



DPBuddy User Guide

Version 3.0





Contents

Introduction.....	8
What’s New in This Release	8
DPBuddy Installation	9
Configuring DataPower to Work with DPBuddy	9
Adding DPBuddy Tasks to Your Build	10
Upgrading from DPBuddy 2.3.....	11
DPBuddy License	11
Recommended Development and Build/Deployment Process.....	11
Developing DataPower Artifacts	12
Creating a Build in a Test Environment	12
Promoting the Build to a "Higher" Environment.....	13
Deploying to a Production Environment.....	14
Enabling Verbose Logging	14
Properties and Attributes Used by All Tasks	14
Connection Properties.....	14
Schema Validation Properties	15
Examples.....	16
Auto-Saving Property	17
License File	17
Use or Regular Expressions to Select DataPower Objects	18
Examples.....	18
Using DPBuddy to Transform DataPower Configuration and XML Files	19
"transform" Type.....	20



- Attributes Common to All Transform Actions 21
 - "setText" Transform Action 21
 - "replaceText" Transform Action 22
 - "add" and "update" Transform Actions 22
 - "delete" Transform Actions 22
 - "dpExclude" and "dpInclude" Transform Actions 22
 - "namespace" Nested Element 23
- Managing Environment-Specific Properties/Variables 23
 - "environment" Task 24
 - Attributes 24
 - Examples 25
- Auditing DPBuddy Tasks 25
 - Properties for Configuring Audit 27
- Tasks for Changing and Exporting Configuration 28
 - "import"/"buddyImport" Task 28
 - Attributes 29
 - Examples 30
 - "transformFiles" Nested Element 30
 - "transform" Nested Element 31
 - "modifyConfig" Task 31
 - Attributes 32
 - "transform" Nested Element 32
 - "fileset" Nested Element 33
 - "configuration" Nested Element 33
 - Examples 33
 - "delConfig" Task 33
 - Attributes 33
 - "object" Nested Element 34



- Examples..... 34
- "export" Task..... 34
 - Attributes..... 35
 - "exportObject" Nested Element..... 37
 - "transform" Nested Element..... 37
 - "namePattern" Nested Element..... 38
 - Examples..... 38
- Tasks for Working with Files and Directories 38
 - "copy" Task..... 39
 - Attributes..... 39
 - "dpFileset" Nested Element 40
 - "transform" Nested Element..... 40
 - Examples..... 40
 - "delete" Task 41
 - Attributes..... 41
 - Examples..... 42
 - "mkdir" Task 42
 - Attributes..... 42
 - "dir" Nested Element..... 42
 - Examples..... 43
 - "rmdir" Task..... 43
 - Attributes..... 43
 - "dir" Nested Element..... 43
 - Examples..... 43
 - "download" Task 43
 - Attributes..... 44
 - Examples..... 44
 - "downloadFile" Task..... 45



- Attributes..... 45
- Examples..... 45
- Tasks for Saving Configuration and Creating Checkpoints..... 45
 - "save" Task 38
 - Examples..... 38
 - "checkpoint" Task..... 45
 - Attributes..... 46
 - Examples..... 46
- Tasks for Quiescence/Un-quiescence 47
 - "quiesce" and "unquiesce" Tasks 47
 - Attributes..... 47
 - "object" Nested Element..... 48
 - Examples..... 48
 - "quiesceDomain" and "unquiesceDomain" Tasks..... 48
 - Examples..... 48
- Tasks for Checking DataPower Status 48
 - "status" Task..... 48
 - Examples..... 49
 - "serviceStatus" Task 49
 - Attributes..... 49
 - Examples..... 49
 - "assertStatus" Task..... 50
 - Attributes..... 50
 - Examples..... 51
 - "assertFreeSpace" Task 51
 - Attributes..... 51
 - Examples..... 51



