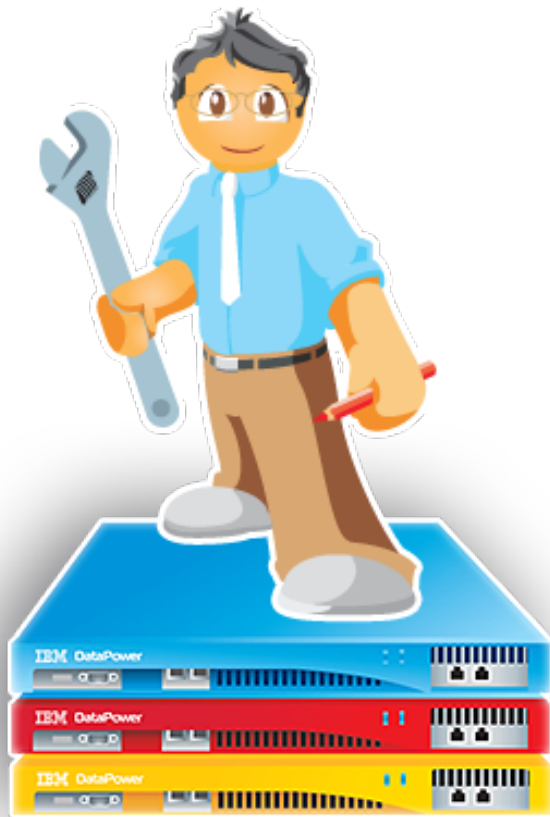




DataPower Buddy

User Guide

Version 2.3



Contents

Introduction.....	4
DPBuddy Support Services	5
What's New in This Release	6
Installation and DataPower Configuration.....	6
Adding DPBuddy Tasks to the Build	7
Upgrading from Older Versions of DPBuddy.....	7
Common Properties and Attributes	7
Connection Properties.....	7
Schema Validation Properties	8
Auto-Saving Property	9
Grouping Common Properties Using Environment Prefix.....	10
Ant Tasks for Updating Configuration	11
"import" Task.....	11
Attributes.....	11
Examples.....	12
"setConfig" Task	12
Attributes.....	12
"fileset" Nested Element	13
"configFile" Nested Element	13
Examples.....	13
"modifyConfig" Task.....	14
"save" Task	14
Examples.....	14
"checkpoint" Task.....	14
Attributes.....	14

Examples.....	15
Ant Tasks for Working with Files and Directories	15
"copy" Task.....	15
Attributes.....	15
"dpFileset" Nested Element	16
Examples.....	16
"downloadFile" Task.....	16
Attributes.....	16
Examples.....	17
"mkdir" Task	17
Attributes.....	17
Examples.....	17
"rmdir" Task.....	17
Attributes.....	17
Examples.....	18
Export and Backup Ant Tasks	18
"export" Task.....	18
Attributes.....	18
"exportObject" Nested Element.....	19
"namePattern" Nested Element.....	20
Examples.....	20
"backup" Task.....	20
Attributes.....	21
"domainPattern" Nested Element.....	22
Examples.....	22
Ant Tasks for Working with DataPower Logs	22
"tailLog" Task.....	22

Attributes.....	22
Log Entry Format	24
“where” Nested Element.....	24
“domainPattern” Nested Element	25
Examples.....	25
Miscellaneous Ant Tasks	26
"action" Task.....	26
Attributes.....	26
Examples.....	26
"fileRequest" Task	26
Attributes.....	27
Examples.....	27
"status" Task.....	27
Examples.....	27

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 the popular build tool, [Apache Ant](#).

Integration with Ant makes dpbuddy a perfect tool for automating release management and ongoing development of DataPower artifacts, such as XSLT, WSDL, schema and DataPower configuration files. Traditional administration tasks, such as backups, can also be easily automated using combination of Ant and dpbuddy.

You can download a detailed example of dpbuddy usage from <http://myarch.com/files/dpbuddy/example/build.xml>. Examples can also be found in the dpbuddy distribution available from myarch.com/dpbuddy.

Ant provides good support for configuration properties. For heavy-duty scripting, it integrates with many scripting languages. Specifically, for python integration, we can recommend our open-source "[PAnt](#)" framework. Ant is also capable of running tasks in parallel, which is useful

for managing multiple devices and domains. In short, we see Ant as a general-purpose framework ideally suited for automating DataPower administration tasks.

