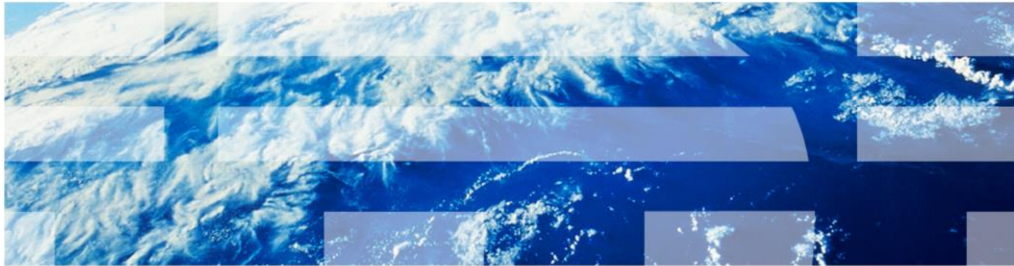
IBM

IBM PureApplication System

Command-line interface



© 2012 IBM Corporation

There are three user interfaces in the IBM PureApplication™ System: the administrative console, the rest APIs, and the command-line interface.

This presentation covers the command-line interface option.

IBM

Table of contents

- Overview
- Download and installation
- Syntax and help
- Command examples
- Summary

 © 2012 IBM Corporation

This presentation will provide a high level overview of the command-line interface in the IBM PureApplication System. It will discuss how to download and install the tool, provide some basics about the command-line interface syntax and help features, and will provide various command examples to help understand the tool.

Section

Overview

© 2012 IBM Corporation

This section provides a brief overview of the command-line interface tool.

Overview

- Scripted or command-line non-graphical interface
- Runs Python scripts or individual Python C commands in a Jython scripting environment
- Interactive, command, or batch modes
- Built upon the REST APIs
- Not necessarily a one-to-one mapping between the console interface and command-line interface
- Provides both workload console and system console support
  - Workload console support: **deployer.<object.***>**
  - System console support: **admin.<object.***>**
  - Many new objects

4 Command-line interface © 2012 IBM Corporation

The command-line interface enables you to manage your environment remotely, in a non-graphical interface, either in a command-line format or a script format. It provides a Jython scripting environment, by running scripts or Python C commands in three different modes, namely interactive mode, command mode, and batch mode. Under the covers, the command-line interface is built upon the REST APIs, so whatever can be done with a REST API should be able to be done with the command-line interface. Sometimes, looking at the REST API documentation correlating to a command-line interface command will help you better understand what the command-line interface is doing.

With the command-line interface, you can perform most functions that are available from the console, however, not all functionality found in the administrative console has a one-to-one mapping to the command-line interface. Some functions can only be done using the administrative console and some can only be done with the command-line interface. For example, the virtual application builder functionality is only available from the administrative console. Conversely, at this time, some pattern exports and imports can only be done with the command-line interface.

Overall the PureApplication System command-line interface has few structural changes from the IBM Workload Deployer command-line interface. The biggest change from Workload Deployer is the separation of commands into two categories that correlate with the separation of commands in the administrative console, namely the system console commands and the workload console commands. In the command-line interface. the system console commands are prefixed with the new “admin” verb, while the workload console commands are prefixed with the “deployer” verb as they were in Workload Deployer. And there are new commands for all the new or replaced objects in the PureApplication System.

IBM

Section

Download and installation

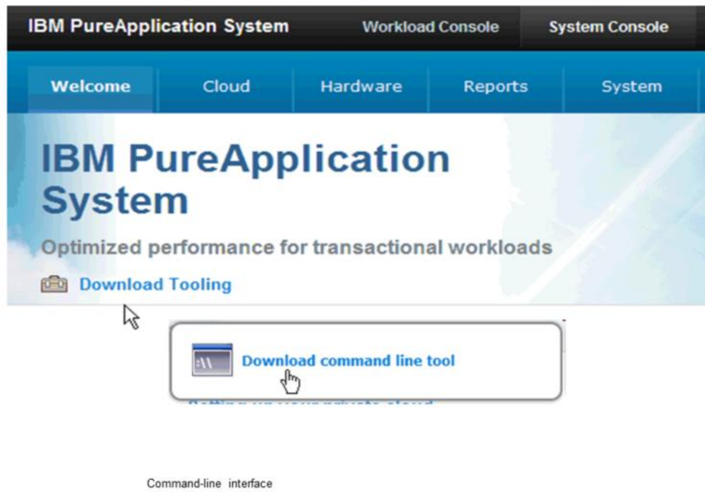
© 2012 IBM Corporation

The next section will discuss the command-line interface download and installation.

IBM

Download the command-line interface tool

- Download the command-line interface tool to your local system
  – Any platform with a supported JRE
- See information center if:
  – running on Windows® Server 2003 or 2008, or
  – Using multi-byte character sets



6 Command-line interface © 2012 IBM Corporation

If you have never downloaded the command-line interface tool from a PureApplication System, then you will need to do so. The Workload Deployer version of the tool will not work for a PureApplication System.

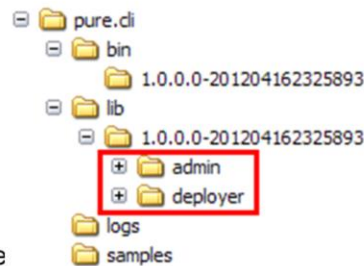
To download the command-line interface, go to the welcome page in either the workload console or the system console, select the download link, and download and extract the archive file to your local workstation. You can run the command-line interface on any platform that contains a supported JRE.

