DPBuddy User Guide

Version 3.1

Contents

Introduction 8

DPBuddy Installation and Configuration 9

DPBuddy Installation 9

Upgrading from DPBuddy 3.0 10

Upgrading from DPBuddy 2.3 10

Configuring DataPower to Work with DPBuddy 10

Configuring Apache Ant to Work with DPBuddy 11

Verifying Connectivity with DataPower 12

Advanced Configuration 12

DPBuddy License 13

DPBuddy's Command-Line Interface (CLI) 13

Using DPBuddy with Apache Ant 15

Enabling Verbose Logging 15

Recommended Development and Build/Deployment Process 16

Developing DataPower Artifacts 16

Creating a Build in a Test Environment 16

Promoting the Build to a "Higher" Environment 17

Deploying to a Production Environment 18

Common Attributes/Options and Properties 18

DataPower Connection Attributes/Options 18

Environment Attributes/Options 19

Schema Validation Attributes/Options 20

Examples 21

"Save" Attribute/Option 21
License File Attribute/Option 22
Use or Regular Expressions to Select DataPower Objects 22
Examples 23
Using DPBuddy to Transform DataPower Configuration and XML Files 23
"transform" Type 25
Attributes Common to All Transform Actions 25
"setText" Transform Action 26
"replaceText" Transform Action 26
"add" and "update" Transform Actions 26
"delete" Transform Actions 26
"dpExclude" and "dpInclude" Transform Actions 27
"namespace" Nested Element 27
Managing Environment-Specific Properties/Variables 27
"env" Attribute/Option and "environment" Task 28
Attributes of the "environment" Task 29
Examples 29
Auditing DPBuddy Tasks/Commands 30
Properties for Configuring Audit 31
Tasks/Commands for Changing and Exporting Configuration 33
"import"/"buddyImport" 33
Attributes/Options 33
Examples 36
"transformFiles" Nested Element 36
"transform" Nested Element 37
"modifyConfig" 37
Attributes/Option 38

"transform" Nested Element 38
"fileset" Nested Element 38
"configuration" Nested Element 39
Examples 39
"delConfig" 39
Attributes/Options 40
"object" Nested Element 40
Examples 40
"export" 40
Attributes/Options 41
"exportObject" Nested Element 43
"transform" Nested Element 44
"namePattern" Nested Element 44
Examples 44
"save" 45
Examples 45
Tasks/Commands for Working with Files and Directories 45
"copy" 45
Attributes/Options 45
"dpFileset" Nested Element 47
"transform" Nested Element 47
Examples 47
"delete" 48
Attributes/Options 48
Examples 49
"mkdir" 49
Attributes/Options 49
"dir" Nested Element 49

Examples 50
"rmdir" 50
Attributes/Options 50
"dir" Nested Element 50
Examples 50
"download" 50
Attributes/Options 51
Examples 51
"downloadFile" 52
Attributes/Options 52
Examples 52
Tasks/Commands for Dealing with Configuration Checkpoints 52
"checkpoint" 52
Attributes/Options 53
Examples 53
"rollback" 53
Attributes/Options 53
Examples 53
"delCheckpoint" 53
Attributes/Options 54
Examples 54
Tasks/Commands for Quiescence/Un-quiescence 54
"quiesce" and "unquiesce" 54
Attributes/Options 54
"object" Nested Element 55
Examples 55
"quiesceDomain" and "unquiesceDomain" 55

Examples 56
Tasks/Commands for Checking DataPower Status 56
"status" 56
Examples 56
"serviceStatus" 56
Attributes/Options 57
Examples 57
"assertStatus" 57
Attributes/Options 58
Examples 58
"assertFreeSpace" 58
Attributes/Options 58
Examples 58
"assertState" 59
Attributes/Options 59
"object" Nested Element 59
Examples 60
"assertActiveService" 60
"object" Nested Element 60
Examples 60
"assertOpenPorts" 60
Attributes/Options 61
Examples 61
Tasks/Commands for Working with DataPower Logs 61
"tailLog" 61
Attributes/Options 61
Log Entry Format 63

"where" Nested Element 63
"domainPattern" Nested Element 64
Examples 64
"log" 65
Attributes/Options 65
Examples 66
"setLogLevel" 66
Attributes/Options 66
Examples 66
Tasks/Commands for Flushing DataPower Cache 66
"flushStylesheetCache"/"flushDocumentCache"/"flushXMLCache" 66
Attributes/Options 67
Examples 67
"flushAAACache" 67
Attributes/Options 67
Examples 67
"flushMiscCache" 67
Attributes/Options 68
"flushAllCache" 68
Backup Tasks/Commands 68
"backup" 68
Attributes/Options 68
"domainPattern" Nested Element 70
Examples 70
"secureBackup" 70
Attributes/Options 70
Examples 71

Tasks/Commands for Resetting/Restarting Domains 71
"restartDomain" 72
Example 72
"resetDomain" 72
Example 72
Miscellaneous Tasks/Commands 72
"action" 72
Attributes/Options 73
Examples 73
"somaRequest" 73
Attributes/Options 73
Examples 73
"wsrrSynchronize" 74
Attributes/Options 74
Examples 74
"testConnection" 74

Introduction

DataPower Buddy ("DPBuddy") is a tool for automating administration and management of IBM WebSphere DataPower appliances. The tool supports configuration export, configuration import, file transfer, backups and many other functions, as described in this document.

Each of these functions is implemented as a command (a.k.a. command-line interface - CLI) that can be run directly from command line/shell and as a custom task for the Apache Ant build tool.

For example, to import a domain, you can run the following command:

```
dpbuddy import -file domain.zip -assertObjectsUp -save
```

This command will import the configuration and files contained in "domain.zip", verify that all objects and services are up and running after the import and save the domain's configuration.

To implement the same functionality in an Ant build script, you could create an Ant target containing the "import" task (all DPBuddy tasks have "dp:" XML namespace prefix):

```
<target name="import.domain" >
    <dp:import file="domain.zip" assertObjectsUp="true" save="true"  />
</target>
```

You can find many more examples in your DPBuddy distribution under "samples" or online. The directories "samples/cli-shell" and "samples/cli-win" contain examples of commands. The "samples/ant-tasks" folder contains examples of Ant tasks. You can also look at "samples/ant-end-to-end" and dp-build.deploy.xml, which implements an end-to-end build/deploy process using DPBuddy.

Internally DPBuddy uses the DataPower SOAP interface a.k.a. SOAP Configuration Management ("SOMA") for all of its functions. You can find more details about SOMA in DataPower InfoCenter or in this redbook, but no knowledge of SOMA is required to be able to use DPBuddy. Reliance on SOMA means that DPBuddy runs all of its commands remotely over HTTPS using the port dedicated to the XML management interface. You can find this port under Network/Management/XML Management Interface in DataPower WebGUI.

DPBuddy Installation and Configuration

DPBuddy Installation

DPBuddy requires Java SE Runtime Environment 6 or better. All JVM types are supported, including Oracle, IBM and OpenJDK. DPBuddy is pure Java, so it runs on any OS where Java is available. In some cases, DPBuddy may require a large heap size so a 64-bit JRE is recommended.

Both DPBuddy and Apache Ant will first check if the JAVA_HOME environment variable is defined. If it is defined, this Java installation will be used. If JAVA_HOME is not defined, then java executable must be added to the path. To verify, run the "java -version" command.

Install DPBuddy by following these simple steps:

- Download the distribution from myarch.com/dpbuddy and unzip/untar the content of the file into a directory (tar -xvf on Linux).
- Define the DPBUDDY_HOME environment variable and point it to the directory containing DPBuddy installation, e.g., /home/dpbuddy/dpbuddy-3.1.
- Add %DPBUDDY_HOME%\bin or $DPBUDDY_HOME/bin to your PATH environment variable.
- Open a command line/terminal window and type "dpbuddy -h". You should see a help screen containing the list of commands supported by the tool.

If you already have a previous version of DPBuddy installed, you will need to update your "taskdef" Ant tag to point to the location of the new version.

Upgrading from DPBuddy 3.0

The DPBuddy 3.1 distribution contains Apache Ant 1.9.3 jar files. To avoid classloader conflicts, exclude Ant jars from the taskdef used to define DPBuddy tasks:

```
<taskdef uri="antlib:com.myarch.dpbuddy">
    <classpath>
        <fileset dir="${dpbuddy.home}/lib" excludes="ant*"/>
    </classpath>
</taskdef>
```

All DPBuddy 3.1 tasks are backward compatible with 3.0.

Upgrading from DPBuddy 2.3

DPBuddy 3.1 is mostly backward compatible with DPBuddy 2.3 with a few exceptions:

- The namespace for the tasks has changed to "com.myarch.dpbuddy". Make sure to update it in the "project" tag and in the "taskdef".
- The "request" task was renamed to somaRequest.
- The "setConfig" task was deprecated; use import instead.
- modifyConfig no longer supports the nested "configFile" element, you can replace it with the new transform element.
- "toDir" is now a required attribute of the copy task. You can also provide "toDir" at the nested "dpFileset" level.
- Environment prefix mechanism has been substantially enhanced; you may need to change the way your properties are named to take advantage of the hierarchical prefixes.

Configuring DataPower to Work with DPBuddy

You need to make sure that you have XML Management enabled in all devices that will be managed by DPBuddy. You can do this from DataPower WebGUI (Network/Management/XML Management).

DPBuddy uses the "old" DataPower v2004 message format for file transfer (the 2004 schema is more efficient, since it allows for uploading multiple files at once). This means that the "SOAP Configuration Management (v2004)" checkbox must be checked on the XML management screen.

Be careful performing these changes on the appliances residing in a production DMZ -- make sure that the management ports are not available to the outside world.

We also recommend enabling internal logging as documented in the DataPower Info Center. DataPower does not return error messages to the client. If a DPBuddy command fails and you

get "Internal Error" from the device, you would need to look for the root cause in the DataPower system log in the default domain. DataPower captures the error only if internal logging is enabled. You can enable it by navigating to "Troubleshooting" from the control panel, then to "Logging." Set "Enable Internal Logging" and click on "Set Log Level." We recommend setting the level to "Warning", since anything more detailed (e.g., "Information") results in needlessly verbose messages.

If you're creating a new account that will be used by DPBuddy, make sure that the password for this account has been reset. Before running DPBuddy, make sure that you're able to login to WebGUI using this account. The account must be a "privileged" account.

Note that by default DataPower uses a self-signed certificate for SSL. By default, DPBuddy ignores certificate validation errors. You can change this behavior by setting "dp.trust.all.certs" to "false" as described in Common Attributes/Options and Properties. In this case, you will need to manually add the DataPower certificate to your JDK's keystore.

It is highly recommended (although not absolutely required) to place all machines hosting DPBuddy and all DataPower appliances in the same time zone. DataPower relies on ZIP format for export/import; this means that file timestamps have no time zone information. This may lead to various issues if an export/import archive needs to be modified during your build/deployment process.

Configuring Apache Ant to Work with DPBuddy

Using DPBuddy from Apache Ant provides more flexibility and better integration with various build/deployment and release management tools than DPBuddy's CLI. It is also more efficient since there is no need to start a JVM for every command.

DPBuddy requires Apache Ant 1.8 or above.

If you don't have Apache Ant installed on your system, download and install it as documented at http://ant.apache.org/manual/install.html. Note that on many Linux systems you can also install Ant using a package manager, e.g., you can use "apt-get -u install ant" on Ubuntu.
We also recommend running Ant with the "-e" option, as it provides a much cleaner output. You can set this option globally by setting the ANT_ARGS environment variable to "-e".

DPBuddy tasks are packaged in a single Antlib. You may wish to familiarize yourself with the Ant documentation of the Antlib feature, especially the use of namespaces in Ant, before configuring DPBuddy tasks in your build file.

To declare DPBuddy tasks, add "taskdef" to your build file and point it to the DPBuddy installation:

```
<property name="dpbuddy.home" location="your location" />
<taskdef uri="antlib:com.myarch.dpbuddy">
    <classpath>
        <!-- We need to exclude Ant jars packages with DPBuddy -->
        <fileset dir="${dpbuddy.home}/lib" excludes="ant-*"/>
    </classpath>
</taskdef>
```

You must also declare the namespace used by DPBuddy tasks:

```
<project name="your project" xmlns:dp="antlib:com.myarch.dpbuddy" >
```

The namespace's URI must match the URI used in taskdef. You must use "antlib:com.myarch.dpbuddy" as the URI.