- "assertState" Task 51
 - Attributes..... 51
 - "object" Nested Element..... 52
 - Examples..... 52
- "assertActiveService" Task 52
 - "object" Nested Element..... 52
 - Examples..... 53
- Tasks for Working with DataPower Logs..... 53
 - "tailLog" Task..... 53
 - Attributes..... 53
 - Log Entry Format 55
 - "where" Nested Element..... 55
 - "domainPattern" Nested Element..... 56
 - Examples..... 56
 - "log" Task..... 57
 - Attributes..... 57
 - Examples..... 57
 - "setLogLevel" Task..... 58
 - Attributes..... 58
 - Examples..... 58
- Tasks for Flushing DataPower Cache..... 58
 - "flushStylesheetCache"/"flushDocumentCache"/"flushXMLCache" Tasks..... 58
 - Attributes..... 58
 - Examples..... 59
 - "flushAAACache" Task 59
 - Attributes..... 59
 - Examples..... 59
 - "flushMiscCache" Task 59



- Attributes..... 59
- "flushAllCache" Task..... 60
- Backup Tasks 60
 - "backup" Task..... 60
 - Attributes..... 60
 - "domainPattern" Nested Element..... 61
 - Examples..... 62
 - "secureBackup" Task 62
 - Attributes..... 62
 - Examples..... 63
- Tasks for Resetting/Restarting Domains 63
 - "restartDomain" Task..... 63
 - Example 63
 - "resetDomain" Task..... 63
 - Example 63
- Miscellaneous Tasks 64
 - "action" Task..... 64
 - Attributes..... 64
 - Examples..... 64
 - "somaRequest" Task..... 65
 - Attributes..... 65
 - Examples..... 65
 - "wsrrSynchronize" Task..... 66
 - Attributes..... 66
 - Examples..... 66



Introduction

DataPower Buddy ("DPBuddy") is a tool for automating administration of IBM WebSphere DataPower appliances. The tool supports export, import, backup, file transfer and many other functions. DPBuddy is implemented as a set of custom tasks for [Apache Ant](#) build tool.

Once you [installed and configured DPBuddy](#), you can start defining your Ant target containing various DPBuddy tasks.

For example, an Ant target for copying files to your DataPower appliance may look like this:

```
<target name="upload.files"
  description="Upload other files to the local:/ filesystem of
the appliance">
  <dp:copy cleanDirectories="true" flatten="true">
    <!-- Destination directory can be specified at the fileset
level -->
    <dpFileset dir="${services.files.home}" toDir="/xslt"
includes="**/*.xslt"/>
    <dpFileset dir="${services.files.home}" toDir="/services/wsdl"
includes="**/*.wsdl **/*.xsd"/>
  </dp:copy>
  <!-- Flush both documents and stylesheets caches for all XML
managers -->
  <dp:flushXMLCache/>
</target>
```

You can find many more examples in your DPBuddy distribution under "samples" or [online](#). You can start by looking at [dp-build.deploy.xml](#) which implements an end-to-end build/deploy process using DPBuddy. "samples/ant-tasks" folder contains task-specific samples and examples.

DPBuddy uses 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](#), however, 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.

What's New in This Release

DPBuddy Version 3 is a major release with many changes and fixes. The notable new features include:

- [Enhanced XML transformation capabilities](#) that can be used with "copy", "import", "export" and other tasks.
- [Tasks for validating states and statuses](#) of DataPower services and objects.



- [Tasks for quiescence/un-quiescence](#).
- Tasks for [deleting files](#) and [downloading files](#).
- [Tasks for cache management](#).
- [Secure backup support](#).
- Many bug fixes.

DPBuddy Installation

DPBuddy requires Ant 1.8 or above and JDK 1.6 and above. All JDK 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 need large heap size so 64-bit JDK is recommended.

To install DPBuddy:

- Download and install Apache Ant as documented here: <http://ant.apache.org/manual/install.html>. Note that on many Linux systems you can install Ant using package manager, e.g., you can use "apt-get -u install ant" on Ubuntu. You should also set "[ANT_OPTS](#)" environment variable to increase the JVM's heap size. "-Xmx1648m" is the recommended setting. If you're planning to download or upload very large binary files, you may need to further increase the maximum size. We also recommend running Ant with "-e" option as it provides a much cleaner output. You can set this option globally by [setting ANT_ARGS](#) environment variable to "-e".
- Download DPBuddy from myarch.com/dpbuddy, unzip/untar the content of the distribution in a directory.

