source "/Demo/Drafts/" --target "/Projects" -z
```

The following example moves and renames a file:

```
python CADSTool.py moveResource --source "/Demo/Drafts/MyReport.rptdesign" --target
"/Approved/Report.rptdesign" -z
```

Messages

The following messages may display when using this API:

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

The removeLabel operation

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 "/Demo/Drafts/MyReport.rptdesign"
--label "Draft" -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.`

The removeSecurity operation

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 "/Projects/MyReport.rptdesign"
--principal "icrod" --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.`

The search operation

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

Syntax

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

The value of <*criteria*> is the search string used to search metadata for all files and folders in the repository. This is a required parameter.

Example

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

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.`

The setLabel operation

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 "/Demo/Drafts/MyReport.rptdesign" --version
"1:2006-08-25 21:15:49.453" --label "Beta" -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.`

The setMetadata operation

Applies metadata properties to files and folders in the repository. Table 2-3 lists the metadata properties and whether they can be applied to files and/or folders.

Table 2-3
Metadata properties and resource types

Metadata Property	**Resource Type**
Author	File
Description	File or Folder
Title	File or Folder
Expiration Date	File or Folder
Keyword	File
Topics	File
Custom Metadata	File or 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 or folder. This is an optional parameter.
- <*title*> is the title of the file or 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 or 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 or folder. This is an optional parameter.
- <*version*> is the specific version of the file or folder to apply metadata on. This is an optional parameter.
- <*label*> is the labeled version of the file or folder to apply metadata on. This is an optional parameter.

- <*topic*> is the topic to apply to the file or folder. This is an optional parameter.
- <*customProperty*> is the custom property values to apply to the file or 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 "/Demo/Drafts/MyReport.rptdesign" --version
"0:2006-08-25 21:15:49.453" --keyword "Quarterly"
--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 or folder exists.  Please try again.`
- `<source> Error setting metadata.`

The uploadFile operation

`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 *Drafts*:

```
python CADSTool.py  uploadFile --source "C:\Demo\MyReport.rptdesign"
--target "/Demo/Drafts" -z
```

If *MyReport.rptdesign* already exists in the*/Demo/Drafts* folder, use the `--createVersion` parameter:

```
python CADSTool.py  uploadFile --source "C:\Demo\MyReport.rptdesign"
--target "/Demo/Drafts"  --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

Process management scripting offers the ability to work with jobs. This area includes the following functionality:

- Executing jobs
- Retrieving job histories
- Retrieving job details

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 2-4 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 2-1 and Table 2-2.

Table 2-4
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 IBM® SPSS® Collaboration and Deployment Services. The syntax and examples shown contain the minimum number of required parameters.

The deleteJobExecutions operation

Deletes the specified job execution objects.

Syntax

```
python CADSTool.py deleteJobExecutions --execIds "<execIDs>" -z
```

The value of <*execIDs*> is a space-delimited list of identifiers for the executions to delete. This is a required parameter.

Examples

```
python CADSTool.py deleteJobExecutions --execIds
"0a58c33d002ce9080000 010e0ccf7b01800e" -z
```

Messages

The following messages may display when using this API:

- `Execution Id not specified.`

The executeJob operation

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 "/Demo/Jobs/Reports" -z
```

The following example runs the job synchronously with notifications:

```
python CADSTool.py  executeJob --source "/Demo/Jobs/Reports" --notification -z
```

The following example runs the job asynchronously without notifications:

```
python CADSTool.py  executeJob --source "/Demo/Jobs/Reports" --async -z
```

The following example runs the job asynchronously with notifications:

```
python CADSTool.py  executeJob --source "/Demo/Jobs/Reports" --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.`

The getJobExecutionDetails operation

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 identifier 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`

The getJobExecutionList operation

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

Syntax

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

The value of <*source*> is the fully qualified path of the job in the repository. This is a required parameter.

Example

```
python CADSTool.py  getJobExecutionList --source "/Demo/Jobs/Reports" -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.`

Chapter

3 The PESImpl module

The IBM® SPSS® Collaboration and Deployment Services - Essentials for Python facility allows interaction with IBM® SPSS® Collaboration and Deployment Services 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(user, password, host, port, ssl=True)
```

The parameters for the `PESImpl` constructor are as follows:

- *user* corresponds to the username
- *password* corresponds to the password associated with the specified user
- *host* designates the name of the repository server
- *port* specifies the port number for the repository server
- *ssl=True* indicates that the repository server uses the secure sockets layer (SSL) protocol for encrypting communications. If the ssl parameter is set to *False*, or if the parameter is omitted when creating the `PESImpl` object, the server communications will not use SSL. When using *ssl=True*, the repository server must be configured for SSL. For more information, see the administrator documentation.

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

Content repository API

Content repository scripting offers the ability to work with repository resources, such as files and folders. This area includes the following functionality:

- Creating and deleting folders
- Uploading and downloading files
- Exporting and importing folders
- Managing labels, security, and metadata

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

Methods

The following sections list all content repository methods supported for IBM® SPSS® Collaboration and Deployment Services.