You can also copy all the jar files that come with DPBuddy to ${user.home}/.ant/lib directory. If you choose this option, you don't need to provide an explicit "taskdef". Please consult the Ant manual for more details.

Verifying Connectivity with DataPower

To verify DPBuddy's connectivity with DataPower, run the following command:

```
dpbuddy testConnection -url <your DataPower hostname> -user <your
DataPower username> -password <your DataPower password> -domain default
```

Or from Ant:

```
<target name="test.connection" >
    <dp:testConnection url="dp-local" user="dpbuddy"
password="dpbuddy01" domain="default"/>
</target>
```

Advanced Configuration

If you're planning to use DPBuddy to download or upload large binary files (bigger than 30MB), we recommend increasing the maximum heap size for the JVM used to run DPBuddy.

DPBuddy CLI supports the DPBUDDY_OPTS environment variable, which can be used to pass custom options to the JVM. By default, this variable is set to "-Xms256m -Xmx1024m", meaning that the maximum heap size is set to 1GB.

To change the heap size for Ant, set the "ANT_OPTS" environment variable.

"-Xmx1648m" is the recommended setting for both Ant and DPBuddy CLI.

If you're going to run DPBuddy behind a proxy, add the following Java system properties to the "DPBUDDY_OPTS" or "ANT_OPTS" variables: -Dhttps.proxyHost=<your proxy host> -Dhttps.proxyPort=<your proxy port> (note "https" prefix), or use Ant's setProxy task.

DPBuddy License

All DPBuddy downloads are bundled with a 60-day evaluation license. You do not need to request or install the evaluation license; it is generated automatically when you run DPBuddy for the first time.

Once the evaluation license is expired, you'll need to purchase the license from MyArch or request an extension of your trial license. Please send your request to license@myarch.com or submit it at http://myarch.com/dpbuddy-license.

To install a permanent license that you received from us, simply save the license file and add the property "dpbuddy.license.file" to your configuration file or Ant build file. This property must point to the location of the license file. Alternatively, you can provide the "license" option or attribute it to a DPBuddy command or task.

DPBuddy's Command-Line Interface (CLI)

All DPBuddy commands have the following format:

```
dpbuddy <command> <options>
```

"command" is any of the commands/tasks described in the document. The command (or "-h") must be the first argument.

In this document the words "task" and "command" are used interchangeably. Every DPBuddy command has a corresponding Ant task with the same name.

The options for every command are listed under "Options/Attributes". Each option must be preceded by "-". In this document, "option" and "attribute" are also used interchangeably; every command line option has a corresponding Ant attribute and vice versa.

For example, the "import" command supports the "file" option. To run the command, type "`dpbuddy import -file domain.zip`".

The value is optional for Boolean options. The Boolean options always default to "true". For example, to run "import" in the dry-run mode, supply "-dry-run" option:

```
dpbuddy import -file domain.zip -dry-run
```

This is equivalent to:

```
dpbuddy import -file domain.zip -dry-run true
```

You can also provide explicit "false" value:

```
dpbuddy import -file domain.zip -dry-run false
```

All commands support the "-h" option, which displays the list of all supported options and their descriptions, e.g., "dpbuddy import -h".

There are several other options that are common to all DPBuddy commands. All these options are listed in Common Attributes/Options. For example, to specify a DataPower domain, you can run the "import" command with the "-domain" option:

```
dpbuddy import -file domain.zip -domain my-domain
```

You can specify the default value for any common option in a configuration file. The configuration file should have the properties (name=value) format. For example, to specify a DataPower user used to connect to DataPower, define "dp.username=my-domain" property in the configuration file. Note that the name of the property could be different from the name of the option. Usually, properties have the "dp." prefix. This is done for the sake of compatibility with Ant property names.

DPBuddy follows these steps to locate and load the configuration file:

- If the "-configFile" command-line option is specified, DPBuddy will use the value of this option as the path to the configuration file.
- If the "DPBUDDY_CONFIG" environment variable is specified, DPBuddy will use the value of this variable as the path to the configuration file.
- DPBuddy will attempt to load the file "dpbuddy.properties" from the current directory (the directory where the command is run).
- DPBuddy will attempt to load the file "dpbuddy.properties" from the user's home directory using the path <home>/.dpbuddy/dpbuddy.properties.

If the configuration file was not found, DPBuddy will print a warning message.

DPBuddy uses Apache Commons Configuration to parse the configuration file. Commons Configuration's format supports many advanced features, including variable interpolation. This means that a property in the configuration file can reference another property's value using "${}" notation. "${}" can also be used directly on the command line as an option's value:

```
dpbuddy import -file "\${import.file.path}"
```

In this example "import.file.path" is the property defined in the configuration file. Note that "$" must be escaped using "\$" on Linux to prevent the shell from treating it as a shell variable.

DPBuddy pre-populates all built-in Ant properties and Java system properties, so any of these properties can be used on the command line or in the configuration file:

```
dpbuddy tailLog -lines "-1" -logFile "\${java.io.tmpdir}/dp.log" -
appendTimestamp
```

You can also split your DPBuddy configuration into multiple files and use "includes" to create a master file, e.g., "include=another_config.properties".

You can use the sample configuration file included in the distribution under "samples/cli-shell" and "samples/cli-win" as the starting point for your DPBuddy configuration file.

We recommend using the prefix-based property resolution mechanism which allows you to prefix all environment-dependent properties with the appropriate environment/device/domain prefixes. You can then point DPBuddy to a specific environment using the "env" option:

```
dpbuddy import -file domain.zip -env dev
```

Using DPBuddy with Apache Ant

DPBuddy Ant tasks do not have any special requirements. Add DPBuddy task definition to your build file, create appropriate Ant targets and add the tasks as needed. The samples packaged with the DPBuddy distribution should provide a good starting point.

Enabling Verbose Logging

DPBuddy prints detailed debug information when it runs in verbose mode. This information includes SOAP/XML generated by DPBuddy and responses returned by DataPower.

To simplify the viewing of request/response XML, DPBuddy does not print Base64-encoded strings used by DataPower to transfer files.

To run DPBuddy in verbose mode, provide the "-v" option to a DPBuddy command or to Ant, e.g.,

```
dpbuddy import -file domain.zip -v
```

If you're using the environment task or environment prefixes, you could also set the property "dpbuddy.trace.properties" to "true" in the configuration file or in an Ant properties file/build file. When this property is set, DPBuddy will print information explaining how a property was resolved.

Recommended Development and Build/Deployment Process

DPBuddy can support any build/deployment/release process; however, you will get the most value out of DPBuddy if you follow a structured release process with strict separation of environments and reliance on version control. In short, the build/deployment process of your DataPower artifacts should be similar to the process used for any "regular" Java EE or Java application. The process should include the steps for creating the build, deploying it to a test environment, performing testing, promoting the build to a "higher" environment and so on.

A suggested process is described below. Your DPBuddy distribution includes sample files implementing this process. You can find these files under samples/ant-end-to-end, where "dp-build-deploy.xml" is the name of the Ant file. The text below references the relevant target names in this file. DPBuddy provides tasks/commands to fully automate all the steps described below, except for the testing step.

Developing DataPower Artifacts

You can have many domains used for development, e.g., you could set up different domains for testing different services. Developers can modify configuration/artifacts in their domains as needed by following these general steps:

- Create/update DataPower services and objects using WebGUI.
- Upload various files used by DataPower services using the "copy" task. Target: "upload.files".
- Conduct testing using JUnit, SoapUI or other tools.
- Export DataPower configuration using the "export" task and check in the configuration file under version control. We recommend checking in XML configuration files (as opposed to exported .zip files) so that version control can be used to track changes and diff the files. Note that DPBuddy will automatically clean and format the exported XML configuration so that it can be checked in right away. Target: "export.services".

We recommend replacing environment-dependent values with variables during export. This can be done using the "transform" element of the "export" task. As a result, you will create a "canonical" environment-independent configuration. Changes to this configuration can be easily tracked using a version control system.

Creating a Build in a Test Environment

Test environment or a test domain can be used for assembling all services, objects and files required for the release. After this is done, the entire domain (which could include multiple domains in a complex system) should be treated as a releasable artifact. The export of the test domain is analogous to a deployable Java EE application file, such as a WAR or an EAR archive.

- Delete and create the test domain. You should always start from a "clean slate" to avoid issues that might be caused by objects created during an earlier test cycle. To create an empty domain simply export a domain definition from the "default" domain (by navigating to "Application Domains" in WebGUI or by running DPBuddy's export task) and import the definition with the new name. Targets: "export.domain.definition", "create.clean.domain".
- Upload certificates/keys required for crypto objects. Keys should be stored in a separate protected area, so they have to be uploaded as a separate step. Target: "upload.certs".
- Check out a release baseline/label from version control. The baseline must include all files and configuration artifacts required to build a fully functioning domain. You can utilize a CI/Build server such as Jenkins/Hudson to perform this task.
- Import configuration of the crypto objects. Crypto configuration (but not the keys themselves) could be stored under version control, and can be imported using the "import" task similar to any other DataPower service or object. Target: "import.crypto.config".
- Upload files used by DataPower services and objects. This should cover all XSLT, WSDL, XSD and all the other files that have to reside on the DataPower filesystem (usually, under "local:/"). Target: "upload.files".
- Import DataPower configuration, including all objects and services. You may need to update your configuration to adapt it to the target environment using DPBuddy's configuration transformation capabilities. Target: "import.services.config".
- Verify the import and save the domain's configuration. You can use various DPBuddy assertion tasks, such as "assertState" for this purpose. Target: "verify.import".
- Export the entire domain. This export constitutes the "build" of DataPower artifacts that could be promoted to other environments after testing is done. You can use a Maven repo or version control to store the domain export. You should handle it same way you deal with binary build artifacts produced by application builds, such as WAR and EAR files. Target: "export.domain".
- Conduct automated or manual testing of deployed services.

Promoting the Build to a "Higher" Environment

The domain export created during the "build" step should be deployed (promoted) to other environments as a whole; ideally there should be a single "import" target to perform the entire deployment.

- Quiesce the target domain. You can quiesce the entire domain or specific services running in the domain. Target: "quiesce.domain".
- Clean the domain. You could either delete and create the domain or clean it. If you don't want to re-upload keys every time you deploy, it might be enough to simply reset the domain (using the "reset" task) and delete the content of the "local:/" filesystem (using the "delete" task). Target: "clean.domain".

- Import the "build" (the export of the test domain). Depending on your process, you may need to first retrieve the file from version control or from a Maven repo. Alternatively, you could promote the build using your build/CI server. Target: "import.domain".
- Similar to deploying to a test environment, you should verify that all newly imported objects and services are up and running. Target: "verify.import".
- Create the checkpoint and export the domain's configuration in XML format (without files). This configuration could be put under version control to create a complete history of all changes in a specific domain. Target: "checkpoint".
- Conduct smoke testing.

Deploying to a Production Environment

Deploying to a production environment should generally follow the steps outlined in the previous section. There are normally multiple devices employed in production so you will need to repeat these steps multiple times. This may be scripted in Ant/Gradle (see "deploy.2devices" target) or, if a manual smoke testing/verification is involved, the same target could be executed multiple times.

Common Attributes/Options and Properties

There are a number of attributes/options that are common to all DPBuddy tasks/commands.

Any common attribute/option has a corresponding property that can be used to specify a default value in the configuration file or in a build file. This keeps you from having to repeat the same attribute for every task. A value from the attribute/option overrides a value from the property. For example, the "dp.domain=test" property can be overridden using the "domain='test1'" attribute set at a task level.

Every common property has two aliases, one using dots as a word separator (following Ant naming conventions) and the second one using lower camel case. The second alias should be used when running DPBuddy from Groovy or Gradle (or any other scripting language) since Ant naming conventions conflict with the commonly used dot notation for accessing object properties.

DataPower Connection Attributes/Options

DPBuddy needs to know the URL of the SOAP interface (XML Management interface) and a user name and a password in order to be able to connect to a device.