If you are running on a Windows Server 2003 or a Windows Server 2008, see the information center for information about customizing the command-line interface registry. Additionally, if you are using a multi-byte character set, then also see the information center for a command-line interface registry customization required. Traditional and simplified Chinese, Korean, and Japanese languages are some examples that require this registry customization.

IBM

Extract command-line interface

- Command-line Interface download is packaged as “**pure.cli<firmware>.zip**”
- Requires Java Runtime Environment (JRE), version 6 or above
  - CLI will run on any platform with a JRE
- Extract into local file system
- Always run from <**CLI Install>/pure.cli/bin**
- Upon connect, will automatically download the required version if necessary
- The version directories are never deleted
- **cli.log** file under the **logs** directory for debugging purposes
- Many script samples under the **samples** directory



 © 2012 IBM Corporation

The command-line interface download package is a .zip file. This single archive file contains everything required regardless of platform you are running from, except for the JRE. The command-line interface requires Java runtime environment version 6 or above, which you must provide yourself. The command-line interface will run on any platform that supports the JRE.

Extract this package with your favorite archive tool into a directory of your choosing. The sub-directory created is called “pure.cli” (pure dot CLI). Under this directory is the “bin” directory, with a sub-directory with the name of the PureApplication System firmware level you downloaded the command-line interface from. Similarly under this directory is the “lib” directory with the firmware specific sub-directory names. Note the “admin” subdirectory here contain the system commands, and the “deployer” sub-directory contains the workload commands.

As you access other racks with different firmware levels, upon connection to the system the command-line interface will automatically download the correct version of the command-line interface required. When this occurs, you will get an informational message indicating the version retrieved. These versioned directories do accumulate over time as they never get automatically deleted.

The bin directory is the directory you need to start the command-line interface interpreter from. Avoid starting the interpreter from any other directory.

There is a “logs” directory which contains a log file to help with diagnostics of failed commands. There also is a “samples” directory that contains many sample scripts for some common functions using the command-line interface.

Section

IBM

Syntax and help

8 Command-line interface © 2012 IBM Corporation

This section covers the various modes you can interact with the command-line interface and the general syntax of the commands and data.

Command-line interface operational modes

1. Execute in interactive mode
   - **pure –h <host> -u <user> -p <password>**
   - Supports command history
   - Supports subset of Emacs cmnds with control-key bindings only (from JLine)
   - Type "**exit**" to exit from interactive mode
2. Execute in single command mode
   - **pure –h <host> -u <user> -p <password> -c "<command>"**
3. Execute in batch mode
   - **pure –h <host> -u <user> -p <password> -f "<filepath>" <script_args>**
   - Python or Jython scripts

For security purposes, do not provide password on the command – it becomes exposed. You will subsequently be prompted for the password to be entered in non-display mode

9 Command-line interface © 2012 IBM Corporation

As stated earlier, there are three different modes in which you can interact with the command-line interface, interactive mode, single command mode and batch mode. All three modes minimally require the host, user ID and password.

Interactive mode requires you to start the interpreter first, which is done when the host, user and password are provided as parameters to the "**pure**" command. Once the interpreter has started, you are free to enter commands interactively. The interactive mode supports command history and a subset of Emacs editor commands. The Emacs bindings come from JLine which is part of Jython, which supports only control-key bindings. Typing "exit" will allow you to exit from interactive mode. .

Single command mode allows you to run a single command provided with the –c parameter. Batch mode allows you to pass in a Python or Jython script and any associated script arguments with the –f parameter.

All three operational modes require the host name or IP address of the PureApplication System, and a valid user ID and password on that system. However, if you provide the password on the command, it becomes exposed. Therefore, good practice is to not provide the password on the command itself. When not provided, you will then get a message indicating an "invalid user name or password was provided", and you will subsequently be presented with a password prompt. At the password prompt, you can then enter the password in a non-display mode and it is not displayed in your command window.

IBM

Command-line interface syntax

- Command-line interface commands are written in Jython syntax with the “dot” operator

    deployer.<object_type>[.<command>(<arguments>)]

    deployer.<object_type>[.<command>[(<data_or_file>)]]

- Some object types require a command method; most do not
- Data passed into commands or displayed from commands is:
  – Passed directly within the command or indirectly with a file
  – Typically in dictionary (Python data type) format

```
[
 {
  “name1”:”value1”,
  “name2”:”value2”,
   ....
 },
 {
  “name1”:”value3”,
  ...
 }
]
```

  – Can also be in list, string or number Python data formats

10 Command-line interface © 2012 IBM Corporation

The command-line interface uses the Jython syntax to construct the commands, which uses the Jython “dot” operator. One always starts a command with either the keyword “deployer” or the keyword “admin”, followed by the dot operator, and then followed by an object type. The next couple of slides will discuss object types in more detail. For now, they are the typical objects within the PureApplication System, such as patterns, virtual applications, databases, users, groups, and so on.

For most object types, if you enter the command at this point with nothing following the object type, all instances of that object type are displayed. An example of this type of object is the environmentprofiles object.

For some object types, you are required to follow the object type with a command method, otherwise an error is generated. An example of this type of object is the environmentprofileclouds object. Typically these command methods are Java methods, and each object type has it’s unique set of methods. As seen on these slides, the help command can help you determine the valid methods for the object type. The command method can require additional arguments, or it can require additional data, provided either instream or from a file.