Notes:

- For all methods 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 or folder, the latest version is used.
- For all methods that require a path to files or folders in the repository, either the path or the object URI can be used. The object URI can be obtained by viewing the object properties in IBM® SPSS® Collaboration and Deployment Services 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')
```

The advanceSearch method

`advanceSearch(`*`criteriaDict`*`,`*`submittedHierarchy`*`)`

Searches for files and folders in the repository, based on various parameters passed as input. 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

Note that currently `expirationStartDate` and `expirationEndDate` do not work when used in conjunction with other search fields (such as title, author, etc).

Table 3-1
Input parameters for advanceSearch

Field	Use	Type	Description	Example Value
criteriaDict	Required	Dictionary	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`	`{ "author":"admin", "title":"search", "label":"label 1", }`
submittedHierarchy	Optional	Boolean	Indicates whether to search the *Submitted Jobs* folder	`True` or `False`

Table 3-2
Return value for advanceSearch

Type	Description
`PageResult`	Structure in which each row corresponds to a search match.For more information, see the topic The PageResult class on p. 50.

Table 3-3
Exceptions for advanceSearch

Type	Description
`InsufficientParameterException`	Mandatory parameters are not specified.

Example

The following sample returns all file versions labeled *Production* that have a keyword value of *Quarterly*.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
critDict = {'label':'Production','keyword':'Quarterly'}
sResults = pesImpl.advanceSearch(critDict)
sRows = sResults.getRows()
for sRow in sRows:
   print "Author: ", sRow.getAuthor()
   print "Title: ", sRow.getTitle()
   for child in sRow.getChildRow():
      print "Version: ", child.getVersionMarker()
      print "Label: ", child.getVersionLabel()
      print "Keywords:", child.getKeyword()
      print "URI:", child.getUri()
```

The applySecurity method

```
applySecurity(source,principal,permission,provider,cascade)
```

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

Table 3-4
Input parameters for applySecurity

Field	**Use**	**Type**	**Description**	**Example Value**
source	Required	String	The fully qualified path or object URI of the file or folder in the repository	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000010dcee0eaa28219"`
principal	Required	String	The user (such as *admin*) to apply to the specified file or folder as part of the ACL	`admin`
permission	Required	String	The type of permission to apply to the specified file or folder	`READ, WRITE, DELETE, MODIFY_ACL, OR OWNER`
provider	Optional	String	The security provider to use for applying security to users (such as *Native*)	`Native`
cascade	Optional	Boolean	Propagates the security settings to all files and subfolders within the specified folder	`True` or `False`

Table 3-5
Return value for applySecurity

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

Table 3-6
Exceptions for applySecurity

Type	**Description**
`ResourceNotFoundException`	The source file does not exist.
`InsufficientParameterException`	Required parameters are not specified.
`IllegalParameterException`	The specified user or security provider name is incorrect.

Example

The following sample assigns the *READ* permission for the designated file to the user *icrod*.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
bSuccess = pesImpl.applySecurity(source="/Projects",principal="icrod",permission="READ",
    provider="Native")
```

The cascadeSecurity method

`cascadeSecurity(`*`source`*`)`

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

Table 3-7
Input parameters for cascadeSecurity

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path or object URI of the folder in the repository	`"/Temp Folder"` or `"0a58c3670016a7860000 010dcee0eaa28219"`

Table 3-8
Return value for cascadeSecurity

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

Table 3-9
Exceptions for cascadeSecurity

Type	Description
`ResourceNotFoundException`	The source folder does not exist.
`InsufficientParameterException`	Required parameters are not specified.

Example

The following sample cascades the security for the folder *Projects* to all children of the folder.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
bSuccess = pesImpl.cascadeSecurity(source="/Projects")
```

The copyResource method

`copyResource(`*`source`*`,`*`target`*`)`

Copies a file or folder to another folder in the repository. The specified source file or folder can be renamed when it is copied. See The moveResource method on p. 42 for more information on renaming.

Table 3-10
Input parameters for copyResource

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path or object URI of the file or folder in the repository	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000 010dcee0eaa28219"`
target	Required	String	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 or folder when it is copied.	`"/New Folder"` or `"/New Folder/abc.dat"`

Table 3-11
Return value for copyResource

Type	Description
`String`	URI of the copied file or folder

Table 3-12
Exceptions for copyResource

Type	Description
`ResourceNotFoundException`	The source file or target folder does not exist.
`InsufficientParameterException`	Required parameters are not specified.

Example

The following sample copies the *Drafts* folder to a folder named *Projects*.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
uri = pesImpl.copyResource(source="/Demo/Drafts/MyReport.rptdesign",target="/Projects")
print uri
```

The createFolder method

`createFolder(`*`source`*`)`

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

Table 3-13
Input parameters for createFolder

Field	Use	Type	Description	Example Value
source	Required	String	The folder to create in the repository	`"/Temp Folder/Sample Folder"`

Table 3-14
Return value for createFolder

Type	Description
`String`	URI of the created folder

Table 3-15
Exceptions for createFolder

Type	Description
`InsufficientParameterException`	Required parameters are not specified.
`ResourceAlreadyExistsException`	The specified folder already exists in the repository.

Example

The following example creates a folder named *Drafts* as a child of the *Demo* folder. If a problem creating the folder occurs, an exception message is sent to the console.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
try:
   uri = pesImpl.createFolder(source="/Demo/Drafts")
   print "URI for the folder is:", uri
except:
   print "Unhandled exception in createFolder."
```

The deleteFile method

`deleteFile(`*`source`*`,`*`submittedHierarchy`*`)`

Deletes a file from the repository. All versions of the file are deleted.

Table 3-16
Input parameters for deleteFile

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path or object URI of the file in the repository	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000 010dcee0eaa28219"`
sub-mitted-Hierar-chy	Optional	Boolean	Indicates whether the file is in the *Submitted Jobs* folder	`True` or `False`

Table 3-17
Return value for deleteFile

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

Table 3-18
Exceptions for deleteFile

Type	Description
`ResourceNotFoundException`	The source file does not exist.
`InsufficientParameterException`	Required parameters are not specified.
`IllegalParameterException`	The specified resource to delete is a folder.