Following is the list of common connection properties/attributes:

Property name	Attribute/ Option	Description	Required
dp.url dpUrl	url	URL of the SOAP interface (XML Management interface). The path defaults to "/service/mgmt/current", which always corresponds to the most up-to-date DataPower schema. If the port is not specified, DPBuddy will attempt to use the port 5550. You can specify just the host name of the appliance, in which case DPBuddy will default to "https:" and the default port.	Yes
dp.username dpUsername	user CLI alias: -u	Valid username with administrative privileges.	Yes
dp.password dpPassword	password CLI alias: -p	Password.	Yes
dp.domain dpDomain	domain	Target domain for DPBuddy tasks.	Yes
dp.trust.all.certs dpTrustAllCerts	trustAllCerts	If this property is set to "true," DPBuddy will ignore SSL certificates' validation errors when connecting to a device. If this property is not set, or set to "false", you will need to add the DataPower certificate to your JDK's keystore. You can obtain the DataPower certificate from the browser when you access Web GUI. Defaults to "true".	No

Environment Attributes/Options

Environment prefix is used to point to different sets of properties belonging to different environments. Please refer to Managing Environment-Specific Properties/Variables for more details.

Property name	Attribute/ Option	Description	Required
dp.env.prefix dpEnvPrefix	env	Sets the "environment prefix" which is used to determine what properties to use. Once the prefix is set, DPBuddy will attempt to pre-pend this prefix to every requested property name and check if the prefixed property is defined.	No

Schema Validation Attributes/Options

DPBuddy can optionally validate SOMA XML requests against the DataPower SOMA XML schema. Several DPBuddy tasks, such as "action", "request" and "modifyConfig" allow the user to specify free-form XML as part of the request. DataPower itself validates all XML requests, but for security reasons, it does not return validation errors to the client. Instead, it returns the "internal error" message, which is not very helpful for troubleshooting. Validation errors can then be found in the device's logs, but scanning the logs takes time. DPBuddy, on the other hand, validates XML on the client so that validation error messages are displayed right away.

The following DPBuddy properties and attributes control schema validation:

Property name	Attribute/ Option	Description	Required
dp.validate dpValidate	validate	If set to "true," validate all XML SOMA requests. Note that enabling this setting globally (using "dp.validate" property) will result in slower performance for "import" and some other tasks. Defaults to "false" except for the tasks "modifyConfig" and "somaRequest".	No
dp.schema dpSchema	schemaFile	Full path to "xml-mgmt-ops.xsd" or the directory containing all DataPower schema files. DataPower schema files can be downloaded	No

		from the "store" filesystem of the device. The schema files include "xml-mgmt-ops.xsd," "xml-mgmt.xsd," "xml-mgmt-base.xsd" and, starting with firmware version 6, "xml-mgmt-b2b.xsd". DPBuddy comes bundled with the schema files for 6.0.1 firmware. When schema validation is enabled, DPBuddy will default to this schema. You may want to download the schemas from your device and specify the "dp.schema" property if you have a different firmware version.	

Examples

```
dp.validate=true
dp.schema=./schemas
```

"Save" Attribute/Option

DPBuddy is capable of automatically saving domain configuration after a change. This is equivalent to clicking "Save Config" in WebGUI.

Most of DPBuddy's configuration changing tasks support the "save" attribute/option. If this attribute is set to "true", the configuration will be saved after successful completion of the task.

You can also enforce auto-saving globally by setting the "dp.auto.save" property. If this property is set, all the configuration-changing tasks will save domain configuration upon completion.

The "save" attribute/option takes the precedence over "dp.auto.save" property.

The "save" attribute and "dp.auto.save" property are supported by the following tasks: "import", "modifyConfig", "delConfig", "resetDomain", "assertStatus", "assertState", "action", "somaRequest", "checkpoint" and "rollback".

You must explicitly invoke the save task to persist domain configuration if "dp.auto.save" or "save" attribute are not set or are set to "false".

Property name	Attribute/ Option	Description	Required

dp.auto.save dpAutoSave	save	If set to "true", saves the domain configuration. Defaults to "false".	No

License File Attribute/Option

Please see the DPBuddy License section for more information about the license file. This setting can be ignored if you're using a trial version of DPBuddy.

Property name	Attribute/ Option	Description	Required
dpbuddy.license.file dpbuddyLicenseFile	license	Location of the license file.	No

Use or Regular Expressions to Select DataPower Objects

Many DPBuddy tasks/commands can be applied to multiple DataPower configuration objects. You can see a full list of the configuration objects available in your environment by expanding "Objects" in the DataPower WebGUI.

Each object has a "class" (type) and a name. Classes are defined in the DataPower XML Management schema. You can see the list of all classes under the definition of "AnyConfigElement" in "xml-mgmt.xsd".

If you look at any DataPower configuration file created using "export", child elements of the "configuration" element are class names.

DPBuddy allows for using regular expressions to match DataPower objects. For example, delConfig task supports nested "object" elements defining what configuration objects to delete. Similarly, the export task's nested "exportObject" elements define what objects to export.

You can also use the "classPattern" and "namePattern" options/attributes to specify the regexps. These options should be used with DPBuddy CLI. Nested elements are not supported by DPBuddy commands; they are only supported by Ant tasks.

DPBuddy selects objects based on matching class names and/or object names. Only one of the two is required; you can select configuration objects based on just the class name, based on just

the object name, or based on both. In the case of "export"/"exportObject", all objects will be selected if both class and name attributes are omitted.

Note that your regular expression needs to match the entire name and not just a substring within a name, so use ".*" where needed.

Examples

Match all HTTP front side handlers:

```
<object class="HTTPSourceProtocolHa.*" />
```

Match all objects (irrespective of the class) with the name starting with "Test":

```
<object name="Test.*"/>
```

Match all HTTP front site handlers whose name starts with "Test":

```
<object class="HTTPSourceProtocolHa.*" name="Test.*"/>
```

You can use "or" regular expressions if you need to select objects matching multiple classes and/or names. The example below selects all WS proxies and firewalls:

```
<exportObject class="(WSGatew.*|.*Firewall.*)" />
```

Using DPBuddy to Transform DataPower Configuration and XML Files

There is often a need to apply environment-specific modifications to the DataPower configuration. This may include changing ports, removing or adding some configuration elements and other changes.

DataPower offers the deployment policy-based mechanism that can be executed on the device as part of importing the configuration. DPBuddy fully supports deployment policies, and also provides additional XML transformation capabilities. DPBuddy supports many transformation actions that can be executed as part of the "import", "export", "modifyConfig" or "copy" tasks. The actions include "add", "update", "delete", "setText", "replaceText", "include" and "exclude". They are explained in detail later in this section.

All actions except "dpInclude", "dpExclude" and "replaceText" rely on XPath expressions. The actions are applied to all elements or attributes that match the expression.

This simple example changes the local port of the front-side HTTP handler to the value provided by the "ws.port" Ant property. The transformation is applied to the "export.xml" file located inside the zip file.

```
<dp:import file="${wsproxy.zip.file}" >
    <transform>
        <setText xpath="//LocalPort"  value="${ws.port}" />
    </transform>
</dp:import>
```

See "samples/ant-tasks/transform.xml" file located in your DPBuddy distribution or online for more examples.

DPBuddy's transformation actions have many powerful features:

- Can be applied to any XML file, not just DataPower configuration files. You can transform any XML file included in "import" or "copy".
- Support Ant and Groovy/Gradle variables inside XPath or in XML fragments.
- Reusable: you can define a reusable group of transformation actions and then reference it from any other group.
- Can be turned on and off depending on a Boolean expression that can reference any Ant property or Groovy/Gradle variable. For example, a transformation action could be triggered only for certain environments.
- Support Groovy expressions for complex transformations. E.g., you can apply a Groovy/Java method to an existing value.
- Easy to debug: transformed files are saved locally and also logged using verbose/debug mode.
- Can be developed locally without having to connect to a DataPower device. Both the "import" and "copy" tasks support "transformOnly" mode, which runs transformations but does not connect to the device.

Any transformable file could have Ant/Groovy/Gradle variables references in any text node or in any attribute. DPBuddy's transformation logic always attempts to resolve Ant/Groovy/Gradle variables after all transformation actions have been applied. The transformation will fail if any of the variables remains unresolved.

In case you have many transformation rules, it might be beneficial to specify the transformations in a separate file and then include this file in your main DataPower build/deploy script.

"transform" Type

"transform" is the container for all transform actions, such as "add", "setText" and so on. Transformation actions are executed in the order they are specified inside "transform".

"transform" is supported by the following DPBuddy tasks: "import", "export", "copy" and "modfyConfig".

When used with import, "transform" could be specified within the transformFiles element. "transformFiles" determines which files are going to be transformed.

For other tasks, such as copy or modifyConfig, "transform" is specified within the "fileset" or "dpFileset" element, so the transformations are applied to all the files that matched "include" and "exclude" patterns of the fileset.

"transform" supports "refid" and "id" attributes similar to Ant's "path" element. "transform" with "id" can be defined outside of any Ant target. "transform" can also contain any number of nested "transform" actions with "refid". DPBuddy will first execute actions referenced by nested transforms.

Attributes Common to All Transform Actions

Attribute	Description	Required
xpath	XPath expression the action will be applied to. Not applicable to the "dpInclude", "dpExclude" and "replaceText" actions.	Yes
matchRequired	Fail if xpath doesn't match any element/attribute. Defaults to "true".	No
if	A Groovy Boolean expression (has to return true or false). The expression has access to all Ant properties as Groovy variables. In addition, there is a special "dp" variable containing the following properties: "url", "username" and "domain". These properties are populated with connection information for the device and domain to which the task executing this transform action is being connected. The expression has to evaluate to "true" in order for the transform action to be executed.	No

"setText" Transform Action

If the action's XPath expression matched elements, “setText” sets the value of the text node of these elements.

If the action's XPath expression matched attributes, “setText” sets the value of these attributes.

Attribute	Description	Required
value	Value to set matched elements or attributes to.	At least one is required
expression	A Groovy expression-returning String. The expression has access to all Ant properties as Groovy variables. The expression can also use the special variable "currentValue", which is set to the current value of the matched text node or the attribute.	

"replaceText" Transform Action

"replaceText" searches for the provided search string in all attributes and text nodes. If the string is found, it will be replaced with the value specified in the "replaceWith" attributes.

The search is case-sensitive.

This action does not use XPath, so all attributes and text nodes are going to be checked.

Attribute	Description	Required
textToReplace	Text to replace.	Yes
replaceWith	Replacement string.	Yes

"add" and "update" Transform Actions

"add" and "update" add XML fragment nested within the action tags to the matching elements. In other words, the root element of the XML fragment becomes the child of the matched element. "update" removes all children of the matching element prior to adding.

"add" and "update" cannot be applied to attributes.

"delete" Transform Actions

"delete" simply removes all matching elements.

"delete" cannot be applied to attributes.

"dpExclude" and "dpInclude" Transform Actions

These actions don't use XPath. Instead, they support regular expressions matching DataPower configuration classes and objects.

"dpExclude" removes all configuration objects matching class names/object names. Conversely, "dpInclude" removes all configuration objects that do not match class names/object names specified in the action.

Attribute	Description	Required
class	Regular expression matching names of DataPower configuration classes.	At least one is required
name	Regular expression matching names of DataPower configuration objects.	

"namespace" Nested Element

All namespace prefixes used in XPath expressions of the transform actions have to be explicitly defined using the "namespace" element. Note that transforming DataPower configuration files does not require any namespace definitions.

Attribute	Description	Required
prefix	Prefix of the XML namespace that will be used inside XPath expressions.	Yes
uri	XML namepsace's URI.	Yes

Managing Environment-Specific Properties/Variables

A typical organization uses multiple DataPower devices, and multiple domains in each device. Usually, at least one device is used for development and testing; multiple devices could be used in production.

Different devices have different IPs, ports and potentially other parameters (environment variables or environment properties). DPBuddy provides an extensible and flexible mechanism for managing environment properties. DPBuddy's mechanism can be used instead of or in addition to any traditional approach for managing environment properties, such as specifying all properties for a given environment in a separate file. It can be used with the DPBuddy CLI (by the virtue of configuration files) or from Ant.

Internally Ant delegates property resolution to the property helper classes described in the Ant User Manual. DPBuddy comes with a simple property helper implementation (a.k.a. "property provider") that relies on property name transformation based on name prefixes. You can easily build property providers that use other mechanisms to resolve properties – XML files, external database, corporate CMDB and so on. This section describes the logic of DPBuddy's default property provider.