Most of the data passed to and from commands is in the Python dictionary format, which is similar to JSON. If you are not familiar with the Python dictionary format, then keep a few things in mind when constructing or browsing your dictionary objects. (1) The entire set of dictionary objects are enclosed in square brackets (“[“ and “]”). (2) Each object is enclosed in the curly brackets (“{“ and “}”), and multiple objects are separated by a comma. (3) Each object has one or more name-value pairs called object members. Both the name and value are contained individually within quotation marks and separated by a colon. Each name-value pair is separated by a comma.

Dictionary formatted data can be specified directly on the command-line or placed in a file with the file location being passed to the command. Data can also be exchanged with some commands in some of the other Python data formats, such as the list, string or number formats.

Terms: resource and resource collections (1 of 2)

- "Resource" versus "utility" commands
- For many of the commands, the command-line interface uses the terms:
  - "**Resource collection**" to correlate to a collection of like objects, and
  - "**Resource**" to correlate to an individual object within a resource collection.
- For example:
  - When you are viewing all the virtual system patterns, you are accessing a **resource collection**.
  - When you are viewing or updating one individual virtual system pattern, you are accessing a **resource** within a **resource collection**.
- **Resources** (representing an individual object) and have both **properties** and **methods**
- **Resource collections** (representing the entire collection of one type of **resource**) only have **methods** – they do not have **properties**
- Command results and Intermediate results from nested commands are also either a **resource** or a **resource collection** (or one or more properties)

11 Command-line interface © 2012 IBM Corporation

At one level, commands are categorized as workload commands or system commands. At another level, they are also categorized as "resource commands" versus "utility commands". Utility commands are discussed later in this presentation. Resource commands are discussed here first.

The command-line interface uses these two terms for the resource commands. (1) A "resource collection" correlates to a collection of like objects, such as the collection of all system patterns or the collection of all users. (2) A "resource" correlates to an individual object and all it's properties within a resource collection. For example, all the combined properties of a specific user is a resource. These two terms are valid for all workload console commands and many (but not all) system console commands. Some system console commands are "utility commands" without the concept of resources or resource collections.

There is a further distinction between a resource and a resource collection. A resource has both properties and methods. The properties correlate to the dictionary name-value pairs for the fields within a resource. The methods correlate to "actions" taken against the resource. A resource collection only has methods, which are actions against the resource collection as a whole. It is not possible for a resource collection to have individual properties. Also keep in mind that all command results or command intermediate results from nested commands can be a resource, a resource collection, or one or more properties.

IBM

Terms: resource and resource collections (2 of 2)

- Display of a resource collection

```
>>> admin.volumes
[
{
    ...
    "harddiskdrive": None,
    "id": "19b29d39-1327-4e1b-ae6c-66ef622e0b87",
    "isosvolume": "true",
    "label": "Volume",
    "name": "Shared Vol1",
    ...
}
]
```

- Display of a resource

```
>>> admin.volumes[0]
{
  ...
  "harddiskdrive": None,
  "id": "19b29d39-1327-4e1b-ae6c-66ef622e0b87",
  "isosvolume": "true",
  "label": "Volume",
  "name": "Shared Vol1",
  ...
}
```

 © 2012 IBM Corporation

This slide is providing examples of what a display of a resource collection versus a resource looks like. Note that when displaying a resource collection, it will have the square brackets around the entire result, while the display of a resource will have the curly brackets around the result.

A common mistake made, especially when nesting commands, is to not realize whether the full or intermediate result is a resource or a resource collection, and thus attempt to use the wrong method against the result.

Help introduction

- **>>> help(deployer)** – workload console commands
- **>>> help(admin)** – system console commands
- **Resource related**
  - System and workload console
  - Always use plural object in command
    - deployer.part
    - deployer.parts
    - deployer.database
    - deployer.databases
    - admin.computenode
    - admin.computenodes
- **Utility related**
  - System console only
    - admin.dateandtime
    - admin.snmp
    - admin.dns

13 Command-line interface

© 2012 IBM Corporation

The next three slides provide information about the help facilities in the command-line interface.

When doing the “help deployer” command, as seen in the first bullet in the slide, all associated workload resource objects available to the command-line interface are listed. Similarly, when doing the “help admin” command, as seen in the second bullet, all associated system resource objects and utilities available are listed.

When reviewing these lists, you will see that some of the object types come in singular and plural pairs, such as part and parts, or computenode and computenodes. These are the “resource commands” discussed in the last couple of slides. Note that this singular and plural forms of these resource commands are for the help function only. The actual resource command will always use the plural form for the object.

All the other objects that only come in one form are for the utility commands. These objects have no concept of resources and resource collections. All utility commands are system console commands, each with their own unique format. “Date and time”, “snmp” and “dns” are examples of utility commands.

Help commands

While the information center has doc for all commands, every command also has online help information. This slide shows help information for the three types of commands, namely a resource command, a resource collection command, and a utility command.

The first bullet is an example of a help command for a resource command. It lists the command's available methods and properties.

The second bullet is an example of a help command for a resource collection command. Note that it only lists methods, since there are no properties associated with a resource collection.

The third bullet is an example for a utility command. Note that there are no methods or properties for this particular utility command, but this can vary by utility command.

When reviewing the lists of the command's methods, note that there are two types of methods – those with two underscores around them, and those without the underscores. Those without underscores typically correspond to actions you will see in the administrative console, such as clone, delete, deploy, refresh, and so on. The ones with the underscores around them are typically methods to help you with your scripting, and do not correspond to anything you can do in the administrative console. For example, the "contains" method will return a true or false Boolean value depending on whether a value is "contained" within the object.

Help commands – Methods and properties

Methods:

1. Resource collection:

   >>> help(deployer.applications.__contains__)
   A function that accepts a single argument and returns a Boolean value indicating if the specified resource is contained within this collection.