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 modifying during your build/deployment process.

If you're going to run DPBuddy behind a proxy, add the following Java system properties to the "ANT_OPTS" variable: -Dhttps.proxyHost=<your proxy host> -Dhttps.proxyPort=<your proxy port> or use [Ant setProxy task](#).

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 it from DataPower WebGUI (Network/Management/XML Management).



DPBuddy uses "old" DataPower v2004 message format for some of its tasks. This means that "SOAP Configuration Management (v2004)" checkbox must be checked on the XML management screen.

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 do 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", anything more detailed (e.g., "Information") results in overly verbose logging.

If you're creating a new account that will be used by DPBuddy, make sure that the password for this account is 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 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 "Properties and Attributes Used by All Tasks". In this case, you will need to add the DataPower certificate to your JDK's keystore.

Adding DPBuddy Tasks to Your Build

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>
    <fileset dir="${dpbuddy.home}/lib" />
  </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. In this case, you don't need to provide explicit "taskdef". Please see [Ant manual](#) for more details.



Upgrading from DPBuddy 2.3

If you're using DPBuddy 2.3, download/unzip the new version and then update your "taskdef" to point to the new version. This release is mostly backward compatible 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".
- "request" task was renamed to [somaRequest](#).
- "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.
- "dp.env.prefix" global property is no longer supported; please replace it with more flexible [environment](#) task.

DPBuddy License

All DPBuddy downloads are bundled with a 60 days 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/contact-us/>.

To install a permanent license that you received from MyArch, simply save the license file and add the property "dpbuddy.license.file" to your Ant build file. This property must point to the location of the license file.

Recommended Development and Build/Deployment Process

DPBuddy can support any build/deployment/release process, however, you will get 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 a 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, "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 to fully automate all steps described below, except for the testing step.

Developing DataPower Artifacts

You can have many domains used for development, e.g., you could setup 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 "copy" task. Target: upload.files.
- Conduct testing using JUnit, soapUI or other tools.
- Export configuration for their services using "export" task and check in the configuration under version control. We recommend checking in XML configuration files (as opposed to exported zip files) so that the version control can be used for tracking changes and diffing the files. Note that DPBuddy will automatically clean and format the exported XML configuration so it could be checked in right away. Target: export.services.

We recommend replacing environment-dependent values with variables during export. This can be accomplished by utilizing 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 (could be 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/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.
- Checkout 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 could be stored under version control and it can be imported using "import" task similar to any other DataPower service or object. Target: import.crypto.config.
- Upload files used by DataPower services and objects. This should include 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. We also recommend exporting the domain's configuration in XML format (without files) upon save. This configuration could be put under version control to create a complete history of all changes in a specific domain. Targets: verify.import, save.
- 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 the 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/create the domain or clean it. If you don't want to re-upload keys every time you deploy, it might be sufficient to just 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. Targets: verify.import, save.
- 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 (typically, two) 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.

Enabling Verbose Logging

DPBuddy prints detailed debug information when Ant is run in verbose mode ("ant -v"). This information includes SOAP/XML generated by DPBuddy and responses returned by DataPower.

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

If you're using [environment task](#), you could also set the property "dpbuddy.trace.properties" to "true".

Properties and Attributes Used by All Tasks

There is a number of attributes that are common to all DPBuddy tasks. These attributes are mostly related to DataPower connection parameters.

All common attributes have a corresponding Ant property. This allows for not repeating the same attribute for every task. A value from the attribute overrides a value from the property. For example, "dp.domain=test" property can be overridden using "domain='test1'" attribute set at a task level.

Every global 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 commonly used dot notation for accessing object properties.