DPBuddy's property provider relies on prefixes to define property namespaces. For example, you can prefix all properties for the "test" environment with the "test." prefix. When a property (e.g., "dp.url") is requested by Ant, the property provider will check if "test.dp.url" is defined and if it is, it will return the value of this property. The property provider pre-pends each known prefix to the requested name of the property and checks if a property with this prefixed name exists. Once the existing property is found, its value is returned to the requesting task/command.

You can concatenate multiple prefixes together. For example, you could have a prefix for an environment and then a sub-prefix for a DataPower domain in that environment, such as "test.domain1.". DPBuddy's property provider will first check if a property with the prefix "test.domain1." is defined. Then it will check if a property with the prefix "test. " is defined. In other words, prefixes create a hierarchy of namespaces. You can define properties specific to a domain using the "test.domain1." prefix and properties specific to the "test" environment using the "test." prefix.

DPBuddy's property provider uses "." as the prefix separator, which is consistent with Ant's property naming style.

The prefix-based property management mechanism is completely generic and can be used to manage any set of properties, not just the ones related to DPBuddy and DataPower.

Properties containing prefixes are regular Ant properties; they can be defined in one or multiple properties files, or inline using the Ant "property" task. They can also be specified in the configuration files used by DPBuddy CLI.

You can define the "dpbuddy.trace.properties" property and set it to "true" if you need to see how the property provider transforms property names. You will need to run Ant in verbose mode (-v option) to see this output.

"env" Attribute/Option and "environment" Task

The environment prefix is specified using the "environment" task, the "env" attribute/option or the "dp.env.prefix" property. The "env" attribute/option is supported by all DPBuddy

tasks/commands. "dp.env.prefix" defaults the environment prefix to the value of this property for all DPBuddy tasks and commands.

The environment prefix can contain one or multiple sub-prefixes separated by ".". The trailing dot is not needed.

Setting the "env" attribute/option is equivalent to invoking the "environment" task right before the current task.

Once the prefix is set, the DPBuddy's property provider will use the supplied set of prefixes for property resolution.

The "environment" task can be called multiple times within a build file. Each invocation will override the prefix defined earlier. Similarly, "env" attribute/option updates the environment prefix value defined earlier. All subsequent tasks will use the new value.

You can also remove the environment prefix by specifying an empty string as the value: prefix="".

Attributes of the "environment" Task

Name	Description	Required
prefix	Environment prefix consisting of dot-delimited sub-prefixes. DPBuddy will try to resolve a property by pre-pending each sub-prefix to the property name starting with the entire prefix string, then cutting the ending sub-prefix and trying the resulting prefix and so on. An empty value (prefix="") will disable the DPBuddy's property resolution mechanism.	Yes
providerClass	Property provider implementation. The class has to implement the "org.apache.tools.ant.PropertyHelper.PropertyEvaluator" and "com.myarch.propertyselector.PropertyProvider" interfaces. Defaults to DPBuddy's property provider.	No

Examples

This example shows how properties can be defined at both the domain and the device level. We use two-part prefixes in the <device>.<domain> format:

```
<!-- DataPower url is defined at the device level -->
<property name="devtestdevice.dp.url" value="https://devtest.dp" />
<!-- Each WS proxy has its own port in each of the domains -->
<property name="devtestdevice.devdomain.wsproxy.port" value="8082" />
<property name="devtestdevice.testdomain.wsproxy.port" value="9082" />

<!--Provide our prefix to DPBuddy's provider for property resolution-->
<dp:environment prefix="devtestdevice.devdomain" />

<!-- dp.url should point to devtest.dp url -->
<echo message="dp.url value: ${dp.url}"/>
<!-- the value of the port should be the one for the devdomain -->
<echo message="wsproxy.port value: ${wsproxy.port}"/>
```

See "samples/ant-tasks/environment.xml" file located in your DPBuddy distribution or online for more examples.

Auditing DPBuddy Tasks/Commands

DPBuddy is capable of creating and maintaining an audit/log file tracking execution of the DPBuddy tasks that make changes to DataPower configuration or to the files residing on DataPower filesystems. The audited tasks include "import", "delConfig", "copy", "delete" and "resetDomain".

DPBuddy creates its audit file in XML format. The file contains a list of events. Each event represents an execution of one of the DPBuddy's tasks listed above.

An event contains the following data elements:

- Name of the task.
- Name of the local user who executed the task.
- Version/release info optionally supplied by the build process using "dp.release.info" property. This value can be used to specify the version of the DataPower artifacts that are being deployed to a domain.
- Timestamp of the event.
- DataPower URL.
- DataPower domain.
- DataPower user name used by the task.
- List of affected (updated or deleted) DataPower configuration objects.
- List of uploaded or deleted files. For uploaded files their source location/timestamp will be captured.

Here is an example of an event generated by the "import" task:

```
<event>
```

```
    <action>import</action>
    <localUserName>developer1</localUserName>
    <releaseInfo>release-1.0.1</releaseInfo>
    <timestamp>2014-01-17 14:34:39</timestamp>
    <dpUrl>https://dp-test:5550/service/mgmt/current</dpUrl>
    <domain>test</domain>
    <dpUserName>dpbuddy</dpUserName>
    <objects>
        <object class="HTTPUserAgent" name="default" />
        <object class="XMLManager" name="default" />
        <object class="Matching" name="testFirewall" />
        <object class="StylePolicyAction"
            name="testFirewall_request_xform_0" />
        <object class="StylePolicyAction"
            name="testFirewall_request_results" />
        <object class="StylePolicyRule" name="testFirewall_request" />
        <object class="StylePolicy" name="testFirewall" />
        <object class="XMLFirewallService" name="testFirewall" />
    </objects>
    <files>
        <file localPath="D:\dobuddy\dpconfigs\XMLFirewall.xml"
            lastModified="2014-01-17 00:14:41" />
    </files>
</event>
```

DPBuddy keeps the audit file in multiple places:

- The audit file is created and maintained on the machine where DPBuddy was executed. The location of the file is defined by "dp.audit.local.file" property.
- A copy of the audit file is also uploaded to the DataPower domain against which a DPBuddy task was executed. This copy of the file will contain only the events relevant to this domain and the device. If "dp.audit.domain" property was set to "default", the default domain will contain all audit entries for this device.

All audit files in all locations are updated independently. For every audit event DPBuddy will download the file from the target domain/device, update it with the new event and then upload the file back to the domain.

By default DPBuddy uploads the audit file to the root of the "chkpoints:" filesystem (as opposed to "local:"). This is done so that the audit file does get exported when the entire "local" filesystem is exported and does not get deleted when "local" is cleaned. The location can be changed using "dp.audit.dp.file" property.

Properties for Configuring Audit

Similar to other common DPBuddy properties, every audit-related property has two aliases, one using dots as a word separator (following Ant's naming conventions) and the second one using

lower camel case. The second alias should be used when running DPBuddy from Groovy or Gradle (or any other scripting language) since Ant naming conventions conflict with the commonly used dot notation for accessing object properties.

Property name	Description	Required
dp.audit dpAudit	If set to "true", create audit entries for auditable DPBuddy tasks. Defaults to "false".	No
dp.audit.local.file dpAuditLocalFile	File name (including path) of the audit file on the machine where DPBuddy runs. Defaults to "${tmpdir}/DPBuddy-audit-log.xml". "tmpdir" is defined by the "java.io.tmpdir" system property of the JDK used to run DPBuddy.	No
dp.audit.save.to.dp dpAuditSaveToDp	If set to "true", update the audit file on the device. Defaults to "true"	No
dp.audit.dp.file dpAuditDpFile	File name (including path) of the audit file on the DataPower domain/device. Defaults to "chkpoints:/dpbuddy-audit-log.xml"	No
dp.audit.domain dpAuditDomain	DataPower domain the audit file is saved (if dp.audit.save.to.dp is set to "true"). Defaults to the target domain of an auditable task, i.e., if "import" runs against "TestDomain", this is where the audit file will be stored. Note that this results in each domain having its own audit file. Alternatively, if this property is set (e.g., to "default"), there will be a single audit file in the specified domain containing audit entries for all the domains of the device.	No
dp.release.info dpReleaseInfo	Text to save in the "releaseInfo" field of the audit file. You can use this property to specify the version number of the released application and/or DataPower configuration.	No

dp.local.user dpLocalUser	User ID of the user who executed the task that resulted in an audit event. If you run DPBuddy using a build server such as Jenkins, this property can be used to capture the logged-in name of the user who started the build. Defaults to: currently logged in user ("user.name" system property in Java).	No

Tasks/Commands for Changing and Exporting Configuration

"import"/"buddyImport"

"import" is the main task for updating DataPower's configuration. It imports previously exported configurations and files into a DataPower domain. The input for import can be created using WebGUI or DPBuddy's export task. The functionality supported by "import" is similar to "Import Configuration" available from WebGUI. "import" can handle a stand-alone XML configuration file or a zip file containing both configuration and files to be uploaded to the DataPower file system.

Upon completion, "import" prints a report with the list of all updated configuration objects and files.

"buddyImport" is the alias of "import". Eclipse Ant editor gives warning when "dp:import" is used as it confuses it with the Ant's built-in "import" task. "dp:buddyImport" can be used to avoid this.

"import" allows you to specify Ant/Groovy/Gradle variables in any text node or an attribute of any XML file that is part of the import. This also includes deployment policy files.

"import" supports transformation actions described in the section Using DPBuddy to Transform XML Files. Transformations can be applied to XML import files or to the files inside a ZIP archive.

Attributes/Options

Name	Description	Required
file	File to import. This could be a zip or an xml/xcfg file.	Yes
deploymentPolicyFile	Path to the deployment policy file on the local filesystem. The deployment policy is automatically uploaded/configured on the device before the request is executed.	No

deploymentPolicyName	Name of the existing deployment policy to use for the import operation. The policy should already exist on the device.	No
overwriteFiles	If set to "true", overwrite existing files with the files included with the import. Defaults to "true" (DataPower default).	No
overwriteObjects	If set to "true", overwrite existing objects with the objects defined as part of this import. Defaults to "true" (DataPower default).	No
rewriteLocalIp	If set to "true", the local address bindings of services contained in the import package will be rewritten to match the equivalent interfaces of the device. Defaults to "false" (DataPower default).	No
dryRun	If set to "true", run import in "dry run" mode. Defaults to "false" (DataPower default).	No
workDir	Path to the directory used to hold temporary files. DPBuddy will unzip the content of the zip file into this directory. This is needed when "transform" is used to transform configuration files contained inside a zip file. Note that DPBuddy cleans this directory every time the import task is executed. If "transform" is not used or when the imported file is XML, "workDir" is ignored. Defaults to ${java.io.tmpdir}/dpbuddy/import. "java.io.tmpdir" is a Java system property. It usually defaults to /tmp on Unix and \Users\<user name>\AppData\Local\Temp on Windows.	No
transformOnly	If set to "true", DPBuddy will transform the files according to the nested "transform" element, but it will not execute the import itself. In this mode, the task does	No

	not connect to DataPower. This is convenient for developing/troubleshooting "transform" actions. Defaults to "false".	
assertObjectsUp	If set to "true", upon completing the import, DPBuddy will invoke the assertState task to verify that all objects in the domain are in "up" state. Defaults to "false".	No
assertPorts	A comma-delimited list of ports to check after the import is done. DPBuddy will invoke assertOpenPorts task to verify that all ports in this list are assigned to active services. If this is not the case, DPBuddy will raise an exception.	No
resolveVars	If set to "true", attempt to resolve variables (${}) inside the export file. DPBuddy will raise an exception if any variables/properties remain unresolved. This setting is equivalent to defining an empty "transform" nested element. Defaults to "false".	No
checkpoint	Creates checkpoint with this name before the import.	No
rollbackOnError	If set to "true", roll back the domain's configuration to the last checkpoint (if it exists), in case of any import or verification errors. Use it in conjunction with the "checkpoint" attribute. Defaults to "false".	No
reset	If set to "true", reset the domain's configuration before the import. Note that this will wipe out all objects and services in the domain. Defaults to "false".	No

quiesce	If set to "true", quiesce the domain before the import. Defaults to "false".	No
unquiesce	If set to "true", un-quiesce the domain after the import is completed. Defaults to "false".	No
domainComments	Sets the comment field of the target application domain to this value. You can use this attribute to specify the build date, version number and other tracking information. You can view this value from DataPower WebGUI by navigating to the "Application Domains" screen from the "default" domain.	No

Examples

This is a simple example of the import task. You can find more examples under "samples" in your distribution or online.

```
dpbuddy import -file dpconfigs/WSProxy.zip
```

```
<dp:import file="${wsproxy.zip.file}" />
```

"transformFiles" Nested Element

"transformFiles" defines what files to transform by the transformation actions contained in the nested "transform" element. "transformFiles" can only be used if the imported file is "zip". For XML import file transformation actions will be applied to the file referenced in the "file" attribute of the task.