2. Resource:

   >>> help(deployer.application.download)
   Download the artifact file or the appmodel dict file of a virtual application pattern.

Properties:

3. Resource:

   >>> help(deployer.application.access_rights)
   The access right of the virtual application pattern.
   This value is automatically generated and cannot be changed.

4. Resource collection; has no properties

15 Command-line interface © 2012 IBM Corporation

Further details about methods and properties can be obtained with interactive help commands. Three examples are shown in this slide.

The first help command is for the resource collection called “applications”, and it’s method called “contains”.

The second is for the resource called “application”, and it’s method called “download”.

The third is for the resource called “application” and it’s property called “access_rights”.

The fourth comment is again noting that a resource collection has no properties.

Section

Command examples

 © 2012 IBM Corporation

This section provides various examples of commands that will further demonstrate many of the features of the command-line interface.

Types of command results

IBM

1. >>> admin.groups
```
[
 {
  "created": Feb 20, 2012 9:29:51 PM,
  "description": None,
  "id": 1,
  "name": "Everyone",
  "owner": (nested object),
  "roles": (nested object),
  "updated": Feb 20, 2012 9:29:51 PM,
  "users": (nested object)
 },
 {
  "created": Mar 13, 2012 1:11:31 PM,
  "description": "Auditor Group",
  "id": 3,
  "name": "Auditor_Group",
  "owner": (nested object),
  "roles": (nested object),
  "updated": Mar 13, 2012 1:11:31 PM,
  "users": (nested object)
 }
]
```

CAUTION: (1) indexes can change "on the fly", and (2) IBM does not support any concept of order to the objects with indexes

2. >>> admin.groups[1]
```
{
 "created": Mar 13, 2012 1:11:31 PM,
 "description": "Auditor Group",
 "id": 3,
 "name": "Auditor_Group",
 "owner": (nested object),
 "roles": (nested object),
 "updated": Mar 13, 2012 1:11:31 PM,
 "users": (nested object)
}
```

3. >>> admin.groups[2]
   IndexError: index out of range: 2
4. >>> admin.groups[0].name
   u'Everyone'
5. >>> admin.groups[1].name
   u'Auditor_group'

17 Command-line interface © 2012 IBM Corporation

As stated on a previous slide, all resource related commands use the plural form of the object. For example, as seen on all commands on this slide, "groups" must be used instead of "group". Depending on the command, the command results can be one of three things, a resource collection, a resource, or a property.

The first command on the slide displays the entire resource collection of "groups" on the system, in Python dictionary format. Note the square brackets around the result, indicating the result is a resource collection. In this example there are two groups, named "Everyone" and "Auditor group". The second command on the slide displays one particular instance of the "groups" resource collection, using an index. Indexes start with "0", therefore this command with the index of "1" displays the second instance. Note the curly brackets around the result, indicating this result is a resource. The third command on the slide with the index value of "2" fails because there is no third instance. The fourth command on the slide is displaying the name property of the first instance of the "groups" resource collection. Note that the prefix of "u" for the property value indicates the result is uni-code. This prefix can be ignored as any conversion required between code sets is handled by Jython and is transparent to the user. The fifth command on the slide is displaying the name property of the second instance of the "groups" resource collection.

Note that indexes can change "on the fly" if resources are added or deleted from a resource collection. Therefore use them with caution. Additionally IBM does not support any specific ordering of objects within a resource collection with the use of indexes. Therefore, do not rely on indexes to provide objects in any specific order.

Properties, attributes and nested objects

Properties:

- Attributes: fields with actual values
- Nested objects: fields with complex values
  - Majority of nested objects are relationship based – references to other objects

```
>>> admin.groups[1]
    {
      "created": Mar 13, 2012 1:11:31 PM,
      "description": "Auditor Group",
      "id": 3,
      "name": "Auditor_Group",
      "owner": (nested object),
      "roles": (nested object),
      "updated": Mar 13, 2012 1:11:31 PM,
      "users": (nested object)
    }
```

18 Command-line interface © 2012 IBM Corporation

There typically are two types of properties associated with a resource. .

Some resource properties are attributes with actual values. For example, on this slide, the first two properties in the Auditor Group object displayed are called “created” and “description”, and are attributes with values.

Other properties are nested objects, which have complex values that are not directly displayed. Most nested objects are a reference to another resource or resource collection that this resource has relationships with. For example, on this slide, the properties called “owner”, “roles”, and “users” are nested objects, and thus reflect this resource’s relationships with those other resources or resource collections.

The next slide will show how to display the “users” nested object seen on this slide.

IBM

Properties and nested objects

1. \>>> admin.groups[1].users

```
[
  {
    "addons": (nested object),
    ...
    "fullname": "Auditor id",
    "groups": (nested object),
    ...
    "username": "Auditor_01",
    ...
  }
]
```

2. \>>> admin.groups[1].users[0].groups

```
[
  {
    "created": Feb 27, 2012 11:30:40 AM,
    "description": None,
    "id": 1,
    "name": "Everyone",
    "owner": (nested object),
    "roles": (nested object),
    "updated": Feb 27, 2012 11:30:40 AM,
    "users": (nested object)
  },
  {
    "created": Jan 12, 2012 3:09:58 PM,
    "description": "Auditor group test03",
    "id": 2,
    "name": "Auditor_group",
    "owner": (nested object),
    "roles": (nested object),
    "updated": Jan 18, 2012 5:15:06 PM,
    "users": (nested object)
  }
]
```

 © 2012 IBM Corporation

This slide demonstrates some commands that make use of nested objects and nested commands.