Example

The following example deletes the file *MyReport.rptdesign* from the repository.

```
from pes.util.PESExceptions import *
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
try:
   bSuccess = pesImpl.deleteFile(source="/Demo/Drafts/MyReport.rptdesign")
except ResourceNotFoundException:
   print "Specified file does not exist."
except InsufficientParameterException:
   print "No file specified."
except IllegalParameterException:
   print "Item to be deleted is not a file."
```

The deleteFileVersion method

`deleteFileVersion(`*`source`*`,`*`version`*`,`*`label`*`,`*`submittedHierarchy`*`)`

Deletes a specific version of a file from the repository.

Table 3-19
Input parameters for deleteFileVersion

Field	**Use**	**Type**	**Description**	**Example Value**
source	Required	String	The fully qualified path or object URI of the file in the repository	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a78600 00010dcee0eaa28219"`
version	Optional. However, either *version* or *label* must be specified.	String	The specific version of the file to delete	`"0:2006-08-25 21:15:49.453"`
label	Optional. However, either *version* or *label* must be specified.	String	The specific labeled version of the file to delete	`"Version 1"`
submittedHierarchy	Optional	Boolean	Indicates whether the file is in the *Submitted Jobs* folder	`True` or `False`

Table 3-20
Return value for deleteFileVersion

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

Table 3-21
Exceptions for deleteFileVersion

Type	**Description**
`ResourceNotFoundException`	The source file or target folder does not exist.
`InsufficientParameterException`	Required parameters are not specified.
`IllegalParameterException`	The specified resource to delete is a folder.

Example

The following example deletes the version of the file *MyReport.rptdesign* labeled *Test* from the repository.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
bSuccess = pesImpl.deleteFileVersion(source="/Demo/Drafts/MyReport.rptdesign",label="Test")
```

The deleteFolder method

`deleteFolder(`*`source`*`,`*`submittedHierarchy`*`)`

Deletes a folder and its contents from the repository.

Table 3-22
Input parameters for deleteFolder

Field	**Use**	**Type**	**Description**	**Example Value**
source	Required	String	The fully qualified path or object URI of the folder in the repository	`"/Temp Folder"` or `"0a58c3670016a78600 00010dcee0eaa28219"`
submittedHierarchy	Optional	Boolean	Indicates whether the folder is in the *Submitted Jobs* folder	`True` or `False`

Table 3-23
Return value for deleteFolder

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

Table 3-24
Exceptions for deleteFolder

Type	**Description**
`ResourceNotFoundException`	The specified folder does not exist.
`InsufficientParameterException`	Required parameters are not specified.
`IllegalParameterException`	The specified resource to delete is not a folder.

Example

The following example deletes the folder named *Drafts* from the repository. If a problem deleting the folder occurs, an exception message is sent to the console.

```
from pes.util.PESExceptions import *
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
try:
   bSuccess = pesImpl.deleteFolder(source="/Demo/Drafts")
except ResourceNotFoundException:
   print "Specified folder does not exist."
except InsufficientParameterException:
   print "No folder specified."
except IllegalParameterException:
   print "Item to be deleted is not a folder."
```

The downloadFile method

`downloadFile(`*`source`*`,`*`target`*`,`*`version`*`,`*`label`*`)`

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

Table 3-25
Input parameters for downloadFile

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified repository path or object URI of the file to download	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000 010dcee0eaa28219"`
target	Required	String	The fully qualified path (on the local file system) of the folder to which to download the file	`"C:\Temp"`
version	Optional. Either version or label can be specified.	String	The specific version of the file to download	`"0:2006-08-25 21:15:49.453"`
label	Optional. Either version or label can be specified.	String	The specific labeled version of the file to download	`"Version 2"`

Table 3-26
Return value for downloadFile

Type	Description
`Resource`	Container for information about a repository object.For more information, see the topic The Resource class on p. 49..

Table 3-27
Exceptions for downloadFile

Type	Description
`ResourceNotFoundException`	The source file or target folder does not exist.
`InsufficientParameterException`	Required parameters are not specified.

Example

The following sample downloads a version labeled *Production* of the file *MyReport.rptdesign* to the *Shared* directory on the local file system.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
resource = pesImpl.downloadFile(source="/Demo/Drafts/MyReport.rptdesign",
    target="c:/Demo/Shared",label="Production")
```

The exportResource method

`exportResource(`*`source`*`,`*`target`*`)`

Exports a specified repository folder to a designated **.pes* export file on the local file system.

Table 3-28
Input parameters for exportResource

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

Table 3-29
Return value for exportResource

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

Table 3-30
Exceptions for exportResource

Type	Description
`ResourceNotFoundException`	The source file or target folder does not exist.
`InsufficientParameterException`	Required parameters are not specified.
`IllegalParameterException`	The specified target is a folder. The target must be a **.pes* file.