"transformFiles" inherits all the attributes and elements of the Ant patternset. You have to define your include/exclude patterns in such a way that only XML files will be selected by the patternset. DPBuddy will attempt to apply all transformation actions to all matched files.

Usually, transformations are applied to the DataPower configuration file typically named "export.xml" in the zip file. In this case, you just need to specify "export.xml" in the "includes" of the "transformFiles":

```
<transformFiles includes="export.xml">
```

"transformFiles" has to contain at least one "transform" element, as explained earlier in this document.

If "transformFiles" is omitted and the imported file is "zip", transformation actions will be applied to "export.xml" inside the zip file. If "export.xml" does not exist, an error will be thrown.

You can specify multiple "transformFiles", thus applying different transformation actions to different files.

"transform" Nested Element

"transform" contains the transformation actions documented in the section Using DPBuddy to Transform XML Files.

"modifyConfig"

The "modifyConfig" task updates the DataPower configuration from the provided configuration file and/or the configuration information specified inline within the "configuration" tag. The configuration information contains DataPower object definitions in the format defined by the DataPower schema (xml-mgmt.xsd, "AnyConfigElement" type).

As with "import", you can use DataPower's export facility or DPBuddy's "export" task to produce an initial version of the configuration file.

Also similar to "import", the configuration can be transformed using DPBuddy's "transform" actions.

Configuration can contain variables in any of its text nodes or attributes.

Unlike "import", however, "modifyConfig" does not completely overwrite existing configuration objects. Instead, it updates nested elements of the configuration. For example, for a load balancer group (LBG), "modifyConfig" will update only the members included in the request. If an LBG member does not exist, "modifyConfig" will create it. It will not affect any other group members.

DataPower XML schema contains the following explanation for "modifyConfig":

"Vectors with indexes (example: BaseWSDL in ModifyWSGateway) will be merged based on matching index values (example: WSDLSourceLocation for BaseWSDL). Vectors without indexes (example: DebugTrigger in ModifyConfigBase) and simple vectors (example: AcceptedConfig in ModifyConfigDeploymentPolicy) will have all their elements replaced with any new elements supplied via modifyConfig."

You can use "modifyConfig" to dynamically change your configuration, for example, to disable LBG members if you need to shut down some back-end servers.

Please note that DataPower forces XML validation of "modifyConfig" requests (it does not do it for "import"). Unfortunately, in some cases, DataPower produces a non-compliant XML export,

which causes validation to fail. Usually, removing the "Memoization", "DebugMode" and "DebuggerType" elements is sufficient to let it pass. Also, all ports elements (e.g., "HealthPort" in "LBGroupMembers") must have an integer value; you can enter "0" to default these values.

"modifyConfig" will try to validate the request against the DataPower XML schema on the client (you can override this by setting the "validate" attribute to "false"). If the schema validation was turned off, the validation would still fail on the device and you would get the "Internal Error" message.

Paying (i.e. non-trial) DPBuddy customers can contact DPBuddy support if they have questions or issues with defining their "modifyConfig" requests.

Attributes/Option

Name	Description	Required
file	Path to the configuration file. If the task contains "transform" element, the transformations will be applied to the content of the file. The file must be a valid DataPower configuration file. "datapower-configuration" must be the root element. All configuration objects must be enclosed inside the "configuration" element, which must be a child of "datapower-configuration".	No
resolveVars	If set to "true", attempts to resolve variables (${}) inside the configuration. DPBuddy will raise an exception if any variables/properties remain unresolved. Defaults to "true".	No

"transform" Nested Element

"transform" contains transformation actions documented in the section Using DPBuddy to Transform XML Files. This element is optional.

"fileset" Nested Element

You can use the Ant "fileset" element to specify multiple config files that will be used to update DataPower configuration. DPBuddy will combine all matching files into a single request to the device. You can specify multiple "fileset" elements within the same task.

Each fileset could contain a nested "transform" element with the transformation actions. The actions will be applied to the files matched by this fileset. Note that the global "transform" specified as the child of the task will not be applied to the matching files.

"configuration" Nested Element

You can specify any valid DataPower configuration inside this element as an alternative to keeping the configuration in an external file. If the task contains "transform" element, the transformations will be applied to this configuration. The configuration's text nodes and attributes can contain Ant variables.

Examples

The following example shows how to use "modifyConfig" to disable an LBG member:

```
<dp:modifyConfig>
    <configuration>
        <LoadBalancerGroup name="${lbg.name}">
            <LBGroupMembers>
                <Server>${server.name}</Server>
                <Weight>1</Weight>
                <MappedPort>0</MappedPort>
                <Activity/>
                <HealthPort>0</HealthPort>
                <LBMemberState>disable</LBMemberState>
            </LBGroupMembers>
        </LoadBalancerGroup>
    </configuration>
</dp:modifyConfig>
```

"delConfig"

"delConfig" deletes DataPower objects. Similar to "export", it uses regular expression patterns, which match class names and object names, to determine what to delete.

It is often desirable to delete all objects related to a particular DataPower service, e.g., when deleting a Web services proxy, it may be necessary to delete the policy and the rules used by the proxy. You can accomplish this by deleting all objects prefixed with the name of the service, as this is the naming convention used by DataPower when creating services. In case there are dependencies between objects (e.g., a rule might be referenced by a policy and so the policy can be deleted only after the rule is deleted), "delConfig" will attempt to run the delete request multiple times until all matching objects are deleted.

Run this command in "dry-run" mode first to see what objects will be deleted, to avoid deleting items by accident.

You can specify objects to delete using classPattern/namePattern attributes and/or using the nested "object" element.

Attributes/Options

Name	Description	Required
classPattern	Regular expression defining what classes (types) are to be deleted.	No
namePattern	Regular expression defining object names to be deleted.	No
quiet	If set to "true", do not fail if no matching object is found. Defaults to "false", meaning that if any of the nested "object" elements did not result in a match, an exception will be thrown.	No
dryRun	If set to "true", do not delete anything, just print the matched objects. Defaults to "false".	No

"object" Nested Element

Name	Description	Required
class	Regular expression defining what classes (types) to be deleted.	At least one is required
name	Regular expression defining object names to be deleted.	

Examples

This example deletes the Web services proxies with the name starting with "test". If no such proxy exists, the task will complete successfully because of the "quiet" setting.

```
dpbuddy delConfig -quiet -classPattern WSGateway -namePattern "test.*"
```

```
<dp:delConfig quiet="true" dryRun="true">
    <object class="WSGateway" name="test.*" />
</dp:delConfig>
```

You can find more examples under "samples" in your distribution or online.

"export"

The "export" task exports configuration from a domain. The task supports all functions of the WebGUI "Export configuration" screen and provides some additional features.

Both XML/XCFG and ZIP file formats are supported. The format is determined automatically based on the provided file extension. "xml" and "xcfg" are treated as XML, all other extensions are treated as ZIP.

"export" supports the nested "transform" element containing the transformation actions described earlier. This can be used, for example, to replace actual values, such as port numbers with Ant/Groovy/Gradle variables. Later on, when the configuration is imported using "import", the variables will be automatically resolved. Note that "transform" can only be used with the XML configuration format.

DataPower embeds base64-encoded files into the exported XML configuration, including several WebGUI-related files. This makes it difficult to edit the configuration file or track changes. The "export" task removes all files from the exported configuration by default. It also removes the "interfaces" section of the exported configuration. Finally, "export" formats the configuration using indentation. As with "transform", these changes are only performed to XML configuration; configuration in ZIP format is saved unchanged. This feature can be turned off using the "defaultTransform" attribute.

Transforming and formatting XML configuration files makes it easier to store configuration under version control and to track the history of changes.

You can also provide a deployment policy as part of the export task. DataPower will add the policy to the resulting export configuration file. The policy will then be "activated" upon import.

Export and import can be used together to update configuration or to copy configuration between devices/domains. Simply run export first, save the configuration to a file and then run import using the same file.

Attributes/Options

Name	Description	Required
file	Name of the file to which the exported configuration should be saved. The task will create directories containing the file if they don't exist. The file must have "zip" or "xml" or "xcfg" extension.	Yes
classPattern	Regular expression defining what classes (types) to export, e.g., "WSGateway".	No

	If "class" is omitted, all classes will be matched. Specifying "classPattern" and "namePattern" attributes is equivalent to defining the nested "exportObject" element.	
namePattern	Regular expression defining the objects to be exported. Only the objects whose name matched this regexp will be exported. If "name" is omitted, all object names will be matched.	No
namePatterns	Comma-delimited list of regular expression patterns. Only DataPower objects that match at least one pattern will be exported. Objects will be exported irrespective of their class (type). This attribute is useful when you want to export all objects based on a naming convention. It can also be used to export a list of uniquely named objects without having to worry about what their classes are. **Note: this attribute is deprecated. Use the "exportObject" nested element instead.**	No
allFiles	If set to "true", export all files from the local:/ filesystem. If not specified, only the referenced files will be exported. For export in XML format, this setting will be ignored unless "defaultTransform" is set to "false". Defaults to "false" (DataPower default).	No
persisted	If set to "true", export only the persisted configuration. Defaults to "false" (DataPower default).	No
deploymentPolicyFile	Path to the deployment policy file on the local machine. The deployment policy is automatically uploaded/configured on the device before the request is	No

	executed. The deployment policy will be embedded with the export file.	
deploymentPolicyName	Name of the deployment policy to include with the export/backup. The policy has to already be defined on the device. The deployment policy will be embedded with the export file.	No
defaultTransform	If set to "true", applies default transformations to the exported XML configuration. This includes removing "/datapower-configuration/filesonly" and " /datapower-configuration/interface-data" elements and formatting the file using indentation. Defaults to "true".	No

"exportObject" Nested Element

This nested element defines the objects that will be part of the export. Its two key attributes are "class" and "name". Both attributes are optional, and both support regular expressions. This gives you a lot of flexibility; you can define regexps matching classes, objects, or both.

If you omit both "class" and "name", all configuration objects of the domain will be exported. Likewise, if you do not provide any nested "exportObject", all objects will be exported.

"exportObject" supports other attributes, such as "includeDebug". These attributes will be applied to all matched objects.

You can specify multiple "exportObject" elements within the same task. Alternatively, you can use "|" (OR) in your regular expressions. For example, you can use the following expression in "class" to export all Web service gateways and XML firewalls: "(WSGateway|XMLFirewall.*)".

Name	Description	Required
class	Regular expression defining what classes (types) to export, e.g., "WSGateway". If "class" is omitted, all classes will be matched.	No

name	Regular expression defining objects to be exported. Only the objects whose name matched this regexp will be exported. If "name" is omitted, all object names will be matched.	No
refObjects	If set to "true", include all objects referenced/required by this object. Defaults to "true" (DataPower default).	No
refFiles	Include all files referenced by this object. Defaults to "true" (DataPower default).	No
includeDebug	Include debug information in the export. Defaults to "false" (DataPower default).	No

"transform" Nested Element

"transform" contains the transformation actions documented in the section Using DPBuddy to Transform XML Files. This element is optional. It can only be specified if exporting a configuration in XML format.

"namePattern" Nested Element

The "namePattern" element provides an alternative to specifying object name patterns using the "namePatterns" attribute. It is useful when a regular expression pattern contains a comma, which is used as a delimiter in the "namePatterns" attribute.

You can specify multiple "namePattern" elements within the same task.

Note: "namePattern" element is deprecated. Use "exportObject" nested element instead.