DataPower Connection Properties and Attributes

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 name	Description	Required
---------------	----------------	-------------	----------



dp.url dpUrl	url	URL of the SOAP interface (XML Management interface), including protocol and port number. 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 port 5550.	Yes
dp.username dpUsername	username	Valid username with administrative privileges.	Yes
dp.password dpPassword	password	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

Schema Validation Properties and Attributes

DPBuddy can optionally validate SOMA XML requests against the DataPower SOMA XML schema. Several DPBuddy tasks, such as "action", "request" and "modifyConfig", allow for specifying free-form XML as part of the request. DataPower itself validates all XML requests, however, because of 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 device's logs, but scanning the logs takes time. DPBuddy, on the other hand, validates XML on the client so validation error messages are displayed right away.



The following DPBuddy properties and attributes control schema validation:

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

Examples

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



"Save" Property and Attribute

DPBuddy is capable of automatically saving domain configuration after an update. This is equivalent to clicking "Save Config" from WebGUI.

Most of DPBuddy's configuration changing tasks support "save" attribute. 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 "dp.auto.save" property. If this property is set, all the configuration-changing tasks will be saving domain configuration upon completion.

"save" attribute takes the precedence over "dp.auto.save" property.

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

You must explicitly invoke [save task](#) to persist domain configuration if "dp.auto.save" or "save" attribute is not set or set to "false".

Property name	Attribute name	Description	Required
dp.auto.save dpAutoSave	autoSave	If set to true, save domain configuration. Defaults to "false".	No

License File Property

Please see [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 name	Description	Required
dpbuddy.license.file dpbuddyLicenseFile	None	Location of the license file.	No



Use or Regular Expressions to Select DataPower Objects

Many DPBuddy tasks 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 admin UI.

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, [export task's](#) nested "exportObject" elements define what objects to export.

In all these cases you can use regular expressions to select configuration objects. DPBuddy will select 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 or based on just the object name or based on both. In 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.*" />
```



Using DPBuddy to Transform DataPower Configuration and XML Files

There is frequently a need to apply environment-specific modifications to 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 configuration. DPBuddy fully supports deployment policies and it also provides additional XML transformation capabilities. DPBuddy support many transformation actions that can be executed as part of "import", "export", "modifyConfig" or "copy" tasks. The actions include "add", "update", "delete", "setText", "replaceText", "include", "exclude". They are explained in details 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 matched the expression.

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

```
<dp:import file="{wsproxy.zip.file}">
  <transformFiles includes="export.xml">
    <transform>
      <setText xpath="//*[@name='TestHTTPHandler']/LocalPort"
        value="{ws.port}" />
    </transform>
  </transformFiles>
</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 can be triggered only for certain environments.



- Support Groovy expressions for complex transformations. E.g., you can apply some 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 devices. Both "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. 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 if you have many transformation rules, it might be beneficial to specify the transformations in a separate file and then include this file into 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", "modifyConfig".

When used with [import](#), "transform" could be specified within [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 "[path](#)" element. "transform" with "id" can be defined outside of any Ant target. "transform" can also contain any number of nested "transform" with "refid". DPBuddy will first execute actions referenced by nested transforms.

If "transform" does not contain any actions, DPBuddy is only going to perform Ant/Groovy/Gradle variable resolution for the variables defined in XML files. This does not apply to the "export" task, so that "transform" can be used for creating configuration templates containing variables.



Attributes Common to All Transform Actions

Attribute	Description	Required
xpath	XPath expression the action will be applied to. Not applicable to "dpInclude", "dpExclude" and "setText" actions.	Yes
matchRequired	Fail the build if xpath didn'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 to that, there is a special "dp" variable containing the following properties: "url", "username", "domain". These properties are populated with the connection information for the device and domain the task executing this transform action is being connected to. 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, sets the value of the text node of these elements.

If the action's XPath expression matched attributes, 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 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 "replaceWith" attributes.

The search is case-sensitive.

This action does not use XPath, 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, it supports [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 "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 namespace'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.

Under the covers, Ant delegates property resolution to property helper classes described in [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 the default DPBuddy's property provider.

DPBuddy's property provider relies on prefixes for defining 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 Ant.