Example

The following sample exports the contents of the *Drafts* folder to an export file in the *backups* folder of the local file system.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
bSuccess = pesImpl.exportResource(source="/Projects",target="C:\Demo\drafts.pes")
```

The getAccessControlList method

`getAccessControlList(`*`source`*`,`*`submittedHierarchy`*`)`

Retrieves the security access control list (ACL) for the specified file or folder in the repository.

Table 3-31
Input parameters for getAccessControlList

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path or object URI of the file or folder in the repository	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a78600 00010dcee0eaa28219"`
submittedHierarchy	Optional	Boolean	Indicates whether the file or folder is in the *Submitted Jobs* folder	`True` or `False`

Table 3-32
Return value for getAccessControlList

Type	Description
`Dictionary`	A dictionary is displayed containing the user name(s) and the associated permission. For example: `{"admin":"MODIFY_ACL","Joe":"DELETE"}`

Table 3-33
Exceptions for getAccessControlList

Type	Description
`ResourceNotFoundException`	The source file or target folder does not exist.
`InsufficientParameterException`	Required parameters are not specified.

Example

The following example prints the ACL for the file *MyReport.rptdesign*.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
aclDic = pesImpl.getAccessControlList(source = "/Projects/MyReport.rptdesign")
print aclDic
```

The getAllVersions method

`getAllVersions(`*`source`*`,`*`submittedHierarchy`*`)`

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

Table 3-34
Input parameters for getAllVersions

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path or object URI of the file in the repository	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000010dcee0eaa28219"`
submittedHierarchy	Optional	Boolean	Indicates whether the file is in the *Submitted Jobs* folder	`True` or `False`

Table 3-35
Return value for getAllVersions

Type	Description
`List`	A list of resource objects. See The Resource class on p. 49.

Table 3-36
Exceptions for getAllVersions

Type	Description
`ResourceNotFoundException`	The source file does not exist.
`InsufficientParameterException`	Required parameters are not specified.
`IllegalParameterException`	The specified source is a folder.

Example

This example retrieves information about all versions of the file *MyReport.rptdesign*, printing the author, version marker, and version labels for each.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
resourceList = pesImpl.getAllVersions(source="/Demo/Drafts/MyReport.rptdesign")
for resource in resourceList:
   print resource.getAuthor()
   print resource.getVersionMarker()
   print resource.getVersionLabel()
```

The getChildren method

getChildren(*source*,*submittedHierarchy*)

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

Table 3-37
Input parameters for getChildren

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path or object URI of the folder in the repository	`"/Temp Folder"` or `"0a58c3670016a7860000010dcee0eaa28219"`
submittedHierarchy	Optional	Boolean	Indicates whether the folder is in the Submitted Jobs folder	`True` or `False`

Table 3-38
Return value for getChildren

Type	Description
`List`	A list of resource objects. See The Resource class on p. 49.

Table 3-39
Exceptions for getChildren

Type	Description
`InsufficientParameterException`	Required parameters are not specified.
`ResourceNotFoundException`	The folder does not exist.

Example

The following sample retrieves the children of the */Demo/Drafts* folder, printing the title, author, and resource identifier for each.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
resourceList = pesImpl.getChildren(source="/Demo/Drafts")
for resource in resourceList:
   print "Resource title:", resource.getTitle()
   print "Resource author:", resource.getAuthor()
   print "Resource ID:", resource.getResourceID()
```

The getCustomPropertyValue method

`getCustomPropertyValue(`*`propertyName`*`)`

Retrieves the valid values accepted by a specified custom property.

Table 3-40
Input parameters for getCustomPropertyValue

Field	Use	Type	Description	Example Value
propertyName	Required	String	The name of the custom property	`"FreeForm"`

Table 3-41
Return value for getCustomPropertyValue

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`).

Table 3-42
Exceptions for getCustomPropertyValue

Type	Description
`ResourceNotFoundException`	The specified property does not exist.
`InsufficientParameterException`	Required parameters are not specified.

Example

The following sample accesses the values for the custom property *Language*.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
langList = pesImpl.getCustomPropertyValue(propertyName = "Language")
print langList
```

The getMetadata method

`getMetadata(`*`source`*`,`*`version`*`,`*`label`*`,`*`submittedHierarchy`*`)`

Retrieves the metadata attributes of a file or folder in the repository, including any custom properties and topic information.

Table 3-43
Input parameters for getMetadata

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path or object URI of the file or folder in the repository	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000010dcee0eaa28219"`

Field	Use	Type	Description	Example Value
version	Optional. Either version or label can be specified.	String	The specific version of the file or folder	`"0:2006-08-25 21:15:49.453"`
label	Optional. Either version or label can be specified.	String	The specific labeled version of the file or folder	`"Version 1"`
submittedHierarchy	Optional	Boolean	Indicates whether the file is in the *Submitted Jobs* folder	`True` or `False`

Table 3-44
Return value for getMetadata

Type	Description
`Resource`	Container for information about a repository object.For more information, see the topic The Resource class on p. 49.

Table 3-45
Exceptions for getMetadata

Type	Description
`ResourceNotFoundException`	The source file or folder does not exist.
`InsufficientParameterException`	Required parameters are not specified.

Example

The following example accesses the resource identifier for the */Demo/Drafts* folder.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
resource = pesImpl.getMetadata(source="/Demo/Drafts")
resourceid = resource.getResourceID()
```

The importResource method

`importResource(`*`source`*`,`*`target`*`)`

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

Table 3-46
Input parameters for importResource

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path (on the local file system) of the file to import	`"C:\Temp\New.pes"`
target	Required	String	The fully qualified repository path or object URI of the folder into which to import	`"/Temp Folder"` or `"0a58c3670016a7860000010dcee0eaa28219"`

Table 3-47
Return value for importResource

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

Table 3-48
Exceptions for importResource

Type	Description
`ResourceNotFoundException`	The source file or target folder does not exist.
`InsufficientParameterException`	Required parameters are not specified.

Example

The following sample .

