Release Builder

User Guide

Author: Richard Harrison

1 Introduction

1.1 Background

This tool originated out of the need to automate the manual process of creating archives that contained all of the required files to deploy a release to a server but to be able to use gitversion to get the release number. Previously a collection of various scripts where used (bat, awk, sh).

To meet these requirements the ReleaseBuilder was developed which takes an XML file as input and takes care of the required steps, including building / publishing using the command line tools for C#

The ReleaseBuilder supports inclusion of other XML files.

2 XML Configuration

2.1 Location and Naming

By default **ReleaseConfig.xml** in the current directory will be used. This is the usual method.

Using the **–config Filename.xml** it is possible to select any XML file.

The **Root** is defined as being where the config Filename.xml is located unless the (r)oot option is specified on the command line. The Root is where the files (to be included) will be located from.

Files and folders will be located by looking in the folders as follows:

- current folder attribute (if applicable)
- The Root of the release; i.e. where the ReleaseConfig.xml is located or the path specified on the command line
- The current directory where ReleaseBuilder is run

2.2 Variable substitutions

Most of the paths and files can contain variables that will be expanded.

2.2.1 Environment variables

Environment variables are identified by an attribute that starts with a $. The entire attribute will be looked up in the environment and replaced, even if empty.

2.2.2 Build variables

Any attribute can contain any number of the following surrounded by ~, e.g. ~SEMVER~

Name	Example Value
AssemblySemFileVer	1.1.30.0
AssemblySemVer	1.1.30.0
BranchName	feature/develop-r
CommitDate	2024-10-28
CommitsSinceVersionSourcePadded	0
FullBuildMetaData	Branch.feature/develop-r.Sha.56b9179406e564ba274deb9e86648dd4e3e13f46
FullSemVer	1.1.30
InformationalVersion	1.1.30
IntSemVer	10130
LegacySemVer	1.1.30
LegacySemVerPadded	1.1.30
MajorMinorPatch	1.1.30
NuGetVersion	1.1.30
NuGetVersionV2	1.1.30
TARGETPATH	Target output path
PUBLISHROOT	d:\dev\project\aspnet-core; either the folder the release builder was started from or the value specified on the command line using –root
SemVer	1.1.30
Sha	56b9179406e564ba274deb9e86648dd4e3e13f46
TYPE	test
VERSION	1.1.30

2.3 Supported transformations

It is possible to use the transform (as an attribute) on the following verbs to perform transforms

- Build verb, copy action, transform – will transform output filename
- Build verb, copy action, transform-contents – will run the transform on the contents; e.g. replace one string with another.

Available transformations:

set	Set the node or content element to be a variable, e.g. set,$SEMVER
getversion	Gets a version from the argument
replace	String1, string2 – replaces a string with another.
regex-replace	2 arguments, first is the regex to replace and the second the regex to replace with e.g. `<transform-content transform='regex-replace,Version\=.*? Publisher,Version="~AssemblySemFileVer~" Publisher'/>`
when	args, string1, condition, string2; condition can be “eq” “=” “==” or “ne” “!=” “<>”

2.3.1 GetVersion

GetVersion will extract the numerical version (##.##.##) from the parameter which would be either a variable, environment variable,text or a combination of all three.

2.3.2 Replace

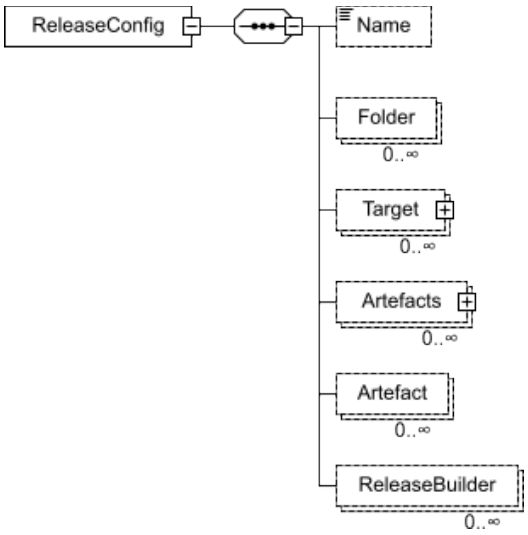
Replace occurrence of one string with another.