Name	Description	Required
pattern	Export object with names matching this regular expression.	Yes

Examples

The following example exports all objects with the name starting with "testService" and of the classes with the names starting with "WSG" (which would match "WSGateway").

```
dpbuddy export -file services.zip -classPattern="WSG.*" -
namePattern="testService.*"
```

```
<dp:export file="${wsproxy.file}" allFiles="false" >
    <exportObject class="WSG.*" name="testService.*" />
</dp:export>
```

You can find more examples under "samples" in your distribution or online.

"save"

The "save" task saves domain configuration. This is equivalent to invoking "Save Config" from WebGUI.

Note that if the Ant property dp.auto.save is set to "true", the configuration will be saved automatically upon the completion of all tasks that make configuration changes (e.g., "import").

Examples

```
dpbuddy save
```

```
<dp:save/>
```

Tasks/Commands for Working with Files and Directories

All tasks in this category support specifying remote paths (paths to files or directories on the device), with or without the filesystem in the form of "filesystem:/". If the filesystem is not provided, DPBuddy will use "local:/".

There is no support for relative remote paths. In other words, "dir1/dir2" is the same with "/dir1/dir2".

"copy"

The "copy" task uploads a file or a set of files to a DataPower device from a local file system. "copy" is capable of uploading multiple files at once.

"copy" also creates the directory structure containing the files similar to the standard Ant copy task. "copy" automatically creates necessary directories on the device to reproduce the local directory tree.

"copy" relies on the nested "dpFileset" element(s) to determine what files and directories to create and upload.

Attributes/Options

Name	Description	Required
toDir	Specifies the target directory for uploaded files on the device. It could include a filesystem which is defined using "filesystem:/"	No

	syntax. If not provided, the filesystem will default to "local:/" You can override "toDir" at the fileset level.	
cleanDirectories	If "true", delete target directories on the device before uploading files. Root directories (i.e., "local:/") will not be cleaned. Use the delete task with patterns to remove all files from "local:/". Defaults to "false".	No
flatten	If "true", do not create any subdirectories; create only root directories on the device. Defaults to "false".	No
flushCache	If set to "true", flush the document and stylesheet caches for all XML managers after all files are copied. Defaults to "false".	No
transformOnly	If set to "true", DPBuddy will transform the files according to the nested "transform" element, but it will not copy the files to the device. In this mode the task does not connect to DataPower. This is convenient for developing/troubleshooting "transform" actions. Defaults to "false".	No
dir	The root of the directory tree of the local files; use it in conjunction with the "includes" attribute. Defaults to "./".	No
includes	Comma- or space-separated list of patterns of files that must be included in Ant pattern format. DPBuddy implicitly creates a nested fileset using the include patterns and the value from the "dir" attribute. If you need to specify "exclude" or other parameters of a fileset, you should explicitly define a nested "dpFileset".	

"dpFileset" Nested Element

"dpFileset" defines what files on the local file system will be uploaded to the device. "dpFileset" supports all attributes and nested elements of the regular Ant "fileset". In addition, you can specify the "toDir" attribute, which defines the root directory (and, optionally, the file system) on the device.

"dpFileset" also supports nested "transform" element explained in the next section.

If "toDir" at the fileset and task levels are omitted, DPBuddy will use the "local:/" filesystem and the base directory of the fileset (the directory specified by the "dir" attribute of the fileset) as the root on the device.

If "flatten" is set to "true", DPBuddy will upload all files into the root directory.

You can specify multiple "dpFileset" elements within the same task.

Name	Description	Required
toDir	Specifies the target directory for uploaded files on the device. It could include a filesystem which is defined using the "filesystem:/" syntax. If not provided, the filesystem will default to "local:/" You can specify default "toDir" at the task level.	No
failIfEmpty	If set to "true", fail if no files matched this fileset. Defaults to "true". Note that this behavior is different from the regular Ant "fileset".	No

"transform" Nested Element

"transform" contains transformation actions documented in the section Using DPBuddy to Transform XML Files. This element is optional. It must be nested within the "dpFileset" element.

These transformation actions apply to all matching files. Use multiple "dpFileset" elements if the transformations should be applied to only a subset of the files being copied.

Note that transformation actions can be applied only to XML files. All non-XML files should be specified in a separate "dpFileset" that does not have any transformations.

Examples

Suppose there is a local directory "services/person/wsdl" containing some wsdl and xsd files.

The following command will create "local:/apps/services/person/wsdl" directory tree on the device and upload the files. Since "cleanDirectories" is set to "true", it will delete the directory "local:/apps/services" before uploading the files.

```
dpbuddy copy -toDir "/apps/services" -includes "**/*.wsdl **/*.xsd" -
cleanDirectories
```

```
<dp:copy cleanDirectories="true">
    <dpFileset toDir="/apps/services"
               dir="services" includes="**/*.wsdl **/*.xsd"/>
</dp:copy>
```

You can find more examples under "samples" in your distribution or online.

"delete"

This task deletes files and directories on the device. All files and directories matching the "include" regexp pattern are deleted.

Run this task in "dry run" mode first to determine what is actually going to be deleted.

Attributes/Options

Name	Description	Required
include	Regular expression pattern defining what files and directories to delete. The match is done against the entire path of a file, including the filesystem (e.g., "local:/"). Note that the filesystem is separated by a single slash, i.e., the pattern "local://.*xsd" will not match anything. You cannot delete the root of the filesystem, though it is possible to delete everything under the root with the pattern "local:/(?!ondisk$).*". Replace "ondisk" with the mountpoint of your disk array if it is not the default.	Yes
matchRequired	If set to "true", fail if no files matched the "include" pattern. Defaults to "true".	No

dryRun	If set to "true", print the files and directories that will be deleted but don't make any changes to the device. Defaults to "false".	No

Examples

Delete all "*.xsd" files from "local:/":

```
dpbuddy delete -include="local:/.*\.xsd"
```

```
<dp:delete include="local:/.*\.xsd" />
```

You can find more examples under "samples" in your distribution or online.

"mkdir"

This task creates one or multiple directories on the device. If a directory path is specified, the task will create all parent directories that don't exist.

Note that the copy task creates all directories automatically so this task does not need to be explicitly invoked.

Use nested "dir" elements to create multiple unrelated directories. All directories will be created as part of a single SOMA request to the device.

Attributes/Options

Name	Description	Required
dir	A directory or a path to create. Could include a filesystem which is defined using "filesystem:/" syntax. If not provided, the filesystem will default to "local:/"	Yes

"dir" Nested Element

Name	Description	Required
path	A directory or a path. Could include a filesystem which is defined using "filesystem:/" syntax. If not provided, the filesystem will default to "local:/"	Yes

Examples

```
dpbuddy mkdir -dir="/dir1/dir2"
```

```
<dp:mkdir dir="/dir1/dir2/dir3"/>
```

"rmdir"

This task deletes directories on the device. This task is slightly faster than the delete task since it does not need to make a query to determine what to delete.

Use nested "dir" elements to delete multiple unrelated directories. All directories will be deleted as part of a single SOMA request to the device.

Attributes/Options

Name	Description	Required
dir	A directory or a path on the device. Could include a filesystem which is defined using 'filesystem:/' syntax. If not provided, the filesystem will default to 'local:/'	Yes
failOnError	Fail if the directory doesn't exist. Defaults to "true".	No

"dir" Nested Element

Name	Description	Required
path	A directory or a path. Could include a filesystem which is defined using "filesystem:/" syntax. If not provided, the filesystem will default to "local:/"	Yes

Examples

```
dpbuddy rmdir -dir="/dir1/dir2" -failOnError false
```

```
<dp:rmdir dir="/dir1/dir2" failOnError="false"/>
```

"download"

This task downloads files from the device to the local file system.

The task downloads all files that matched the provided regexp pattern. Unless the "flatten" attribute is set to "true", the task will recreate the remote directories containing the files locally. Empty directories will not be created.

Attributes/Options

Name	Description	Required
include	Regular expression pattern defining what files to download. The match is done against the entire path of a file, including the filesystem (e.g., "local:/"). Note that the filesystem is separated by a single slash, i.e., the pattern "local://.*xsd" will not match anything.	Yes
toDir	Local directory where to save the files.	Yes
flatten	If set to "true", download all files into "toDir" instead of creating the directory tree replicating remote directory structure. Defaults to "false".	No
matchRequired	If set to "true", fail if no files matched the "include" pattern. Defaults to "true".	No
cleanToDir	If set to "true", remove all files and subdirectories in "toDir" prior to downloading the files. Defaults to "false".	No
dryRun	If set to "true", print the files that will be downloaded without downloading anything. Defaults to "false".	No

Examples

Download all *.xsd files from "local:/":

```
dpbuddy download -include "local:/.*xsd" -toDir download
```

```
<dp:download include="local:/.*xsd" toDir="download"/>
```

You can find more examples under "samples" in your distribution or online.

"downloadFile"

This task downloads a single file from the device. This task is slightly faster than the "download" task since it does not need to make a query to determine which files to download. Use this task to download a single file and use "download" to download multiple files.

Attributes/Options

Name	Description	Required
file	Fully qualified path of the file on the device. If the filesystem in the form of "filesystem:/" is not specified, it will default to "local:/".	Yes
to	Local directory or file name where the remote file will be saved. If this is an existing directory, DPBuddy will preserve the file name. Otherwise, DPBuddy will use the value of "to" as the new file name.	Yes

Examples

This task will save the file to the "dpconfigs" directory:

```
<dp:download file="pubcert://American-Express-Global-CA.pem" to="dpconfigs" />
```

Tasks/Commands for Dealing with Configuration Checkpoints

"checkpoint"

The "checkpoint" task creates a configuration checkpoint with the given name. It is equivalent to using the "Administration/Configuration/Configuration Checkpoint" screen of WebGUI. If the checkpoint with the provided name already exists, it is automatically deleted (this is different from WebGUI's behavior). Note that DataPower can accommodate only a limited number of checkpoints in a domain.

If the checkpoint's name contains dots, they will be replaced with underscores. This makes it easier to accommodate version numbers in the checkpoint name.

Attributes/Options

Name	Description	Required
name	Name of the checkpoint	Yes
appendTimestamp	If true, append timestamp to the checkpoint name. Use with caution as this could quickly create a large number of checkpoints. The timestamp has the format "yyyyMMdd_HHmmss". Defaults to "false".	No

Examples

```
dpbuddy checkpoint -name release-1.0.1  -save
```

```
<dp:checkpoint name="release-1.0.1" save="true"/>
```

"rollback"

This tasks reverts the domain configuration to the specified checkpoint or to the latest checkpoint. Note that the content of the local file system will also be restored using the files that resided on the "local:/" at the time of the checkpoint.

Attributes/Options

Name	Description	Required
name	Name of the checkpoint to which to rollback. Defaults to the latest checkpoint in the domain. If the domain does not have any checkpoints, an exception will be raised.	No

Examples

```
dpbuddy rollback -save
```

```
<dp:rollback save="true"/>
```

"delCheckpoint"

This task deletes configuration checkpoints. Since DataPower can accommodate only a limited number of checkpoints, it may be necessary to delete old checkpoints from time to time.

The default number of checkpoints for a domain is 3. You can increase this limit from the domain configuration screen in WebGUI.

Attributes/Options

Name	Description	Required
name	Name of the checkpoint to remove.	At least one is required
namePattern	Regular expression pattern defining checkpoints to remove. Use ".*" to delete all checkpoints.	
matchRequired	If set to "true", fail if no checkpoints matched the pattern specified in "namePattern". Defaults to "true".	No

Examples

```
dpbuddy delCheckpoint -namePattern ".*1_0_1" -matchRequired false
```

```
<dp:delCheckpoint namePattern=".*1_0_1"/>
```

Tasks/Commands for Quiescence/Un-quiescence

Please refer to the DataPower InfoCenter for the description of the quiescence process.

"quiesce" and "unquiesce"

These tasks quiesce/unquiesce services and objects defined by the nested "object" element. This is equivalent to navigating to each "quiescable" object (such as an HTTP front side handler) in WebGUI and clicking on the "Quiesce" or "Unquiesce" link.

If you provide a name of a service, such as the name of a Web services proxy, DataPower will automatically quiesce/unquiesce all relevant handlers used by this service.

The tasks wait for the completion of the quiesce/unquiesce operation unless the "timeout" attributed is set to -1. The tasks will poll the device until all specified objects reach the "down" operational state for quiesce and "up" state for unquiesce.