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
bSuccess = pesImpl.importResource(source="C:\Demo\drafts.pes",target="/Demo/Drafts")
```

The moveResource method

`moveResource(`*`source`*`,`*`target`*`)`

Moves a file or folder to another folder in the repository. A specified source file can be renamed when it is moved, with the target type and existence determining the final name. The following table describes the behavior of the renaming feature when moving a file:

Table 3-49
File renaming

Target Type	Target Folder Exists	Target Folder Does Not Exist
folder	Source file becomes a child of the target folder.	Source file moves to the parent folder of the specified target and is renamed to the target folder name.
file	Source file moves to the folder containing the target file and is renamed to the target name.	Error reported.

For example, if the source is the file */Temp Folder/Temp.txt* and the specified target is the folder */Demo Folder*, the following results may occur:

- If folder *Demo Folder* exists, *Temp.txt* is moved to *Demo Folder*.
- If folder *Demo Folder* does not exist, *Temp.txt* is moved to “ / “ and renamed to *Demo Folder*.

Alternatively, if the source is */Temp Folder/Temp.txt* and the specified target is the file*/Demo Folder/Abc.dat*, the following results may occur:

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

Table 3-50
Input parameters for moveResource

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path or object URI of the file or folder in the repository	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000010dcee0eaa28219"`
target	Required	String	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 or folder when it is moved.	`"/New Folder"` or `"/New Folder/abc.dat"`

Table 3-51
Return value for moveResource

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

Table 3-52
Exceptions for moveResource

Type	Description
`ResourceNotFoundException`	The specified source does not exist.
`InsufficientParameterException`	Required parameters are not specified.

Example

The following sample .

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
bSuccess = pesImpl.moveResource(source="/Demo/Drafts/MyReport.rptdesign",target="/Approved")
print bSuccess
```

The removeLabel method

`removeLabel(`*`source`*`,`*`label`*`)`

Removes a label from a file in the repository.

Table 3-53
Input parameters for removeLabel

Field	Use	Type	Example Value	Description
source	Required	String	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000010dcee0eaa28219"`	The fully qualified path or object URI of the file in the repository
label	Required	String	`"Version 1"`	The label name to remove

Table 3-54
Return value for removeLabel

Type	Description
`String`	URI of the updated file

Table 3-55
Exceptions for removeLabel

Type	Description
`ResourceNotFoundException`	The source file does not exist.
`InsufficientParameterException`	Required parameters are not specified.

Example

The following sample removes the label *Draft* from the file *MyReport.rptdesign*.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
uri = pesImpl.removeLabel(source="/Demo/Drafts/MyReport.rptdesign", label="Draft")
```

The removeSecurity method

`removeSecurity(`*`source`*`,`*`principal`*`,`*`provider`*`,`*`cascade`*`)`

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

Table 3-56
Input parameters for removeSecurity

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path or object URI of the file or folder in the repository	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000010dcee0eaa28219"`
principal	Required	String	The user (such as *admin*) to remove from the specified file or folder	`admin`
provider	Optional	String	The security provider (such as *Native*) to use for obtaining the information about users	`Native`
cascade	Optional	Boolean	Propagates the security settings to all files and subfolders within the specified folder	`True` or `False`

Table 3-57
Return value for removeSecurity

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

Table 3-58
Exceptions for removeSecurity

Type	Description
`ResourceNotFoundException`	The source file or target folder does not exist.

Type	Description
`InsufficientParameterException`	Required parameters are not specified.
`IllegalParameterException`	The specified user or security provider name is incorrect.

Example

The following sample removes the ACL for the principal *icrod* from the file *MyReport.rptdesign*.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
bSuccess = pesImpl.removeSecurity(source="/Projects/MyReport.rptdesign",principal="icrod")
```

The search method

search(*criteria*)

Searches for files in the repository, returning a list of file versions having metadata content that matches the search criteria.

Table 3-59
Input parameters for search

Field	Use	Type	Description	Example Value
criteria	Required	String	The value used to search file metadata	`"Age"`

Table 3-60
Return value for search

Type	Description
`PageResult`	Structure in which each row corresponds to a search match.For more information, see the topic The PageResult class on p. 50.

Table 3-61
Exceptions for search

Type	Description
`InsufficientParameterException`	Required parameters are not specified.

Example

The following searches for file versions that have the text *Quarterly* in any metadata fields.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
sResults = pesImpl.search(criteria="Quarterly")
sRows = sResults.getRows()
for sRow in sRows:
   print "Author: ", sRow.getAuthor()
   print "Title: ", sRow.getTitle()
   for child in sRow.getChildRow():
      print "Version: ", child.getVersionMarker()
      print "Label: ", child.getVersionLabel()
      print "Keywords:", child.getKeyword()
      print "URI:", child.getUri()
```

The setLabel method

setLabel(*source*,*version*, *label*)

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

Table 3-62
Input Parameters for setLabel

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path or object URI of the file in the repository	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000 010dcee0eaa28219"`
version	Required	String	The specific version of the file	`"0:2006-08-25 21:15:49.453"`
label	Required	String	The label to apply to the file	`"Version 1"`

Table 3-63
Return value for setLabel

Type	Description
`String`	URI of the updated file

Table 3-64
Exceptions for setLabel

Type	Description
`ResourceNotFoundException`	The source file or version does not exist.
`InsufficientParameterException`	Required parameters are not specified.

Example

The following sample assigns the label *Beta* to the second version of the file *MyReport.rptdesign*. The `getVersionMarker` method for a `Resource` object returns the marker for the desired version to be labeled.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
betaVersion = \
    pesImpl.getAllVersions(source="/Demo/Drafts/MyReport.rptdesign")[1].getVersionMarker()
print "Marker for the beta version is:", betaVersion
uri = pesImpl.setLabel(source="/Demo/Drafts/MyReport.rptdesign", version=betaVersion,
    label="Beta")
```

The setMetadata method

`setMetadata(`*`source`*`,`*`version`*`,`*`label`*`,`*`props`*`)`

Applies metadata properties to files and folders in the repository. The following table identifies the metadata properties and whether they can be applied to files and/or folders.