The first command is invoked against the second instance of the groups object that was seen on the previous slide, and is displaying the nested object called “users”. The result is a resource collection of all the users that are in this particular group. In this case, there is only one user in the group, called Auditor.

Note that this command result also has a nested object property called “groups”. This nested object can now be displayed with a nested command, as shown in the second command above. The result is a resource collection that shows the “auditor id” user is contained in two different groups, namely the everyone group and the auditor group.

One can do this type of command nesting indefinitely within a command.

Assigning values to attributes

Assignment of a value to a resource attribute:

1. Display the **"description"** attribute
   >>> admin.groups[0].description
      u'Auditor group'
2. Assign a value to the same attribute
   >>> admin.groups[0].description = "Auditor group test01"
3. Display the updated value of the same attribute
   >>> admin.groups[0].description
      u'Auditor group test01'

© 2012 IBM Corporation

This slide shows a simple assignment of a value to a resource attribute. The first command is displaying the current "description" attribute of the first instance of the "groups" resource collection, in order to show it's original value. The second command is then modifying that value of the "description" attribute just displayed. The third command is displaying it again to prove the change was made.

The change is immediately reflected in the administrative console by pulling up this group or by refreshing the screen if you are already displaying the group.

IBM

Using variables - With a resource attribute

1. Assignment of a resource attribute value to a variable
   >>> MyGroupDescription = admin.groups[0].description
   >>> MyGroupDescription
   u'Auditor group test01'
   >>> admin.groups[0].description
   u'Auditor group test01'
2. Results of updating the resource attribute
   >>> admin.groups[0].description = "Auditor group test02"
   >>> MyGroupDescription
   u'Auditor group test01'
   >>> admin.groups[0].description
   u'Auditor group test02'
3. Results of updating the variable
   >>> MyGroupDescription = "Auditor group test03"
   >>> MyGroupDescription
   u'Auditor group test03'
   >>> admin.groups[0].description
   u'Auditor group test02'

21 Command-line interface © 2012 IBM Corporation

The next set of slides discuss the use of Python variables, and will describe some of the variations of how variables are handled in the command-line interface when they are used to represent attributes, resources or resource collections. This slide describes the first case, the assignment of a resource attribute to a variable.

The first set of three commands is assigning a resource attribute to a variable. The three commands are (1) assigning the “description” attribute to a Python variable called MyGroupDescription, (2) displaying the variable, and (3) displaying the original server-side attribute. Both displays show the same values.

The second set of three commands is modifying the original server-side attribute, and displaying the result from both the server-side and the local variable perspective. Note that updating the server-side resource attribute did not update the variable.

The third set of three commands is modifying the local variable, and displaying the result from both the server-side and the local variable perspective. Note that updating the local variable did not update the server-side resource attribute.

What is demonstrated here is that when the assignment of a resource attribute to a variable is made, there is no further relationship between the variable and the original attribute in the resource. This is expected with the typical definition of a Python variable being a reserved memory location to store a value.

IBM

Using variables - With a resource (1 of 2)

1. Assignment of a resource to a variable:

```
>>> MyGroup = admin.groups[0]
>>> MyGroup
    {
     "created": Jan 12, 2012 3:09:58 PM,
     "description": "Auditor group test01",
     "id": 2,
     "name": "Auditor_group",
     "owner": (nested object),
     "roles": (nested object),
     "updated": Jan 18, 2012 5:06:02 PM,
     "users": (nested object)
    }
```

2. Results of updating the attribute in the server-side resource:

```
>>> admin.groups[0].description = "Auditor group test02“
>>> MyGroup.description
    u'Auditor group test01'
>>> admin.groups[0].description
    u'Auditor group test02'
```

3. Results of updating the attribute in the variable:

```
>>> MyGroup.description = "Auditor group test03"
>>> MyGroup.description
    u'Auditor group test03'
>>> admin.groups[0].description
    u'Auditor group test03'
```

 © 2012 IBM Corporation

This slide describes the assignment of a resource to a variable.

The first set of two commands is assigning an entire resource to a Python variable called MyGroup, and displaying the variable. The display of the variable MyGroup does show that it represents the entire resource.

The second set of three commands is modifying the original server-side attribute, and displaying the result from both the server-side and the local variable perspective. Note that updating the server-side resource attribute did not update the variable.

The third set of three commands is modifying the local variable, and displaying the result from both the server-side and the local variable perspective. Note that updating the local variable did also update the server-side resource attribute, which is not what you saw on the previous slide. The next slide explains why there is this difference.

Using variables - With a resource (2 of 2)

- A variable assigned a resource becomes a reference to the physical server-side resource
- The command-line interface locally caches the results of resource commands (and only resource commands, not attribute or resource collection commands)
- If a variable attribute is updated:
  – The variable references the server-side resource,
  – The server-side attribute gets updated
  – Therefore the local cache also gets updated
  – Therefore viewing both the variable attribute and the server-side attribute provides the same result
- If a server-side attribute is updated:
  – Only the local cache is updated
  – Since the variable is referencing the actual server-side resource, viewing the variable attribute will NOT see the update
  – If the variable is manually refreshed with the refresh() method, then viewing the variable attribute will see the update

23 Command-line interface © 2012 IBM Corporation

This difference of behavior for the updates of variables on the previous slide is explained as follows. When a resource is assigned to a variable, the command-line interface will do two things. (1) It uses the variable as a reference to the actual server-side resource, as opposed to storing the contents of the resource in a separate memory location. (2) When a resource object is first accessed from the command-line interface, it's attributes are cached locally. If updates are made to the resource, the cache is refreshed.