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 is going to 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 "test.domain1." prefix and properties specific to the "test" environment using "test." prefix.

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

The prefix-based property management mechanism is completely generic and can be used for managing 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 Ant "property" task.

You can define "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.

"environment" Task

The environment prefix is specified using "environment" task. The prefix can contain one or multiple prefixes separated by ".". The trailing dot is not needed.

Once "environment" task is called, the DPBuddy's property provider will use the supplied set of prefixes for property resolution.

"environment" can be called multiple times within a build file. Each invocation will override the prefix defined earlier.

You can use "environment" tasks to copy configuration between domains or devices. This can be achieved by calling "environment" and providing the prefix for the source domain, running export, then executing "environment" with the prefix for the target domain and running "import".

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

Attributes

Attribute	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-	Yes



	<p>prefix and trying the resulting prefix and so on.</p> <p>An empty value (prefix="") will disable the DPBuddy's property resolution mechanism.</p>	
providerClass	<p>Property provider implementation. The class has to implement</p> <p>"org.apache.tools.ant.PropertyHelper.PropertyEvaluator" and "com.myarch.propertyselector.PropertyProvider" interfaces.</p> <p>Defaults to DPBuddy's property provider.</p>	No

Examples

This example shows how properties can be defined at both the domain and the device level. We use two-part prefix 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

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", "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 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 [global DPBuddy properties](#), every audit-related 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 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's 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 "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 " checkpoints:/dpbuddy-audit-log.xml"	No
dp.audit.domain dpAuditDomain	DataPower domain where to save the audit file (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 is going to 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 for Changing and Exporting Configuration

"import"/"buddyImport" Task

"import" is the main task for updating DataPower configuration. It imports previously exported configuration 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 for specifying Ant/Groovy/Gradle variables in any text node or an attribute of the deployment policy file. Any XML file used in transformation can also utilize references to Ant/Groovy/Gradle variables.

"import" supports transformation action 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

Attribute	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.	No



	Defaults to "false" (DataPower default).	
workDir	<p>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.</p> <p>Note that DPBuddy cleans this directory every time the import task is executed.</p> <p>If "transform" is not used or when the imported file is XML, "workDir" is ignored.</p> <p>Defaults to <code>{java.io.tmpdir}/dpbuddy/import</code>. "java.io.tmpdir" is a Java system property. It usually defaults to /tmp on Unix and \Users\<user name="">\AppData\Local\Temp on Windows.</user></p>	No
transformOnly	<p>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 not connect to DataPower.</p> <p>This is convenient for developing/troubleshooting "transform" actions.</p> <p>Defaults to "false".</p>	No