3.1 Overview

3.2 ReleaseConfig Tag

The file contains one ReleaseConfig element with a number of children as follows:

Tag	Use	Description
Name	Required	
Folder	Optional	Sets variable based on folder matching on disk. Useful when part of the build process will place files in a location which includes a version number
Target	Required	One or more to define the possible targets Attributes name — Name of the target; e.g. test, live, dev path — Where to put the output files archive-version — Transform to apply to get the version, typically used like this archive-version="GetVersion,~APPPATH~" with the Folder tag setting the APPPATH variable
Artefact	0 or more	Specifies a file or folder to include in the built zip. Deprecated, use Artefacts instead Attributes folder — Adds the contents of a folder to the zip directory — synonym for folder file — adds a file to the zip. Use skip-directories-front to remove a number directories from the zip file, used to create a zip file that has the files/folders without any parent folders
Artefacts	0 or more	Can be conditional, specifies a folder and which files or folders to include into the zip. Can also perform build steps.
ReleaseBuilder	Optional	Continue with another release builder XML files folder — Folder to use process — Whether or not to process the build to create a zip file. Defaults to false file — File to use



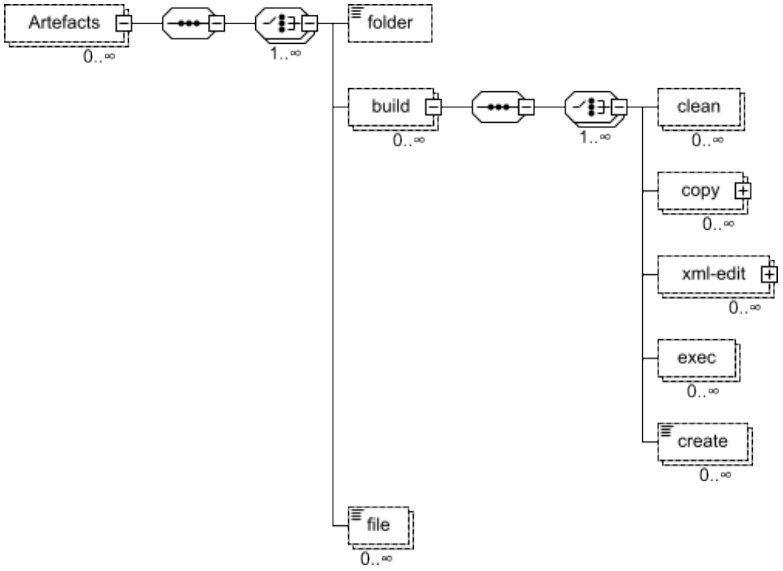
3.3 Artefacts Tag

This tag is the preferred way to specify the contents of the built archive, however it is also possible to use the build tag to perform a number of steps that makes a release that doesn't require any zip file to be created, for example when building an Android APK

In the simplest form this tag can specify which files or folders to include into the zip, however more usually the build step is used to exec processes to create the build, and the clean and copy tags can be used to build a folder structure for inclusion into the zip, i.e. to selectively copy files possibly built using exec or configuration files that are already present. Any number of exec operations can be present.

All build operations will be completed before any files are added to the zip.

3.3.1 Overview of Artefacts tag



4 Reference

4.1 Name Tag

This is required and is the target name that will be used for the output name

4.2 Folder Tag

This tag is used to set a variable based on a matching folder path. The folder path to match can contain a regex and if it does the version tag specifies which matching folder (on disk) will be used to set the variable.

The search will be from the Root.

name attribute

Specifies the variable name to set.

name-version attribute

Sets the specified variable to a version number extract from the selected name. This requires more than 1 dots in the name string.

path attribute

Specifies the path to locate. This can contain regexes. If multiple files match the version attribute defines which one to use.

version attribute

Defaults to “latest” so that the directory with the newest date will be used.

- latest – most recent
- oldest – oldest date
- name – first name when ordered alphabetically
- last-name – last name when ordered alphabetically

4.3 Target Tag

Usually occurs multiple times and defines the target for use with the -t command line parameter.

4.3.1 name attribute

This defines the target name, e.g. test, live, demo

4.3.2 path attribute

This defines where the built artefact is placed. It will be named using the name attribute and the version.

4.3.3 archive-version attribute

By default the version used will be from gitversion; however it is possible that a different version may be required; for example from the path name of a folder. The text will be transformed

4.4 Target.Set Tag

The Set tag when embedded in a <Target> Tag will set a variable to a defined value;

e.g.

```
<Target name="live" path="$SYNC_MYAPP_LIVE" archive-version="GetVersion,~APPPATH~">
    <Set name="InstallURL" value="https://MyApp.example.com/"/>
</Target>
```

4.5 Artefacts Tag

This defines a set of artefacts that are required.

An optional <build> tag can define a set of actions to create the archive. Usually this will copy files into a temporary directory and perform actions.

The <file> and <folder> tags define the files to put in the resulting archive.

Normally the first action in the build section would be to clean the current contents – which will remove all files but no folders.