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 <http://www.redbooks.ibm.com/redpapers/pdfs/redp4446.pdf>. 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.

SOMA message format is governed by DataPower XML schema. The schema can be downloaded from your device. It's located under the "store" file system and it consists of three files: "xml-mgmt-ops.xsd", "xml-mgmt.xsd" and "xml-mgmt-base.xsd". The schema is useful when working with the dpbuddy tasks that utilize enumerations defined by the schema. Such tasks include "action" and "status". dpbuddy also uses the schema for validating SOMA requests on the client before sending them to the device.

If you run Ant in verbose mode ("ant -v"), you'll be able to see raw SOMA XML requests generated by dpbuddy and responses returned by DataPower.

We also recommend running Ant with "-e" option to suppress task name prefix in Ant logs. This makes it easier to read Ant output.

DPBuddy Support Services

dpbuddy is a free tool, but many organizations could benefit from our paid support services. Here are several reasons why you may want to pay for the support:

- There are many DataPower models and firmware versions. We'll make sure that the tool works without a hitch with the device and the firmware that you have.
- Quick turnaround for bug fixes. Bugs happen in any product. You don't want to be stranded during a critical deployment or a configuration change because of a bug.
- Consultation on the tool usage. We will quickly provide answers to all your questions.
- We will help you integrate dpbuddy with your existing build and deployment process and tools.
- You'll have a say in what goes into the next release of dpbuddy. Feature requests from our paid customers take top priority.
- Low annual fee.

If you're interested, please contact us at <http://myarch.com/contact-us>.

What's New in This Release

- Import task logs more detailed statistics including imported files, skipped files and imported objects.
- “exportObject” nested element of the “export” task now supports regular expression for both class and object names.
- “downloadFile” task.
- “mkdir” and “rmdir” tasks for creating and removing remote directories.
- “where” nested element of the “tailLog” task. This element enables filtering of DataPower log entries so only the ones related to specified classes or objects are printed.
- Support for “dryRun” attribute in the “import” task.
- Bug fixes.
- Updated documentation.

Installation and DataPower Configuration

dpbuddy requires Ant 1.7 or above and JDK 1.5 and above. Both IBM and Oracle JDKs are supported. dpbuddy is pure Java, so it runs on any OS where Java is available.

To install dpbuddy:

- Download and install Apache Ant as documented here:
<http://ant.apache.org/manual/install.html>.
- Download dpbuddy from myarch.com/dpbuddy, unzip/untar the content of the distribution in a directory.

On the DataPower side, 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.

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 "Common Properties and Attributes". In this case, you will need to add DataPower certificate to your JDK's keystore.

Adding DPBuddy Tasks to the Build

The easiest way to make Ant aware of the dpbuddy's tasks is to copy dpbuddy-2.3-single.jar to \$ANT_HOME/lib. Then you just need to add dpbuddy namespace declaration to the project element:

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

Alternatively, you can explicitly add "taskdef" to your build file and point it to the dpbuddy installation. Please see "example/build.xml" for more details.

You may also want to familiarize yourself with the [Ant documentation of the Antlib feature](#).

Upgrading from Older Versions of DPBuddy

If you're already using dpbuddy, follow the installation instruction and then update "taskdef" to point to the new version. If you want to use dpbuddy-2.3-single.jar, please make sure that you removed dpbuddy-2.2-single.jar from \$ANT_HOME/lib.

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 "fileRequest".

Common Properties and Attributes

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

For convenience every common attribute has 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.

Connection Properties

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.xmlmgm.url	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, it will default to 5550.	Yes
dp.username	username	Valid username with administrative privileges.	Yes
dp.password	password	Password.	Yes
dp.domain	domain	Target domain for dpbuddy tasks. Defaults to "default".	No
dp.trust.all.certs	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

dpbuddy can optionally validate all SOMA XML requests against the DataPower SOMA XML schema. Several dpbuddy tasks, such as "action", "request" and "setConfig", 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 produces a rather meaningless "internal error" message. 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 and displays error messages right away.