You can specify objects to quiesce/unquiesce using classPattern/namePattern attributes and/or using the nested "object" element.

Attributes/Options

Name	Description	Required
classPattern	Regular expression defining what classes (types) to quiesce/unquiesce.	No

namePattern	Regular expression defining object names to quiesce/unquiesce.	No
timeout	Time, in seconds, to wait for all objects to reach the desired operational state. Exception will be raised if at least one object remains in the invalid state after the timeout. -1 disables waiting and device polling. Defaults to 60 seconds for quiesce, 15 for unquiesce.	No

"object" Nested Element

Each object element must match at least one DataPower configuration object.

Attribute	Description	Required
class	Regular expression defining what classes (types) to quiesce/unquiesce.	At least one is required
name	Regular expression defining object names to quiesce/unquiesce.	

Examples

Quiesce all front-side HTTP handlers whose name starts with "Test".

```
dpbuddy quiesce -classPattern "HTTPSourceProtocolHa.*"  -namePattern
"Test.*"
```

```
<dp:quiesce>
    <object class="HTTPSourceProtocolHa.*" name="Test.*"/>
</dp:quiesce>
```

You can find more examples under "samples" in your distribution or online.

"quiesceDomain" and "unquiesceDomain"

These tasks quiesce or unquiesce a single domain. Upon this command, DataPower will automatically quiesce or unquiesce all protocol handlers running in the domain.

The domain name is provided using the "domain" attribute or dp.domain" property.

You can also provide the timeout attribute explained earlier.

Examples

```
<dp:quiesceDomain domain="testDomain" />
```

```
dpbuddy unquiesceDomain -domain testDomain
```

Tasks/Commands for Checking DataPower Status

"status"

The "status" task retrieves the status of various parameters of a device and prints it to standard out. The status is displayed in the form of "name: value" where "name" is the name of the parameter.

DataPower groups status parameters into "classes" (not to be confused with the classes of DataPower objects). Each status class is responsible for a certain characteristic of the device, such as memory, CPU utilization and so on.

A complete list of status classes can be found under the “StatusEnum” type in xml-mgmt.xsd.

The most useful classes include "MemoryStatus" and "FilesystemStatus". These classes give information about available RAM and disk space. "ObjectStatus" can be used to find out the status of all DataPower objects.

Name	Description	Required
class	Status "class" as defined in xml-mgmt.xsd.	Yes

Examples

```
dpbuddy status -class MemoryStatus
```

```
<dp:status class="ObjectStatus" />
<dp:status class="MemoryStatus" />
<dp:status class="FilesystemStatus" />
```

"serviceStatus"

"serviceStatus" prints the list of active services and their port numbers. This is the same information that is available from WebGUI under "Status"/"Main"/"Active Services".

"serviceStatus" is also capable of querying multiple domains and providing information about services running in each domain.

Attributes/Options

Name	Description	Required
domainPatterns	Comma-delimited list of regular expression patterns defining which domains to query. To query all domains use ".*" Defaults to the current domain. The current domain is specified using "dp.comain" property or "domain" attribute of the task.	No

Examples

```
dpbuddy serviceStatus -domainPatterns ".*"
```

```
<dp:serviceStatus domainPatterns=".*"/>
```

"assertStatus"

"assertStatus" can be used to validate ("assert") the status of various DataPower objects. The "status" task documented earlier simply reports the status. "assertStatus" raises an exception when some parameters don't meet the expectations.

For example, you may want to check that there is enough free memory (RAM) on the device before performing a deployment/import. You can run "assertStatus" and check that the "FreeMemory" parameter of the "MemoryStatus" class is above a certain threshold.

The condition for the assertion is defined using a Groovy Boolean expression. In this expression, you can also refer to any of the parameters of the returned status as Groovy variables. For example, "FreeMemory" is one of the parameters returned in response to querying "MemoryStatus". Your Groovy expression can simply compare the MemoryStatus variable to a certain memory threshold: "FreeMemory>=${dataPowerMemoryThreshold}".

Run "assertStatus" with expression set to "true" first to find out what parameters are returned for each status class. For example, <dp:assertStatus class="MemoryStatus" expression="true" /> will print a line similar to this:

Usage: 25, TotalMemory: 3368389, UsedMemory: 870929, FreeMemory: 2497460, ReqMemory: 903824, HoldMemory: 32895, ReservedMemory: 689911, InstalledMemory: 4058300

Your Groovy expression can reference any of the names in this output as variables.

Attributes/Options

Name	Description	Required
class	Status "class" as defined in xml-mgmt.xsd.	Yes
expression	A Groovy Boolean expression (has to return true or false). All parameters returned in response to querying the status class are also available as Groovy variables. In addition to this, there is a special "dp" variable containing the following properties: "url", "username", and "domain". These properties are populated with the values for the device/domain against which the task is being executed. If the expression returns "false", the task will raise an exception.	Yes

Examples

```
dpbuddy assertStatus -class="LoadBalancerStatus2" -
expression="(Group!='test-group' || Health=='up')"

<property name="dpMemoryThreshold" value="3000000"/>
<dp:assertStatus class="MemoryStatus"
expression="FreeMemory&gt;=${dpMemoryThreshold}"/>
```

"assertFreeSpace"

"assertFreeSpace" is a convenience task that checks that there is enough free disk space on the device. Internally the task utilizes "assertStatus" task described earlier.

Attributes/Options

Name	Description	Required
minFreeSpace	Minimally acceptable free disk space in MB. The task will raise an exception if the free space is below this value.	Yes

Examples

```
<dp:assertFreeSpace minFreeSpace="15000" />
```

You can find more examples under "samples" in your distribution or online.

"assertState"

"assertState" checks the operational state of DataPower objects determined by the nested "object" elements. The task raises an exception if at least one object is not in the desired state ("up" or "down"). The task excludes disabled objects from the check.

For every object in the "down" state, the task will retrieve and display the five most recent log entries for that object, to facilitate troubleshooting.

The task should be run after "import" to ensure that all objects involved in the import operation were created/updated successfully. It is possible for an import operation to succeed while leaving objects/services in the "down" state.

If a nested "object" element is not provided, "assertState" will validate the state of all objects in the domain.

Attributes/Options

Name	Description	Required
classPattern	Regular expression defining what classes (types) to check.	No
namePattern	Regular expression defining object names to check.	No
state	Operational state to check. Allowed values are "up" or "down". An error is raised if at least one object is not in this state. Defaults to "up".	No
activeService	If set to "true", invoke the "assertActiveService" task to check that all objects defined using nested "object" elements are present in the active services list. Defaults to "false".	No

"object" Nested Element

Each object element must match at least one DataPower configuration object.

Attribute	Description	Required
class	Regular expression defining what classes (types) to check.	At least one is

name	Regular expression defining object names to check.	required

Examples

Verify that all Web services proxies and XML firewalls are "up" and listening for requests:

```
dpbuddy assertState -classPattern "(WSGateway|XMLFi.*)" -activeService
```

```
<dp:assertState opState="up" activeService="true">
    <object class="(WSGateway|XMLFi.*)" />
</dp:assertState>
```

"assertActiveService"

"assertActiveService" checks if the services defined using nested "object" elements are actually running and listening for requests. It is possible for a service, such us a Web services proxy, to be in the "up" operational state (and pass the "assertState" check) while still not running/accepting requests. All objects specified using nested "object" elements have to be listed in the active services list returned by the "serviceStatus" task (this list can also be obtained from Status/Main/Active Services in WebGUI).

The task also prints the list of active services and their ports.

"object" Nested Element

Each object element must match at least one DataPower configuration object.

Name	Description	Required
class	Regular expression defining what classes (types) to check.	At least one is required
name	Regular expression defining object names to check.	

Examples

Verify that all Web services proxies and XML firewalls are listening for requests:

```
<dp:assertActiveService>
    <object class="(WSGateway|XMLFi.*)" />
</dp:assertActiveService>
```

"assertOpenPorts"

"assertOpenPorts" checks if the ports specified using the "ports" attribute are assigned to active services. Note that this task does not attempt to actually open the ports; instead it obtains the

list of active services from the device using the serviceStatus task and checks that all the provided ports are in that list. The task will raise an exception if this is not the case.

The task also prints the list of active services and their ports.

Attributes/Options

Name	Description	Required
ports	A comma-delimited list of ports to check.	Yes

Examples

Verify that the default WebGUI and XML management ports are open:

```
dpbuddy assertOpenPorts -ports="9090, 5550" -domain default
```

```
<dp:assertOpenPorts ports="9090, 5550" domain="default" />
```

Tasks/Commands for Working with DataPower Logs

"tailLog"

The "tailLog" task retrieves log entries from a device and prints them to standard output. The task prints the last 48 lines of the log by default.

You can also use the task to save the contents of the DataPower log file locally.

"tailLog" is capable of continuously querying the device and identifying new entries based on timestamps. This works similarly to "tail -f" Unix command.

You can also use "tail" alias from DPBuddy CLI.

"tailLog" can check log entries for errors based on regular expressions. When a log entry contains an occurrence of such an expression, "tailLog" will raise an exception. This allows for using "tailLog" for monitoring DataPower devices, especially in combination with running "tailLog" continuously.

You can also pipe Ant running the "tailLog" target with "grep" to search for specific strings in the log.

Attributes/Options

Name	Description	Required
lines	Number of the most recent log entries to	No

	display. "-1" directs the task to print or save all available log entries. Defaults to 48 lines.	
logTarget	Name of the log target defined on the device. Defaults to "default-log" (DataPower default).	No
domainPatterns	Comma-delimited list of regular expressions specifying domains to get logs from. Log entries from all matching domains are combined together and sorted by their timestamps. Use "domainPattern" nested elements if there is a need to use commas inside the regexp. Defaults to current domain. The current domain can be specified using "dp.domain" Ant property or the "domain" attribute of the task.	No
format	Format string. See the Log Entry Format section.	No
failOnError	If set to "true", fail whenever an error-level log entry is encountered. Defaults to "false".	No
failPatterns	List of comma-delimited regular expression patterns. "tailLog" will raise exception if it finds one of the patterns in the log entry. Patterns are applied to the entire formatted log entry string containing all fields.	No
follow CLI alias: -f	If set to "true", query the device continuously every 3 seconds or according to the interval specified in "followInterval". New log entries (determined based on their	No

	timestamp) are appended to standard output. Defaults to "false".	
followInterval	Interval in milliseconds used to continuously query the device if "follow" is set to "true". This attribute is ignored if "follow" is "false". Defaults to 3,000 milliseconds.	No
logFile	Local file to which the DataPower log file should be saved.	No
appendTimestamp	When logFile is set, append the timestamp to the local log file name. Defaults to "false".	No

Log Entry Format

The "tailLog" task uses java.text.MessageFormat class to format DataPower log entries for display. The default format is "{1,date,yyyy-MM-dd HH:mm:ss} |{2}| {0}{3}". The format string uses numeric IDs for various log fields. tailLog supports the following fields:

- 0: log message
- 1: timestamp
- 2: severity level. "tailLog" prints 'E' for errors, 'W' for warning and 'I' for information-level messages.
- 3: DataPower object name
- 4: transaction ID
- 5: domain name

For example, you can use the following format string to display the domain name:

{5} | {1,date,yyyy-MM-dd HH:mm:ss} |{2}| {0}{3}

"where" Nested Element

"tailLog" can filter log entries received from the device so that only the ones matching the criteria specified in the "where" nested element will be printed.

Note that class names used in "where" are different from the ones used by "export", "delConfig", "quiesce" and other tasks that support nested "object" elements. The class names used in DataPower logs are different from class names of configuration objects. For example,

"wsgw" is how Web services gateway type is referenced in the logs. The configuration class of the same type is "WSGateway".

You can specify multiple "where" elements within the same task.

Name	Description	Required
class	Regular expression defining classes of log entries. Only log entries with matching classes will be printed. To find out class names, navigate to the object you'd like to print log entries for in WebGUI and click on "View log". A log message usually starts with the prefix in the format <class>(<object>), e.g., "wsgw (testServiceProxy):". If not specified, log entries will be printed regardless of classes.	At least one is required
name	Regular expression defining object names of log entries. Only the log entries with the matching objet names will be printed. If not specified, log entries will be printed regardless of object names.	