Therefore, when the variable attribute is updated, the variable references the physical server-side resource, and that attribute is updated and the local cache is updated. This results in both the variable attribute and the server-side attribute having the same value.

However, when the server-side resource attribute is updated, only the local cache is updated, and thus displaying the variable attribute will not show the updated value. To see the updated value, the variable needs to be manually refreshed with the refresh method.

IBM

Using variables - With a resource collection (1 of 2)

1. Assignment of a resource collection to a variable

```
>>> MyGroups = admin.groups
>>> MyGroups
[
 {
  "created": Feb 20, 2012 9:29:51 PM,
  "description": None,
  "id": 1,
  "name": "Everyone",
  "owner": (nested object),
  "roles": (nested object),
  "updated": Feb 20, 2012 9:29:51 PM,
  "users": (nested object)
 },
 {
  "created": Mar 13, 2012 1:11:31 PM,
  "description": "Auditor Group",
  "id": 3,
  "name": "Auditor_Group",
  "owner": (nested object),
  "roles": (nested object),
  "updated": Mar 13, 2012 1:11:31 PM,
  "users": (nested object)
 }
]
```


© 2012 IBM Corporation

The next two slides describe the assignment of a resource collection to a variable.

The set of two commands shown on this slide are assigning an entire resource collection to a Python variable called MyGroups, and then displaying the variable, which shows that the variable does represent the entire resource collection.

The next slide continues with this example.

Using variables - With a resource collection (2 of 2)

2. Results of updating the attribute in the resource collection:

```
>>> MyGroups[0].description = "Auditor group test03"
>>> MyGroups[0].description
u'Auditor group test03'
>>> admin.groups[0].description
u'Auditor group test03'
```

3. Results of updating the attribute in the variable:

```
>>> admin.groups[0].description = "Auditor group test02"
>>> MyGroups[0].description
u'Auditor group test02'
>>> admin.groups[0].description
u'Auditor group test02'
```

Here, the variable is a reference to the physical server-side resource collection, and the CLI does NOT cache command results to a resource collection. Therefore updates made to either the variable or to the server-side object are updating the same thing

© 2012 IBM Corporation

The second set of three commands is modifying the original server-side attribute, and displaying the result from both the server-side and the local variable perspective. Note that updating the server-side resource attribute did also update the variable.

The third set of three commands is modifying the local variable, and displaying the result from both the server-side and the local variable perspective. Note that updating the local variable did also update the server-side resource attribute. These results for a resource collection assignment are again different from what was seen for an attribute assignment and a resource assignment on the previous slides.

This difference is explained as follows. Similar to a resource assignment to a variable, for a resource collection assignment to a variable, the command-line interface uses the variable as a reference to the actual server-side resource collection. However, the difference here from what you saw with a resource assignment is that, for a resource collection, the command-line interface does not cache the attributes locally. This is because resource collections do not have attributes, only resources do. Therefore, in this scenario with a resource collection, it doesn't matter whether you update the attribute through the variable or through the server-side resource collection, you are updating the same physical entity on the server.

IBM

Using variables - Summary

1. Assign a resource attribute to a local variable:
   – Updating the attribute in the local variable DOES NOT update the "original"
   – Updating the attribute in the "original" DOES NOT update the local variable.
2. Assign a resource to a local variable::
   – Updating the attribute in the local variable DOES update the "original"
   – Updating the attribute in the "original" DOES NOT update the local variable, until the variable is refreshed with the refresh() method.
3. Assign a resource collection to a local variable:
   – Updating the attribute in the local variable DOES update the "original"
   – Updating the attribute in the "original" DOES update the local variable.

© 2012 IBM Corporation

This slide summarizes the three scenarios of variable assignments and updates seen in the past five slides.

For the assignment of a resource attribute to a local variable, updates to the variable and to the server-side resource attribute are isolated from each other.

For the assignment of a resource to a local variable, updates to the server-side resource is isolated from the variable until a refresh is done to the variable. Conversely, updates to the variable are not isolated from the server-side resource.

For the assignment of a resource collection to a local variable, updates to either the variable or the server-side resource collection are not isolated from each other.

IBM

Command examples – virtual application pattern deployment

```
1. >>> deployer.applications[16].deploy("Test Deploy",admin.environmentprofiles[0])
     {
       "acl": (nested object),
       "app_id": "a-82d61d10-1523-4786-bf5e-d01e1a1a6f54",
       …
       "deployment_name": "Test Deploy",
       "id": "d-12bfd03b-b6e1-41e5-b352-737d1c191106",
       …
       "status": "LAUNCHING",
       …
     }
2. >>> deployer.virtualapplications[0].status
     u'LAUNCHING'
3. >>> deployer.virtualapplications.get("d-12bfd03b-b6e1-41e5-b352-
   737d1c191106").status
     u'LAUNCHING'
4. >>>deployer.virtualapplications.list({"deployment_name":"Test Deploy"})[0].status
     u'LAUNCHING'
```

27 Command-line interface © 2012 IBM Corporation

This slide demonstrates a deployment of a virtual application pattern to an environment profile with the use of the deploy command method. In the first command, the virtual application pattern resource collection is indexed to access the seventeenth one, hence the index of sixteen. The index value was determined by previously displaying all the application patterns and manually looking for it. Again, use caution with indexes, as they can quickly change.