The following dpbuddy properties and attributes control schema validation:

Property name	Attribute name	Description	Required
dp.validate	validate	If set to "true", validate all XML SOMA requests. Defaults to "false".	No
dp.schema	schemaFile	Full path to xml-mgmt-ops.xsd. This is the main DataPower schema file. DataPower schema files can be downloaded from the "store" filesystem of the device. In addition to "xml-mgmt-ops.xsd" you will also need "xml-mgmt.xsd" and "xml-mgmt-base.xsd". dpbuddy comes bundled with the schema files for 4.0.2.1 firmware. If you set "dp.validate" to "true", dpbuddy will default to this schema. You must specify "dp.schema" if you have a different firmware version.	No

Example:

```
dp.validate=true
dp.schema=schema/xml-mgmt-ops.xsd
```

Auto-Saving Property

dpbuddy can automatically save configuration changes every time domain configuration is updated. This is done by running "save" SOMA command right after completion of "action", "setConfig", and "import" tasks. This command is equivalent to clicking "Save Config" from WebGUI.

Property name	Attribute name	Description	Required
dp.auto.save	autoSave	If set to true, automatically save domain configuration.	No

		Defaults to "false".	
--	--	----------------------	--

You must explicitly invoke "save" Task if "dp.auto.save" is not set or if it is set to "false".

Grouping Common Properties Using Environment Prefix

A typical organization uses multiple DataPower devices. Usually, there is at least one device used for development and testing and two devices used in production.

You can prefix your DataPower properties with a name of an environment or a device name. You can then refer to the device by its name in all dpbuddy tasks. This also allows for defining properties for all DataPower devices in a single file.

Example:

```
dev.dp.username=dpbuddy
dev.dp.password=buddy_123
dev.dp.domain=Test
dev.dp.xmlmgm.url=https://9.59.97.79:5050
```

The prefix has to end with ".".

To reference the device by its environment prefix, use the following property/attribute:

Property name	Attribute name	Description	Required
dp.env.prefix	envPrefix	Prefix used to define a group of DataPower properties. Do not use "." at the end of the prefix.	No

Example:

```
<property name="dp.env.prefix" value="dev" />
```

Ant Tasks for Updating Configuration

"import" Task

"import" task imports a previously exported configuration into a DataPower domain. It provides the functionality similar to "Import Configuration" WebGUI screen. The task utilizes "do-import" SOMA command.

"import" allows for specifying Ant variables in any text node or an attribute of the deployment policy file.

Attributes

Attribute	Description	Required
file	File to import. This could be a zip or an xml 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 has to already be defined on the device.	No
overwriteFiles	If set to true, overwrite existing files with the files included with the import. Defaults to "true" (DataPower default).	No
overwriteObjects	If set to true, overwrite existing objects with the objects defined as part of this import. Defaults to "true" (DataPower default).	No
rewriteLocalIp	If set to "true", the local address bindings of services contained in the import package will be rewritten to match the equivalent interfaces of the device. Defaults to "false" (DataPower default).	No
dryRun	If set to "true", run import in "dry run" mode. Defaults to "false" (DataPower default).	No

Examples

The following example imports the file defined by the "import.file" Ant property and applies "mypolicy.xml" to the configuration after the import. "mypolicy.xml" is automatically configured on the device before the import. If "mypolicy.xml" contains any Ant variables (in the form of "\${name}"), dpbuddy will resolve them before configuring the policy on the device. If the deployment policy defined by "mypolicy.xml" does not exist on the device, it will be created.

```
<dp:import file="${import.file}" overwriteFiles="true"
overwriteObjects="true" deploymentPolicyFile="mypolicy.xml" />
```

"setConfig" Task

"setConfig" task updates DataPower configuration from the provided configuration file. The configuration file contains DataPower object definitions in the format dictated by the DataPower schema (xml-mgmt.xsd, "AnyConfigElement" type).

You can use DataPower export facility or "export" task to produce an initial version of the configuration file.

"setConfig" creates a new object if the object specified in the configuration file does not exist.

"setConfig" is similar to "import", but it has these important differences:

- "setConfig" does not support deployment policies; instead you can use [xpath-based expressions](#) to override values in the configuration file.
- Configuration files can contain Ant variables in any text node or in any attribute.
- Configuration files can be validated locally against DataPower schema (provided "dp.validate" property is set to "true"). Content of the import file can't be validated locally.
- Multiple configuration files can be provided. "setConfig" will merge them into one. "setConfig" supports nested filesets and "configFile" elements to specify multiple configuration files.

Please note that the current DataPower schema is incomplete. If you try to use the file that was created by "export" without any changes, validation will fail. To make it pass, you need to comment out the following elements in the configuration file: Memoization, DebugMode, DebuggerType.

Because of this issue, "setConfig" and "modifyConfig" will try to validate by default (you can override it 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" back.