Table 3-65
Repository Object Properties

Metadata Property	Resource Type
Author	File
Description	File or Folder
Title	File or Folder
Expiration Date	File or Folder

Metadata Property	Resource Type
Keyword	File
Topics	File
Custom Metadata	File or Folder

Table 3-66
Input parameters for setMetadata

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path or object URI of the file or folder in the repository	`"/Temp Folder/Temp.txt"` or `"0a58c3670016a7860000 010dcee0eaa28219"`
version	Optional. Eitherversion or label can be specified.	String	The specific version of the file to be downloaded	`"0:2006-08-25 21:15:49.453"`
label	Optional. Eitherversion or label can be specified.	String	The label of the specific version	`"Label 1"`
props	Required	Dictionary	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.	`{ 'author':'admin', 'title':'newTitle', 'description','desc', 'topic':[a,b], 'customProperty': { 'language':'hindi|english', 'FreeForm': 'abcd' } }`

Table 3-67
Return value for setMetadata

Type	Description
`String`	URI of the file or folder for which metadata was set

Table 3-68
Exceptions for setMetadata

Type	Description
`InsufficientParameterException`	Required parameters are not specified.
`ResourceNotFoundException`	The source file or folder does not exist.

Example

The following sample assigns the keyword *Quarterly* to the *Production* version of the file *MyReport.rptdesign.*

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
pDict = {'keyword':'Quarterly'}
uri = pesImpl.setMetadata(source="/Demo/Drafts/MyReport.rptdesign",version=prodVersion,
    props=pDict)
print uri
```

The uploadFile method

`uploadFile(`*`source`*`,`*`target`*`,`*`versionFlag`*`)`

Saves a file to the repository from the local file system, with the option of creating a new version of the file if it already exists.

Table 3-69
Input parameters for uploadFile

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path (on the local file system) of the file to upload	`"C:\Temp\Temp.txt"`
target	Required	String	The fully qualified path of the destination folder in the repository where the file is to be uploaded	`"/Temp Folder"`
versionFlag	Optional	Boolean	If the specified file already exists, a new version of the file is created	`True` or `False`

Table 3-70
Return value for uploadFile

Type	Description
`String`	URI of the uploaded file

Table 3-71
Exceptions for uploadFile

Type	Description
`ResourceNotFoundException`	The source file or target folder does not exist.
`ResourceAlreadyExistsException`	A file or folder with the same name as the source file exists in the target folder and the `versionFlag` parameter is not specified.
`InsufficientParameterException`	Required parameters are not specified.

Example

This example uploads the file *MyReport.rptdesign* to the */Demo/Drafts* folder in the repository. If the file already exists, a new version of the file is uploaded using the `versionFlag` parameter.

```
from pes.util.PESExceptions import *
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
try:
   uri = pesImpl.uploadFile(source="C:\Demo\MyReport.rptdesign",target="/Demo/Drafts")
   print "URI for the uploaded file is: ", uri
except ResourceAlreadyExistsException:
   uri = pesImpl.uploadFile(source="C:\Demo\MyReport.rptdesign",target="/Demo/Drafts",
      versionFlag=True)
```