Examples

This is a simple example of the import task. See "samples/ant-tasks/import-export.xml" file located in your DPBuddy distribution or [online](#) for more examples.

```
<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 [Ant patternset](#). You have to define your include/exclude patterns in such a way so only XML files are selected by the patternset. DPBuddy will attempt to apply all transformation actions to all matched files.

Most of the 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 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 transformation actions documented in the section [Using DPBuddy to Transform XML Files](#).

"modifyConfig" Task

"modifyConfig" task updates DataPower configuration from the provided configuration file and/or the configuration information specified inline within "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 export facility or DPBuddy "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 the existing configuration object. Instead, it updates nested elements of the configuration. For example, for a load balancer group (LBG), "modifyConfig" will update only the members included into 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, "export" produces a non-compliant XML, which causes the validation to fail. Usually, removing "Memoization", "DebugMode" and "DebuggerType" elements is sufficient to make 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 "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.

Paid DPBuddy customers can contact DPBuddy support if they have questions or issues with defining their "modifyConfig" requests.

Attributes

Attribute	Description	Required
file	<p>Path to the configuration file. If the task contains "transform" element, the transformations will be applied to the content of the file.</p> <p>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".</p>	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 [Ant "fileset"](#) 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 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" Task

"delConfig" deletes DataPower objects. Similar to "export", it uses regexp patterns matching class names and object names to determine what to delete.

Note that "delConfig" does not automatically delete the dependent objects.

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

Attributes

Attribute	Description	Required
-----------	-------------	----------



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

Attribute	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 quiet="true".

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

See "samples/ant-tasks/del-config.xml" file located in your DPBuddy distribution or [online](#) for more examples.

"export" Task

"export" task exports configuration from a domain. The task supports all the functionality 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 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 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.

"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 "defaultTransform" attribute.

Transforming and formatting XML configuration files makes it easier to store configuration under version control and to track 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

Attribute	Description	Required
file	Name of the file to save the exported configuration to. The task will create directories containing the file if they don't exist. The file must have "zip" or "xml" or "xcfg" extension.	Yes
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 could also be used when you need to export a list of uniquely named objects without having to worry about what their classes are.	No



	Note: this attribute is deprecated. Use "exportObject" nested element instead.	
allFiles	<p>If set to "true", export all files from the local:/ filesystem.</p> <p>If not specified, only the referenced files are going to be exported.</p> <p>For export in XML format, this setting will be ignored unless "defaultTransform" is set to "false".</p> <p>Defaults to "false" (DataPower default).</p>	No
persisted	<p>If set to "true", export only the persisted configuration.</p> <p>Defaults to "false" (DataPower default).</p>	No
deploymentPolicyFile	<p>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.</p> <p>The deployment policy will be embedded with the export file.</p>	No
deploymentPolicyName	<p>Name of the deployment policy to include with the export/backup. The policy has to already be defined on the device.</p> <p>The deployment policy will be embedded with the export file.</p>	No
defaultTransform	<p>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.</p> <p>Defaults to "true".</p>	No



"exportObject" Nested Element

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

Attribute	Description	Required
class	Regular expression defining what classes (types) to be exported, 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 transformation actions documented in the section [Using DPBuddy to Transform XML Files](#). This element is optional. It can only be specified if you're exporting configuration in XML format.



"namePattern" Nested Element

"namePattern" element provides an alternative to specifying object name patterns using the "namePatterns" attribute. It is useful when a regular expression pattern contains 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.

Attribute	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").

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

See "samples/ant-tasks/import-export.xml" file located in your DPBuddy distribution or [online](#) for more examples.

"save" Task

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

Note that if 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

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

Tasks 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" Task

"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

Attribute	Description	Required
toDir	Specifies the target directory for uploaded files on the device. It could include a filesystem which is defined using 'filesystem:/' syntax. If not provided, the filesystem will default to 'local:/' You can override "toDir" at the fileset level.	No
cleanDirectories	If "true", delete target directories on the device before uploading files. Root directories (i.e., "local:/") will not be cleaned. Use 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
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



"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 "toDir" attribute that defines the root directory (and, optionally, the file system) on the device.

If "toDir" at the fileset and at the task level 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.

Attribute	Description	Required
toDir	Specifies the target directory for uploaded files on the device. It could include a filesystem which is defined using "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.

The 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 the 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/wsd1" directory tree on the device and upload the files. Since "cleanDirectories" is set to "true", it will delete "local:/apps/services" directory before uploading the files.

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

See "samples/ant-tasks/files.xml" file located in your DPBuddy distribution or [online](#) for more examples.

"delete" Task

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

Attribute	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, however, it is possible to delete everything under the root by using 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	No



	changes to the device. Defaults to "false".	
--	--	--

Examples

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

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

See "samples/ant-tasks/files.xml" file located in your DPBuddy distribution or [online](#) for more examples.

"mkdir" Task

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 [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

Attribute	Description	Required
dir	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

"dir" Nested Element

Attribute	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

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

"rmdir" Task

This task deletes directories on the device. This task is slightly faster than [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

Attribute	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 the build if the directory doesn't exist. Defaults to "true".	No

"dir" Nested Element

Attribute	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

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

"download" Task

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



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

Attributes

Attribute	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 but don't actually download anything. Defaults to "false".	No

Examples

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



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

See "samples/ant-tasks/files.xml" file located in your DPBuddy distribution or [online](#) for more examples.

"downloadFile" Task

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 what files to download. Use this task to download a single file and use "download" to download multiple files.

Attributes

Attribute	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 for Dealing with Configuration Checkpoints

"checkpoint" Task

"checkpoint" task creates a configuration checkpoint with the given name. It is equivalent to using "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 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

Attribute	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_HHmss". Defaults to "false".	No

Examples

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

"rollback" Task

This task 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

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

Examples

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

"delCheckpoint" Task

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

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

Attributes

Attribute	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

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

Tasks for Quiescence/Un-quiescence

Please refer to [DataPower InfoCenter](#) for the description of the quiescence process.

"quiesce" and "unquiesce" Tasks

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 was 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.

Attributes

Attribute	Description	Required
timeout	Time, in seconds, to wait for all objects to reach 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 quiesce/unquiesce.	

Examples

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

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

See "samples/ant-tasks/quiesce.xml" file located in your DPBuddy distribution or [online](#) for more examples.

"quiesceDomain" and "unquiesceDomain" Tasks

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 "domain" attribute or [dp.domain](#) property.

Examples

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

See "samples/ant-tasks/quiesce.xml" file located in your DPBuddy distribution or [online](#) for more examples.

Tasks for Checking DataPower Status

"status" Task

"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 certain characteristic of the device, such as memory, CPU utilization and others.

Complete list of status classes can be found under StatusEnum type in xml-mgmt.xsd.

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

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

Examples

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

"serviceStatus" Task

"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

Attribute	Description	Required
domainPatterns	Comma-delimited list of regular expression patterns defining what 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

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



"assertStatus" Task

"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 certain threshold.

The condition for the assertion is defined using 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 MemoryStatus variable some 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

Attribute	Description	Required
class	Status "class" as defined in xml-mgmt.xsd.	Yes
expression	<p>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.</p> <p>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 the task is being executed against.</p> <p>If the expression returns "false", the task will raise an exception.</p>	Yes



Examples

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

"assertFreeSpace" Task

"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

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

Examples

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

See "samples/ant-tasks/status.xml" file located in your DPBuddy distribution or [online](#) for more examples.

"assertState" Task

"assertState" checks operational state of DataPower objects determined by the nested "object" elements. The task raises 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 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 "down" state.

Attributes

Attribute	Description	Required
state	Operational state to check. Allowed values are "up" or "down". Error is raised if at least one object is not in this state.	No



	Defaults to "up".	
activeService	If set to "true", invoke "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 required
name	Regular expression defining object names to check.	

Examples

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

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

"assertActiveService" Task

"assertActiveService" checks if objects representing services are actually running and listening for requests. It is possible for a service, such as 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.

Attribute	Description	Required
class	Regular expression defining what classes (types) to	At least



	check.	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>
```