Attributes

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

file	Path to the configuration file. Use this attribute if you have a single configuration file and if you don't need to use xpath overrides. Otherwise use "configFile" or "fileset".	No
------	---	----

"fileset" Nested Element

You can use [Ant "fileset"](#) to specify multiple local config files that will be used to update DataPower configuration. dpbuddy will combine all matching files into a single request to the device. If schema validation is enabled, dpbuddy will validate the resulting file before sending the request to the device.

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

"configFile" Nested Element

"configFile" supports overriding values in the configuration file. This is done via nested "override" elements. "override" supports "xpath" and "value" attributes. For each element matching the xpath expression, "setConfig" sets the element's text to the provided value. If the element does not have a child text node, it will be added. If the xpath does not match any elements in the configuration file, the exception will be raised.

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

"configFile" supports the following attributes:

Attribute	Description	Required
file	Path to the configuration file.	Yes

The nested element "override" supports the following attributes:

Attribute	Description	Required
xpath	xpath expression. The expression must match at least one element in the configuration file.	Yes
value	Value to set matched elements to. The value is set by modifying or adding the child text node of each element.	Yes

Examples

The following example illustrates the use of xpath overrides:

```
<dp:setConfig>
  <configFile file="dpconfigs/config-wsproxy.xml">
    <override xpath="//RemoteEndpointPort" value="{endpoint.port}"/>
  </configFile>
</dp:setConfig>
```

"modifyConfig" Task

"modifyConfig" is very similar to "setConfig" task and it supports all the same attributes and nested elements. Unlike "setConfig", "modifyConfig" does not create new objects if they don't exist.

"modifyConfig" utilizes "modify-config" SOMA directive. Note that the schema for elements allowed in "modify-config" is different from "set-config". Elements definitions can be found under "AnyModifyConfig" type in xml-mgmt.xsd.

Please see "setConfig" Task section for information on the task's attributes and elements.

"save" Task

"save" task saves the domain configuration by utilizing SaveConfig SOMA action. 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" or "setConfig"). In this case, you don't need to explicitly call "save".

Examples

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

"checkpoint" Task

"checkpoint" task creates a configuration checkpoint with the given name. It is equivalent to using "Administraion/Configuration/Configuration Checkpoint" screen of WebGUI. The task executes "SaveCheckpoint" SOMA action. 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.

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	No

	format "yyyyMMdd_HHmss". Defaults to "false".	
--	--	--

Examples

```
<dp:checkpoint name="chk1" />
```

Ant 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. It utilizes "set-files" SOMA command.

"copy" also creates the directory structure containing the files similarly 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
cleanDirectories	If "true", delete target directories on the device before uploading files. Defaults to "false".	No
flatten	If "true", do not create any subdirectories; create only root directories on the device. Defaults to "false".	No

"dpFileset" Nested Element

"dpFileset" specifies files on the local file system that will be uploaded to the device. "dpFileset" supports all attributes and nested elements of the regular Ant "[fileset](#)". Its only unique attribute is "prefix". "prefix" defines the root directory (and, optionally, the file system) on the device.

If "prefix" is omitted, dpbuddy will use the "local://" filesystem and the base directory of the fileset (the directory specified by the "dir" attribute) as the root on the device.

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

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

Attribute	Description	Required
prefix	Specifies root directory for uploaded files on a device. It could include a filesystem which is defined using 'filesystem://' syntax. If not provided, the filesystem will default to 'local://'	No

Examples

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

The following command will create "local://apps/services/person/wsd" 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">
  <dpFileset prefix="/apps/services"
            dir="services" includes="**/*.wsdl **/*.xsd"/>
</dp:copy>
```

"downloadFile" Task

This task downloads a file from the device into the specified location on the local file system.

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" />
```

"mkdir" Task

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

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

Examples

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

"rmdir" Task

This task deletes a directory on 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
failOnError	Fail the build if the directory doesn't exist Defaults to "true".	No

Examples

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

Export and Backup Ant Tasks

"export" Task

"export" task exports configuration from a domain. It utilizes "do-export" SOMA command, which, in turn, supports all the features of WebGUI "Export configuration" screen.

Both XML and ZIP file formats are supported. The format is determined automatically based on the provided file name.

Note that if you choose to provide a deployment policy as part of the export task, DataPower will add it to the resulting export configuration file ("export.xml"). This will preclude you from using Ant variables in your deployment policy when you run "import".