```
print "URI for the uploaded file is: ", uri
```

Wrapper classes

The `PESImpl` API includes classes serving as wrappers for objects returned from the web services called by the content repository methods. These wrapper classes provide an interface for displaying the information returned by the methods.

The Resource class

The `Resource` class acts as a simplified wrapper to the repository object `ResourceSpecifer.Resource`, offering access to object-specific information. In addition to the standard metadata associated with repository objects, this class includes any custom metadata information defined for objects in the repository. Table 3-72 lists all methods available in the `Resource` class.

Table 3-72
Methods for the Resource class

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 MIME 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
`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 resource identifier 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 the 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

Method Name	Description
`getCustomMetadata`	Returns any custom properties associated with the object as a dictionary
`getKeywordList`	Returns a list of keywords associated with the object

The IdentificationSpecifier class

This class acts as a simplified wrapper to the repository object `IdentificationSpecifier`, allowing access to identification-specific data for the object. Table 3-73 lists all methods available in the `IdentificationSpecifier` class.

Table 3-73
Methods for the IdentificationSpecifier class

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 version as a string

The PageResult class

This `PageResult` class serves as a container for search results. An individual hit in the results corresponds to a row in the `PageResult` object. For example, a search that returns four resources would yield a `PageResult` object containing four rows. Table 3-74 lists all methods available in the `PageResult` class.

Table 3-74
Methods for the PageResult class

Method Name	Description
`getRows`	Returns a list of `SearchRow` objects. For more information, see the topic The SearchRow class on p. 50.

The SearchRow class

The `SearchRow` class serves as a container for object-level information about an individual search result. You can access metadata about an object using the methods of this class.Table 3-75 lists all methods available in the `SearchRow` class.

Table 3-75
Methods for the SearchRow class

Method Name	Description
`getTitle`	Returns the name of the file or folder
`getAuthor`	Returns the author of the file or folder
`getMIMEType`	Returns the MIME type of the file
`getObjectLastModifiedBy`	Returns the user who last modified the file or folder
`getModified`	Returns the date and time the file or folder was last modified
`getFolderPath`	Returns the location of the file or folder

Method Name	Description
`getFolder`	Returns the name of parent folder of the file or folder
`getParentURI`	Returns the object URI of the parent
`getTopic`	Returns the topics associated with the file or folder
`getChildRow`	Returns the list of `SearchChildRow` objects (see the following section for more information)

To access information at the version level for an object, use the `getChildRow` method to return child rows corresponding to object versions.

The SearchChildRow class

The `SearchChildRow` class serves as a container for version-level information about an individual search result. You can access metadata about an object version using the methods of this class. Table 3-76 lists all methods available in the `SearchChildRow` class.

Table 3-76
Methods for the SearchChildRow class

Method Name	Description
`getExpirationDate`	Returns the expiration date of the file or folder
`getKeyword`	Returns the keywords associated with the version of the file or folder
`getVersionLabel`	Returns the version label of the file or folder
`getDescription`	Returns the description of the file or folder
`getLanguage`	Returns the language
`getVersionCreationDate`	Returns date and time the file or folder was created
`getVersionMarker`	Returns the version marker of the file or folder
`getUri`	Returns the object URI of the file or folder

Process management API

Process management scripting offers the ability to work with jobs. This area includes the following functionality:

- Executing jobs
- Retrieving job histories
- Retrieving job details

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

Methods

The following sections list all process management scripting methods supported for IBM® SPSS® Collaboration and Deployment Services.

Note: For all methods that require a path to files/folders in the repository, either the path or the object URI can be used. The object URI can be obtained by viewing the object properties in IBM® SPSS® Collaboration and Deployment Services Deployment Manager.

The cancelJob method

`cancelJob(`*`executionId`*`)`

Cancels a running job.

Table 3-77
Input parameters for cancelJob

Field	**Use**	**Type**	**Description**	**Example Value**
executionId	Required	String	Execution ID for the job	`0a58c33d002ce90800 00010e0ccf7b01800e`

Table 3-78
Return value for cancelJob

Type	**Description**
`Boolean`	Returns a success message when the job is cancelled

Example

This example terminates execution of the *Reports* job.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
execId = pesImpl.executeJob(source='/Demo/Jobs/Reports', notification = True,
    asynchronous=True)
print "Execution ID: ", execId
status = pesImpl.cancelJob(execId)
print "Successful cancellation: ",  status
```

The deleteJobExecutions method

`deleteJobExecutions(`*`executionId`*`)`

Deletes one or more job executions.

Table 3-79
Input parameters for deleteJobExecutions

Field	**Use**	**Type**	**Description**	**Example Value**
executionId	Required	List	List of job execution identifiers	`[0a58c33d002ce9080000010e0ccf7b01800e, 0a59c33d002ce9080060010e0cdf7b01802r]`

Table 3-80
Return value for deleteJobExecutions

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

Example

This example deletes the executions for the *Reports* job.

```
from pes.util.PESExceptions import *
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
executions = pesImpl.getJobExecutionList(source="/Demo/Jobs/Reports")
execRows = executions.getRows()

# Get the execution ID from the execution history
deleteList = []
for exrow in execRows :
     uuid = exrow.getEventObjId()
     deleteList.append(uuid)

if len(deleteList) != 0:
     print 'Deleting ',len(deleteList) ,' histories'
     pesImpl.deleteJobExecutions(deleteList)
```

The executeJob method

`executeJob(`*`source`*`,`*`notification`*`,`*`asynchronous`*`)`

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

Table 3-81
Input parameters for executeJob

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path (on the local file system) of the file to upload	`"C:\Temp\Temp.txt"`
notification	Optional	Boolean	Indicates whether the job runs with or without notifications. Default is `False`.	`True` or `False`
asynchronous	Optional	Boolean	Indicates whether the job runs asynchronously. Default is `False`.	`True` or `False`

Table 3-82
Return value for executeJob

Type	Description
`String`	The unique identifier for the execution of the job. This identifier is used to reference a particular job execution.

Example

This example initiates execution of the *Reports* job asynchronously with notifications.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
execId = pesImpl.executeJob(source='/Demo/Jobs/Reports', notification = True,
    asynchronous=True)
print "Execution ID: ", execId
```

The getJobExecutionDetails method

`getJobExecutionDetails(`*`executionId`*`,`*`log`*`,`*`target`*`)`

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

Table 3-83
Input parameters for getJobExecutionDetails

Field	Use	Type	Description	Example Value
executionId	Required	String	The execution Id of the job	`0a58c33d002ce9080000 010e0ccf7b01800e`
log	Optional	Boolean	Indicates whether the job log is displayed inline	`True` or `False`
target	Optional	String	The location (on the local file system) to store the logs. Only used in conjunction with the `--log` parameter.	`"c:\logs"`

Table 3-84
Return value for getJobExecutionDetails

Type	Description
`jobExecutionDetails`	Details about a job execution. For more information, see the topic The jobExecutionDetails class on p. 56.

Example

This example retrieves information about job step executions for the job execution with identifier *execId* , sending result for each step to the console.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
execDetails = pesImpl.getJobExecutionDetails(executionId=execId)
print "Job ID: ", execDetails.getUUID()
print "Event ID: ", execDetails.getEventUUID()
print "Started: ", execDetails.getStartDateTime()
print "Ended: ", execDetails.getEndDateTime()
for step in execDetails.getJobStepDetails():
     print "Step ID: ", step.getEventUUID()
     print "Step Name: ", step.getEventName()
     print "Started: ", step.getStartDateTime()
     print "Ended: ", step.getEndDateTime()
     print "Success: ", step.getExecutionSuccess()
```

The getJobExecutionList method

`getJobExecutionList(`*`source`*`)`

Lists the runs for a specific job, including any currently running jobs and completed jobs, for all versions of the job.