Tasks for Working with DataPower Logs

"tailLog" Task

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

You can also use the task to save the content 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.

"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 exception and fail the build. 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

Attribute	Description	Required
lines	Number of the most recent log entries to display. "-1" directs the task to print or save all available log entries. Defaults to 48 lines.	No
logTarget	Name of the log target defined on the device.	No



	Defaults to "default-log" (DataPower default).	
domainPatterns	<p>Comma-delimited list of regular expressions specifying domains to get logs from.</p> <p>Log entries from all matching domains are combined together and sorted by their timestamps.</p> <p>Use "domainPattern" nested elements if there is a need to use comma inside the regexp.</p> <p>Defaults to current domain. The current domain can be specified using "dp.domain" Ant property or the "domain" attribute of the task.</p>	No
format	Format string. See Log Entry Format section .	No
failOnError	<p>If set to "true", fail the build whenever an error-level log entry is encountered.</p> <p>Defaults to "false".</p>	No
failPatterns	List of comma-delimited regular expression patterns. "tailLog" will raise exception and fail the build 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	<p>If set to "true", query the device continuously every 3 seconds or according to the interval specified in "followInterval".</p> <p>New log entries (determined based on their timestamp) are appended to standard output.</p> <p>Defaults to "false".</p>	No
followInterval	<p>Interval in milliseconds used for continuously querying the device if "follow" is set to "true".</p> <p>This attribute is ignored if "follow" is "false".</p>	No



	Defaults to 3,000 milliseconds.	
logFile	Local file to save the DataPower log file to.	No
appendTimestamp	When logFile is set, append the timestamp to the local log file name. Defaults to "false".	No

Log Entry Format

"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}". 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.