The deploy method requires two parameters, namely the deployed application name and the environment profile. The deployed application name is free-form text within quotation marks. In this example, the environment profile name is resolved by nesting a command within a command to derive an argument result. Here, the first instance of the environmentprofiles resource collection intermediate result is used as the environment profile name. The result of the deploy command displays the resulting "virtual application" resource as opposed to the "application pattern" resource that was deployed.

The second command is displaying the status of the deploy. Here you first need to find out what index your deploying virtual application correlates to, and use that index in the command. In this case it was the first one, thus requiring an index of zero. However, if you or anyone else subsequently initiates a second deploy, the index will most likely change, so use caution with indexes.

The third and fourth examples on this slide demonstrate some alternatives to using indexes to access a particular resource within a resource collection. The third command is using the get command method with the "id" property of the resource. Note that the get method does not require the data to be in dictionary format; instead it uses a "key". Keys are not available for every object type. The fourth command is using the list method to find resources within a resource collection that have a property (or multiple properties) of a specific value, in dictionary format. Here, the property is the deployment_name and the value is "Test Deploy".

Command examples - Miscellaneous

1.deployer.applications.get(<app_id>).clone(<app_name>, <app_type>)

2.deployer.plugins.getMeta(<plugin name>)

3.deployer.patterntypes.get(<shortname>, <version>).acceptLicense()

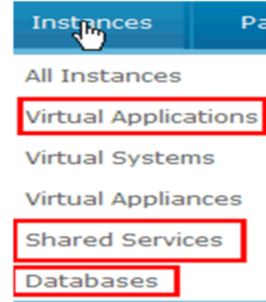
4.deployer.virtualimages[“Portal”]

5.deployer.virtualapplications.terminate(<depl_id>)

6.deployer.virtualapplications.get(<depl_id>).shareuser(<user name>,<access rights>)

7.deployer.applications.list({'app_type':'service'})

8.deployer.virtualapplications. list({“app_type”:”database”})



CLI virtualapplications types:

"app_type": "application"

"app_type": "service"

"app_type": "database"

28 Command-line interface © 2012 IBM Corporation

This slide provides examples of commands for various actions that you can typically do from the administrative console.

Example 1 is cloning a virtual application pattern with a particular application ID.

Example 2 is displaying the metadata for a plug-in with a particular plug-in name.

Example 3 is accepting the license for a pattern type.

Example 4 is listing all virtual images that have the text “Portal” embedded in the name property.

Example 5 is terminating a deployed virtual application with a particular deployment ID.

Example 6 is giving a particular user access to a deployed virtual application.

Example 7 is listing all application patterns with an application type of “service”.

Example 8 is displaying all deployed database instances.

Note in example 7 that, to the command-line interface, database patterns, shared services patterns, and web application patterns are three types of a virtual application pattern. Similarly, in example 8, all deployed databases, shared services, and web applications are three types of a virtual application instance. These are further examples of there not always being a 1-to-1 correlation between what the administrative console provides versus the command-line interface.

![slide]

Command examples - Create options

1. **>>> admin.users.create(deployer.wizard)** - (use "!" to break out)

```
Enter ?? for help using the wizard.
username: Sam
fullname: Sam Bush
password (optional): password
email: sam@us.ibm.com
{
  "addons": (nested object),
 …
  "deploymentoptions": 0,
  "email": "sam@us.ibm.com",
  "environmentprofiles": (nested object),
  "fullname": "Sam Bush",
…
```

2. **>>> admin.users.create([{"username":"Martha","fullname":"Martha White","password":"password","email":"mw@us.ibm.com"}])**

3. **>>> admin.users.create("<local file path>")**

 © 2012 IBM Corporation

For objects that have the create method to create a new resource within a resource collection, one can use either the wizard class or the manual input format.

The first command on this slide shows the wizard format, where you are prompted for the value of each property individually. If you need to break out of the wizard without creating the object, use the exclamation mark.

The second and third commands show the manual input formats. In the second example, the data is provided in Python dictionary format as part of the command itself. In the third example, the data is provided in Python dictionary format within an input file.

Exports and imports of virtual system patterns

- Exports and imports of virtual system patterns can only be done with the command-line interface in batch mode.
- Script package archives are downloaded by default
- Optionally add-ons and images can be downloaded

1. Virtual systems pattern export (batch mode):

pure –h <host> -u <user> -f .../samples/exportPattern.py -p "My Pattern" -t "c://temp/My Pattern.zip“

- By default, downloads all script packages
- **“--downloadAll”** parm also downloads all add-ons and images
- **“-d <filter_file>** allows filtered download of scripts, add-ons and images

2. Virtual system pattern import (batch mode only):

> pure –h <host> -u <user> -f .../samples/importPatterns.py -s "c://temp/My Pattern.zip”

30 Command-line interface

© 2012 IBM Corporation

For web application patterns, there are options in the administration console that allow you to export and import that type of pattern. However, for database patterns and virtual system patterns, you need to do the pattern exports and imports with the command-line interface. This slide provides the commands for exporting a virtual system pattern, to be run in batch mode.

Virtual system patterns are composed of script packages, add-ons and images. By default, the export command will only download all script packages. There is an option to also download all add-ons and images. There is also an option to filter the script packages, add-ons and images to be downloaded.

The first command in the slide is exporting a virtual system pattern in interactive mode using the “exportPattern” script. Information is also provided in the slide on the parameters to download all add-ons and images, and to filter the components to be downloaded.

The second command is importing the previously exported virtual system pattern, using the “importPatterns” script.

Exports and imports of database patterns

Exports and imports of database patterns can only be done with the command-line interface, in interactive mode.