Table 3-85
Input parameters for getJobExecutionList

Field	Use	Type	Description	Example Value
source	Required	String	The fully qualified path of the job in the repository.	`"/testJob"`

Table 3-86
Return value for getJobExecutionList

Type	Description
`PageResult`	Container for the list of executions for a job. See For more information, see the topic The PageResult class on p. 55..

Example

This example retrieves the executions for the *Reports* job, sending information about each execution to the console.

```
from pes.api.PESImpl import PESImpl
pesImpl = PESImpl("admin", "spss", "localhost", "8080")
executions = pesImpl.getJobExecutionList(source="/Demo/Jobs/Reports")
execRows = executions.getRows()
if execRows:
   for exrow in execRows:
      print "Job Path: ", exrow.getPath()
      print "Object ID: ", exrow.getObjId()
      print "Event ID: ", exrow.getEventObjId()
      print "Version ", exrow.getVersionMarker()
      print "Started: ", exrow.getEventStartDateTime()
      print "Ended: ", exrow.getEventEndDateTime()
```

Wrapper classes

The `PESImpl` API includes classes serving as wrappers for objects returned from the web services called by the process management methods. These wrapper classes provide an interface for displaying the information returned by the methods.

The PageResult class

This `PageResult` class serves as a container for job execution results, allowing retrieval of job execution specific data . An individual job execution corresponds to a row in the `PageResult` object. For example, a job that had been executed four times corresponds to a `PageResult` object containing four rows. Table 3-87 lists all methods available in the `PageResult` class.

Table 3-87
Methods for the PageResult class

Method Name	Description
`getRows`	Returns a list of `Row` objects, each representing an execution of a job. For more information, see the topic The Row class on p. 56.

The Row class

The `Row` class serves as a container for job-level information about a job execution. You can access metadata about a job execution using the methods of this class.Table 3-88 lists all methods available in the `Row` class.

Table 3-88
Methods for the Row class

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

The jobExecutionDetails class

This class is returned from the `getJobExecutionDetails` method. It stores the run details for a job and includes a list of `jobStepExecution` objects providing information about each step in the job. Table 3-89 lists all methods available in the `jobExecutionDetails` class.

Table 3-89
Methods for the jobExecutionDetails class

Method Name	Description
`getJobStepDetails`	Returns a list of `jobStepExecutionDetails` objects. For more information, see the topic The jobStepExecutionDetails class on p. 57.
`getArtifactLocation`	Returns a list of job artifact locations
`getCompletionCode`	Returns the completion code of the job execution
`getEndDateTime`	Returns the end date and time of the job execution
`getEventName`	Returns the event name of the job execution
`getEventUUID`	Returns the event ID of the job execution
`getExecutionState`	Returns the run state of the job execution
`getExecutionSuccess`	Returns success or failure status of the job execution

Method Name	**Description**
`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 execution
`getStartDateTime`	Returns the start date and time of the job execution
`getUserName`	Returns the name of the user who ran the job
`getUUID`	Returns the execution ID of the job

The jobStepExecutionDetails class

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 3-90 lists all methods available in the `jobStepExecutionDetails` class.

Table 3-90
Methods for the jobStepExecutionDetails class

Method Name	**Description**
`getJobStepChldExecution-List`	Returns a list of `jobStepChildEx-ecutionDetails` objects. For more information, see the topic The jobStepChildExecutionDetails class on p. 58.
`getArtifactLocation`	Returns a list of job step artifact locations
`getCompletionCode`	Returns the completion code of the job step
`getEndDateTime`	Returns the end date and time of the job step
`getEventName`	Returns the event name of the job step
`getEventUUID`	Returns the event ID of the job step
`getExecutionState`	Returns the run state of the job step
`getExecutionSuccess`	Returns success or failure status of the job step
`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 step
`getStartDateTime`	Returns the start date and time of the job step
`getUserName`	Returns the name of the user who ran the job step
`getUUID`	Returns the execution ID of the job step

The jobStepChildExecutionDetails class

The `jobStepChildExecutionDetails` class serves as a container for child executions of individual job steps. For example, an iterative report job step produces a child execution for each iteration of the step. You can access metadata about the child executions using the methods of this class. Table 3-91 lists all methods available in the `jobStepChildExecutionDetails` class.

Table 3-91
Methods for the jobStepChildExecutionDetails class

Method Name	Description
`getArtifactLocation`	Returns a list of child execution artifact locations
`getCompletionCode`	Returns the completion code of the child execution
`getEndDateTime`	Returns the end date and time of the child execution
`getEventName`	Returns the event name of the child execution
`getEventUUID`	Returns the event ID of the child execution
`getExecutionState`	Returns the run state of the child execution
`getExecutionSuccess`	Returns success or failure status of the child execution
`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 child execution
`getStartDateTime`	Returns the start date and time of the child execution
`getUserName`	Returns the name of the user who ran the child execution
`getUUID`	Returns the execution ID of the child execution

Notices

Trademarks

Index