Attribute	Description	Required
-----------	-------------	----------



class	<p>Regular expression defining classes of the log entries. Only the log entries with the matching classes will be printed.</p> <p>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):".</p> <p>If not specified, log entries will be printed regardless of classes.</p>	At least one is required
name	<p>Regular expression defining object names of log entries. Only the log entries with the matching object names will be printed.</p> <p>If not specified, log entries will be printed regardless of object names.</p>	

"domainPattern" Nested Element

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

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

Attribute	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 last 100 lines of the combined log.

```
<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:

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



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>
```

See "samples/ant-tasks/log.xml" file located in your DPBuddy distribution or [online](#) for more examples.

"log" Task

"log" creates a log entry in the DataPower log on the device. Logging a message in the device log could 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

Attribute	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 Administration/Miscellaneous/Configure Log Category in WebGUI. Defaults to "all".	No

Examples

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



"setLogLevel" Task

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 minimum event priority for "all" category.

Attributes

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

Examples

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

Tasks for Flushing DataPower Cache

DataPower caches stylesheets, XML documents and various other items. Oftentimes, there is 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" Tasks

These tasks flush XSLT and/or XML document caches:

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

Attributes

Attribute	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.

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

See "samples/ant-tasks/cache.xml" file located in your DPBuddy distribution or [online](#) for more examples.

"flushAAACache" Task

This task flushes AAA policy cache.

Attributes

Attribute	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" Task

This task allows for flushing the following DataPower caches: DNS, RBM, ND (neighbor discovery).

Attributes

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



"flushAllCache" Task

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 RBM and DNS cache types.

Backup Tasks

"backup" Task

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

The function provided by the this task is equivalent to that of "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 and keys. You need to 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 the "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 it is more flexible to use "export".

Attributes

Attribute	Description	Required
file	Name of the zip file to save the backups to. 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_HHmms".	No



	Defaults to "false".	
domainPatterns	Comma-delimited list of regular expression patterns defining what domains to back up. To backup all domains simply use ".*" Use "domainPattern" nested elements if there is a need to use comma inside the regexp. Defaults to current domain. The current domain is specified using "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 comma which is 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.

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

"secureBackup" Task

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 "toDir" attribute. Creating the backup files may take several minutes depending on the size of your filesystems.

Secure backups are always performed in the default domain.

Secure backup may take a few minutes to complete.

Note that secure backup files could 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 the downloading can begin.

Attributes

Attribute	Description	Required
cert	Name of the crypto certificate containing the public key used in encrypting the secure backup. 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.	Yes
toDir	Local directory where to save the backup files to. 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_HHmss".	No



	Defaults to "false".	
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":

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

Tasks 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" Task

This task restarts the domain.

Example

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

"resetDomain" Task

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

Example

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



Miscellaneous Tasks

"action" Task

This task allows for extending DPBuddy with various DataPower administration functions that are not implemented by other DPBuddy tasks.

A complete list of available actions can be found in xml-mgmt.xsd file, under the type "AnyActionElement". Documentation for actions can be found in the [DataPower command reference documentation](#) under "Global mode" and under "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, SOMA counterpart for ["save error-report" command](#) is "ErrorReport" action. This action can be executed as following:

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

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

Attributes

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

Examples

"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>
```

See "samples/ant-tasks/ misc.xml" file located in your DPBuddy distribution or [online](#) for more examples.

"somaRequest" Task

"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 "validate" attribute is set to "false".

Attributes

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

Examples

```
<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" Task

This task performs a synchronization of WSRR content with the WSRR server. See ["wsrr-synchronize" command documentation](#) for more details.

Attributes

Attribute	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" />
```