"domainPattern" Nested Element

The "domainPattern" nested element provides an alternative to specifying domain patterns in the "domainPatterns" attribute. It is useful when a regular expression contains commas, which are used as a delimiter in the "domainPatterns" attribute.

You can specify multiple "domainPattern" elements within the same task.

Name	Description	Required
pattern	Get logs from the domains matching this regexp pattern.	Yes

Examples

The following example collects log entries from system logs in the "default" domain and all domains starting with "dev". It displays the last 100 lines of the combined log.

```
dpbuddy tail -domainPatterns "dpbuddy-.*, e2e.*"  -lines 100
```

```
<dp:tailLog domainPatterns="default, dev.*"  lines="100" />
```

The following task automatically retrieves new log entries until it encounters '|E|' or '|W|' anywhere in a log entry:

```
dpbuddy tail -f -failPatterns "\|[E|W]\|"
```

```
<dp:tailLog failPatterns="\|[E|W]\|" follow="true" >
```

```
dpbuddy tail -f | grep  "\|[E|W]\|"
```

The following example logs only the entries for all Web services gateways starting with "testService" and all XML firewalls.

```
<dp:tailLog failOnError="false" >
    <where class="wsgw" object="testService.*" />
    <where class="xmlfire.*" />
</dp:tailLog>
```

You can find more examples under "samples" in your distribution or online.

"log"

"log" creates a log entry in the DataPower log on the device. Logging a message in the device log can be useful for audit and troubleshooting purposes. For example, you can create a log entry before applying a set of changes to the device.

Attributes/Options

Name	Description	Required
message	Message to log.	This attribute or nested text.
level	Log entry's severity level, one of the following: EMERG, ALERT, CRITIC, ERROR, WARN, NOTICE, INFO, DEBUG Defaults to WARN.	No
category	DataPower log category. You can find the list of all categories under the Administration/Miscellaneous/Configure Log	No

	Category in WebGUI. Defaults to "all".	

Examples

```
dpbuddy log -level info -category all  -message="Hello from DPBuddy"
```

```
<dp:log>
    Hello from DPBuddy!
</dp:log>
```

"setLogLevel"

This task sets the log level of the default log category ("all"). This is equivalent to navigating to "Manage Log Targets"/"Event Subscriptions" in WebGUI and editing the minimum event priority for "all" category.

Attributes/Options

Name	Description	Required
level	Log entry's severity level, one of the following: EMERG, ALERT, CRITIC, ERROR, WARN, NOTICE, INFO, DEBUG	Yes

Examples

```
dpbuddy setLogLevel -level DEBUG
```

```
<dp:setLogLevel level="DEBUG" />
```

Tasks/Commands for Flushing DataPower Cache

DataPower caches stylesheets, XML documents and various other items. There is often a need to flush the cache. For example, the stylesheet/XSLT cache should be flushed after XSLT files were updated on the file system. DPBuddy provides several tasks to flush DataPower caches.

"flushStylesheetCache"/"flushDocumentCache"/"flushXMLCache"

These tasks flush XSLT and/or XML document caches:

- "flushStylesheetCache" flushes the XSLT cache.
- "flushDocumentCache" flushes the XML document cache.
- "flushXMLCache" flushes both the XML document and XSLT caches.

Attributes/Options

Name	Description	Required
xmlManager	Regular expression matching the names of XML managers that should be flushed. Exception is raised if no matching XML manager was found. Defaults to ".*" (matches all XML managers).	No

Examples

Flush the document cache of the default XML manager.

```
dpbuddy flushDocumentCache -xmlManager "def.*"
```

```
<dp:flushXMLCache xmlManager="def.*"/>
```

You can find more examples under "samples" in your distribution or online.

"flushAAACache"

This task flushes the AAA policy cache.

Attributes/Options

Name	Description	Required
policy	Regular expression matching the names of AAA policies that should be flushed. Exception is raised if no matching AAA policy was found. Defaults to ".*" (matches all AAA policies).	No

Examples

Flush "test" AAA policy.

```
<dp:flushAAACache policy="Test.*"/>
```

"flushMiscCache"

This task allows you to flush the following DataPower caches: DNS, RBM and ND (neighbor discovery).

Attributes/Options

Name	Description	Required
cacheTypes	Comma-delimited list of cache types. A cache type must be one of the following: DNS, RBM or ND Flushing the RBM cache removes all cached user names and passwords from memory.	Yes

"flushAllCache"

This task invokes "flushXMLCache", "flushAAACache" and "flushMiscCache" tasks.

"flushXMLCache" and "flushAAACache" are invoked with ".*" regexp to flush all XML managers and AAA policies. "flushMiscCache" is invoked with the RBM and DNS cache types.

Backup Tasks/Commands

"backup"

"backup" exports files and all configuration objects from one or multiple domains.

The function provided by this task is equivalent to that of the "Create a backup of the entire system" or "Create a backup of one or more application domain" options of the "Export configuration" function of WebGUI.

"backup" does not export user accounts, certificates or keys. Use secure backup if you need to include certificates and keys.

The backups are saved as zip files on the local file system.

Note that, unlike "export", you can restore the output of "backup" only into the domain the backup was taken from.

"backup" only backs up the files located in the "local:/" filesystem.

Use "backup" when you need to export/backup multiple domains. In most other cases, "export" command/task should be preferred as it provides more flexibility.

Attributes/Options

Name	Description	Required

file	Name of the zip file where the backups should be saved. The resulting zip file will contains individual zip files for each domain. The task will create directories to save the file to if they don't exist.	Yes
appendTimestamp	If true, automatically append timestamp to the file name. The timestamp has the format "yyyyMMdd_HHmmss". Defaults to "false".	No
domainPatterns	Comma-delimited list of regular expression patterns defining which domains to back up. To backup all domains simply use ".*" Use "domainPattern" nested elements if there is a need to use commas inside the regexp. Defaults to current domain. The current domain is specified using the "dp.comain" property or "domain" attribute of the task.	No
persisted	If set to "true", backup only the persisted domain configuration. Defaults to "false".	No
deploymentPolicyFile	Path to the deployment policy file on the local filesystem. The deployment policy is automatically uploaded/configured on the device before the request is executed. The deployment policy will be embedded with the export file.	No
deploymentPolicyName	Name of the deployment policy to include with the export/backup. The policy has to already be defined on the device.	No

"domainPattern" Nested Element

"domainPattern" provides an alternative to specifying domain patterns in the "domainPatterns" attribute. It is useful when a regular expression pattern contains commas, which are used as a delimiter in the "domainPatterns" attribute.

You can specify multiple "domainPattern" elements within the same task.

Attribute	Description	Required
pattern	Backup domains matching this regexp pattern.	Yes

Examples

The following example backs up all domains with the name starting with "dev". The timestamp will be automatically added to the file name.

```
dpbuddy backup -file backups/backup.zip -domainPatterns "dev.*" -
appendTimestamp
```

```
<dp:backup file="backups/backup.zip" domainPatterns="dev.*"
appendTimestamp="true" />
```

"secureBackup"

This task performs secure backup of the device. Unlike regular backup, secure backup includes certificates, keys and user passwords. You can find more details about secure backups in InfoCenter or in this developerWorks article.

"secureBackup" first creates secure backup files on the appliance and then automatically downloads all the files to a local directory specified using the "toDir" attribute. Creating backup files may take several minutes depending on the size of your filesystems.

Secure backups are always performed in the default domain.

Note that secure backup files may be quite large. DPBuddy always streams downloaded files directly to disk, so it is able to download files of any size. However, DataPower may not be able to stream files, so it may take a few minutes to prepare the file on the device (it has to encode the file using base64 encoding) before downloading can begin.

Attributes/Options

Name	Description	Required
cert	Name of the crypto certificate containing the public key used to encrypt the secure backup.	Yes

	You must create the crypto certificate before using this task. The crypto certificate can be created from WebGUI by navigating to Objects/Crypto Configuration/Crypto Certificate.	
toDir	Local directory where the backup files should be saved. The directory will be created if it doesn't exist.	Yes
appendTimestamp	If true, automatically append timestamp to the directory name specified in "toDir". The timestamp has the format "yyyyMMdd_HHmmss". Defaults to "false".	No
includeISCSI	If set to "true", the iSCSI device will be included in the backup. Defaults to "true" (DataPower default).	No
includeRAID	If set to "true", the RAID device will be included in the backup. Defaults to "true" (DataPower default).	No
dpDir	DataPower directory for storing secure backup files. This could be any temporary directory. Defaults to "temporary:/secure-backup".	No

Examples

Create secure backup and download the backup files to "testfiles/sbackup":

```
dpbuddy secureBackup -cert crypto-backup-test -toDir testfiles/sbackup
-appendTimestamp -dpDir "temporary:/secure-backup"
```

```
<dp:secureBackup cert="crypto-backup-test" toDir="testfiles/sbackup"
    includeISCSI="false" includeRAID="false" appendTimestamp="true" />
```

Tasks/Commands for Resetting/Restarting Domains

Both of the tasks explained in this section run against the domain provided by the "domain" attribute or "dp.domain" property.

"restartDomain"

This task restarts the domain.

Example

```
dpbuddy restartDomain -domain default
```

```
<dp:restartDomain domain="default"/>
```

"resetDomain"

This task resets the configuration of the domain. It deletes all configuration objects in the domain. The task does not affect files.

Example

```
dpbuddy resetDomain
```

```
<dp:resetDomain/>
```

Miscellaneous Tasks/Commands

"action"

This task allows you to extend DPBuddy with various DataPower administration functions which are not implemented by other DPBuddy tasks.

A complete list of available actions can be found in the xml-mgmt.xsd file, under the type "AnyActionElement". Documentation for actions can be found in the DataPower command reference documentation under "Global mode" and "Initial login and common commands". Note that many actions can be executed using other DPBuddy tasks. For example, DPBuddy provides dedicated tasks for flushing caches, so there is no need to know how to do it using "action".

To execute actions/commands without parameters, simply provide the action's name using the "name" attribute of the "action" task.

For example, the SOMA counterpart for "save error-report" command is "ErrorReport" action. This action can be executed as follows:

```
<dp:action name="ErrorReport" />
```

Most actions, however, take some parameters. For these actions you need to provide a nested XML fragment with all the necessary XML elements for the action. The XML fragment can be nested directly inside the "action" task.

Attributes/Options

Name	Description	Required
name	Name of SOMA action as per the DataPower XML schema.	No

Examples

The "ping" action (see ping command documentation) requires "RemoteHost" as a parameter.

"ping" invocation could be encoded using the following XML fragment:

```
<dp:action>
    <Ping>
        <RemoteHost>${ping.host}</RemoteHost>
    </Ping>
</dp:action>
```

You can find more examples under "samples" in your distribution or online.

"somaRequest"

The "somaRequest" task executes an arbitrary SOMA request defined in an external file.

Do not specify SOAP envelope XML elements in the file; DPBuddy will add them automatically.

You can use Ant variables in any text node or in any attribute of the XML file.

This task will attempt to validate XML request against DataPower schema unless the "validate" attribute is set to "false".

Attributes/Options

Name	Description	Required
file	SOMA request file.	Yes
printResponse	If set to "true", print XML response from the device. The SOAP envelope is stripped out and not printed. Defaults to "false".	No

Examples

```
dpbuddy somaRequest -printResponse -file dpconfigs/ping-remote-host.xml
```

```
<dp:somaRequest file="ping-remote-host.xml" />
```

Here is the content of "ping-remote-host.xml":

```
<?xml version="1.0" encoding="UTF-8"?>
<dp:request xmlns:dp="http://www.datapower.com/schemas/management" >
    <dp:do-action>
        <Ping>
            <RemoteHost>${ping.host}</RemoteHost>
        </Ping>
    </dp:do-action>
</dp:request>
```

"wsrrSynchronize"

This task synchronizesWSRR content with the WSRR server. See the "wsrr-synchronize" command documentation for more details.

Attributes/Options

Name	Description	Required
subscription	Specifies the name of a WSRR subscription or a WSRR Saved Search subscription object. Content previously retrieved using this subscription is immediately synchronized with the WSRR server specified by the subscription.	Yes

Examples

```
<dp:wsrrSynchronize subscription="testSubscription" />
```

"testConnection"

This task accesses the device and retrieves and prints its firmware version. This task could be used to test connectivity with DataPower. The task does not have any attributes/options