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" 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. Note: this attribute is deprecated. Use "exportObject" nested element instead.	No
allFiles	If set to true, export all local files.	No

	Defaults to "false" (DataPower default).	
persisted	If set to true, export only the persisted configuration. Defaults to "false" (DataPower default).	No
deploymentPolicyFile	Path to the deployment policy file on the local 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. The deployment policy will be embedded with the export file.	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, but you need to specify at least one. Both attributes support regular expressions. This gives you a lot of flexibility; you can define regexps matching classes or objects or both.

"exportObject" support other attributes, such as "includeDebug". Values of these attributes will be applied to all matched objects.

The easiest way to find out class names is to export the desired object manually from WebGUI and then look at the element name of the object in export.xml. The element name is the class name.

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

Attribute	Description	Required
class	Regular expression defining what classes (types) to be exported, e.g., "WSGateway". You can find a complete list of all classes in "xml-mgmt.xsd".	At least one is required

name	Regular expression defining objects to be exported. Only the objects whose name matches the regexp will be exported.	
refObjects	If set to true, include all object references/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

"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"). All matching objects will be exported with the debug information because of "includeDebug=true".

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

"backup" Task

"backup" executes "do-backup" SOMA command to backup an entire domain or multiple domains. It is also possible to backup all domains.

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

Attributes

Attribute	Description	Required
file	<p>Name of the zip file to save the backups to. The resulting zip file will contains individual zip files for each domain.</p> <p>The task will create directories to save the file to if they don't exist.</p>	Yes
appendTimestamp	<p>If true, automatically append timestamp to the file name. The timestamp has the format "yyyyMMdd_HHmms".</p> <p>Defaults to "false".</p>	No
domainPatterns	<p>Comma-delimited list of regular expression patterns defining what domains to back up.</p> <p>To backup all domains simply use ".*"</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 is specified using "dp.comain" property or "domain" attribute of the task.</p>	No
persisted	<p>If set to true, backup only the persisted domain configuration.</p> <p>Defaults to "false".</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</p>	No

	defined on the device.	
--	------------------------	--

"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" />
```

Ant 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.

"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 a regular 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.

The task utilizes "get-log" SOMA command.

Attributes

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

lines	Number of the most recent log entries to display. Defaults to 48 lines.	No
logTarget	Name of the log target defined on the device. Defaults to "default-log" (DataPower default).	No
domainPatterns	Comma-delimited list of regular expressions specifying domains to get logs from. Log entries from all matching domains are combined together and sorted by their timestamps. Use "domainPattern" nested elements if there is a need to use comma inside the regexp. Defaults to current domain. The current domain can be specified using "dp.domain" Ant property or the "domain" attribute of the task.	No
format	Format string. See "Log Entry Format" section.	No
failOnError	If set to "true", fail the build whenever an error-level log entry is encountered. Defaults to "false".	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	If set to "true", query the device continuously every 3 seconds or according to the interval specified in "followInterval". New log entries (determined based on their timestamp) are appended to standard output.	No

	Defaults to "false".	
followInterval	Interval in milliseconds used for continuously querying the device if "follow" is set to "true". This attribute is ignored if "follow" is "false". Defaults to 3000 ms.	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.

You can specify multiple “where” elements within the same task.

Attribute	Description	Required
class	Regular expression defining classes of the log entries. Only the log entries with the matching classes will be printed. To find out class names, navigate to the object you’d like to print log entries for in WebGUI and click on “View log”. A log message usually starts with the	At least one is required

	<p>prefix in the format <class><object>, e.g., "wsgw (testServiceProxy):".</p> <p>If not specified, log entries will be printed regardless of classes.</p>	
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>
```

Miscellaneous Ant Tasks

"action" Task

"action" task executes an arbitrary SOMA action. Actions allow for performing many DataPower administrative tasks, such as flushing caches and resetting domains and passwords. Full list of SOMA actions can be found in xml-mgmt.xsd file, under the complex type "AnyActionElement".

For those actions that don't have child elements, you can simply specify the action's name using the "name" attribute of the task. Examples of such actions include "ResetThisDomain" and "FlushDNSCache".

For all other 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

This is the action that does not have child elements:

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

This is the action with child elements:

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

"fileRequest" Task

"fileRequest" 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 an attribute of the XML file.

Attributes

Attribute	Description	Required
file	SOMA request file.	Yes

Examples

```
<dp:fileRequest 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>
```

"status" Task

"status" 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" />
```