1. DBaaS pattern export:

>>> deployer.applications.list({"app_name":"My DBaaS"})[0].download("c://temp/My DBaaS.zip")

2. DBaaS pattern import:

>>> deployer.applications.create("c://temp/My DBaaS.zip")

 © 2012 IBM Corporation

This slide provides the commands for exporting a web application pattern, to be run in interactive mode.

The first command is exporting a database pattern.

The second command is importing the previously exported database pattern.

IBM

Workload troubleshooting sample commands

1. Download all the log files:
   – **>>> deployer.diagnostics.get('c://temp/diagnostics.zip')**
2. Display the last 20 lines of the error log:
   – **>>> deployer.errors.tail(20)**
3. Display the last 1000 lines of the trace file and print to a file (in command mode):
   – **>>> pure –h <host> -u <user> -p <password> -c “deployer.trace.tail(1000)" >c://temp/trace1000.txt**

© 2012 IBM Corporation

This slide gives some examples of workload related troubleshooting commands.

The first command provides a full set of workload troubleshooting logs. If a destination file is not provided, the logs are downloaded to the directory you run the CLI from.

The second command displays the last twenty lines of the error log.

The third command is run in command mode, and will write the last one thousand lines of the trace file to a file on your workstation.

IBM

System command examples

Some examples of system commands:

1.>>> **admin.leds.list({'severity':'error'})**

2.>>> **admin.events.list({'category':'Alert'})**

3.>>> **admin.tracesettings.list({'component':'Authorize'})**

4.>>> **admin.systemlogs[0]**

5.>>> **admin.storagedevices[0].statistics**

6.>>> **admin.computenodes.list({'state':'failed'})**

© 2012 IBM Corporation

This slide is giving some basic examples of system console commands.

The first command is listing all LEDS with a severity of “error”.

The second command is listing all events with a category of “alert”.

The third command is listing the trace settings for the component called Authorize.

The fourth command is displaying information about the latest collection set of system logs.

The fifth command is displaying the statistics for one of the two storage devices.

The sixth command is displaying all compute nodes that are in a failed state.

IBM

Some things to watch out for

1. Python uses the backslash "\" as an escape character, so only use forward slashes when providing file names.
2. Python uses the JLine support for Emacs, and support is somewhat limited on Windows; Linux® works better with the command-line interface. On Windows, the backspace, down and up arrows, and cursor placement can be not as you expect, especially for commands that span multiple lines.
3. All methods must be invoked with a set of parentheses. If the method has no arguments, you still need the parentheses suffixed to the method: **deployer.virtualsystems[2].stop().**
4. Every time a method is called on an object, Python inserts a reference to the object itself as the first argument. Therefore an error message can state something like "two arguments required, but three were given", whereas only one is actually required.

```
>>> deployer.applications[0].download("c://temp/vp.zip","temp")
TypeError: download() takes exactly 2 arguments (3 given)
>>> deployer.applications[0].download("c://temp/vp.zip")
>>>
```

34 Command-line interface © 2012 IBM Corporation

This slide summarizes some common things that can cause problems that are difficult to resolve until you understand the issue.

The first topic relates to Python using the backslash character as an escape character, therefore if a backslash is used in the wrong place and interpreted as an escape character, you can get unpredictable errors. Therefore only use forward slashes.

The second topic relates to Python's JLine support, which is limited on Windows, but does work quite well on Linux. Therefore if you are using Windows, you might have some challenges, especially with long multi-line commands.

The third topic is a reminder that all command methods must be invoked with a set of parentheses, even when there are no arguments.

The fourth topic relates to Python's usage of references to objects whenever a command method is invoked. Python will insert that reference to the object as the first argument in the method. Therefore, even if you do not supply any arguments to the method, there is one hidden argument being passed. Therefore the number of arguments passed is always one greater then what you provide. This can cause confusion with some error messages that indicate the method is expecting one more argument then you yourself are providing.

IBM

Summary

- There are three ways to interact with the PureApplication System
  - Administrative console
  - REST APIs
  - Command-line interface
- Setting up an operational command-line interface environment is a two minute activity
  - Download and extract
- The command-line interface can be run in three modes: interactive, command and batch modes.
- Commands are divided into workload console commands using the **deployer.*** form, and system console commands using the **admin.*** form.
- On another level, the commands are segregated into "resource" commands and "utility" commands.
- The command-line interface syntax is robust and flexible, and allows for full scripting of most functions.
- This presentation listed a very small subset of available command-line interface operations
  - For a complete list see the PureApplication System information center.

 © 2012 IBM Corporation

In summary.

There are three ways to interact with the PureApplication System. They are the administrative console, REST APIs and the command-line interface. The focus in this presentation is on the command-line interface.

Getting a command-line interface environment working is a matter of a simple download from the PureApplication System welcome page and extracting the archive file to your workstation.

The command-line interface can be run in three modes, namely in interactive, command and batch modes.

The commands are divided into workload commands and system commands. On another level, they are divided into resource command and utility commands.

If you are looking to script and automate PureApplication System tasks, the command-line interface is a great option. There are numerous samples in the tool's download directory to get you started and to help you understand how to script your tasks.

This presentation went through a very small subset of the available commands and syntax, but hopefully it demonstrated to you how robust and flexible the command-line interface package is.

Resources

Jython doc: http://wiki.python.org/jython/

Python doc: http://wiki.python.org/moin/

 © 2012 IBM Corporation

This slide provides some links to further doc about Jython and Python.

Trademarks, disclaimer, and copyright information

IBM, the IBM logo, ibm.com, PureApplication, and System i are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Windows, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.

© Copyright International Business Machines Corporation 2012. All rights reserved.