4.5.1 active attribute

Usually of the format “when,~VARIABLE~,eq,Test”

supported comparisons are eq,=,== or ne,!=,<>

4.5.2 folder attribute

Defines the location where the processing will be performed. This folder will be included in the search path for files and folders before the other places.

4.5.3 build Tag

Used to copy files into a folder, usually because a subset is required or to add specific configuration files for the target type.

Each sub tag will be executed in the order that it appears in the config file. Usually the clean tag would be first however it is possible that in a complex build with mulitple steps the clean command might be used more than once.

4.5.3.1 folder attribute

If specified the folder attribute will be added to the search path in addition to the Artefacts path. It is therefore best to use relative notation.

4.5.3.2 clean Tag

Delete a folder contents including sub folders. Required attribute “folder” must specify the root of the path to be cleaned.

e.g. <clean folder="files/"/>

4.5.3.2.1 folder attribute

Folder to copy clean. (required)

4.5.3.2.2 match attribute

Pattern match; defaults to “*.*”

4.5.3.2.3 include-folders attribute

When true will remove both files and folders.

4.5.3.3 copy Tag

The copy tag can appear multiple times inside the <build> tag and is used to copy a selection of files usually into folder that will be used for the final archive

4.5.3.3.1 From

Folder to copy from. (required)

4.5.3.3.2 to

Destination Folder (required)

4.5.3.3.3 name

New name – only useful when match will identify a single file (optional)

4.5.3.3.4 match

Filename matching string; e.g. *.dll or a single file name

4.5.3.3.5 recursive

Copies files recursively; creating directories as needed

4.5.3.3.6 transform

Transform output filename; works with both files and folders

4.5.3.3.7 transform-contents

Run the specified transform on the file contents during copy. Only text files are supported. Files are copied with File.Copy – if this attribute is specified the resulting file will be opened, transformed and set.

4.5.3.4 create Tag

Required attribute file=”” to specify the output file.

The inner text of the tag will be the contents of the file created, with variables replaced as normal.

4.5.3.5 exec

Starts a process

4.5.3.5.1 app

The app to execute; usually a file.exe but could be a batch file or other extensions. This will be located in the tools path and environment path. If it cannot be found then an error will be given

4.5.3.5.2 args

Command line arguments; will be expanded.

4.5.3.5.3 Folder

The folder where the executable is to be launched.

4.5.3.5.4 required-exit-codes

Comma separated list of exit codes that indicate success, defaults to 0. If not present the exit code will not be checked.

4.5.3.6 modifyTag

This will modify the contents of a file by applying the transformation child nodes of tag **transform-content**

4.5.3.6.1 file attribute

Specifies the file to modify

4.5.3.7 release-builder

Loads a new release builder and prepares all – but does not by default create an archive. To create an archive the process="true" is required.

4.5.3.7.1 folder

The folder where the new build will be run

4.5.3.7.2 file

File to load; default ReleaseConfig.xml

4.5.3.7.3 nobuild

Defaults to parent nobuild; whether or not to run the build steps

4.5.3.7.4 process

Boolean attribute that defaults to false that will create the archive.

4.5.3.8 xml-edit

Permits edit of an XML file using Xpath elements

4.5.3.8.1 file attribute

Specifies the file to edit. Must exist and be well formed XML

4.5.3.8.2 omit-declaration

Set to true to not emit the standard XML declaration as the first line.

4.5.3.8.3 node

Specifies the Xpath of node(s) to edit via the path attribute and the action to perform. The action can be any valid transformation.

e.g. to change the version node for an ABP project.

```
<xml-edit file="common.props" omit-declaration="true">
    <node path="//PropertyGroup/Version" action="set,~SemVer~"/>
</xml-edit>
```

4.5.4 file Tag

Tag contents specifies a file or match to include in archive

The optional folder tag specifies where to look for the file

4.5.4.1.1 newname

New name

4.5.4.1.2 folder

Specifies folder where copied file(s) are to be placed.

4.5.5 folder Tag

Folder to include in the output; all sub folders will be included

4.6 Artefact tag

As an alternative, or in addition to the <Artefacts> number of <Artefact> tags can specify files or folders to add to the archive.

Normally multiple Artefact tags are used to define what is included in the output

4.6.1 directory Attribute

specifies a folder and subfolders to include. This can either identify a specific folder or a match expression that will be used to match files in the root path.

4.6.2 folder Attribute

Synonym for directory

4.6.3 file Attribute

Attribute specifies a single file or a set of files to match.

4.6.4 Root attribute

The root defines where an attribute is located on disk. This will be used when locating files within a folder/directory. Individual files specified via the file attribute do not use this tag.

4.6.5 ReleaseBuilder

Loads a new release builder and prepares all – but does not by default create an archive. To create an archive the process="true" is required.

4.6.5.1 folder

The folder where the new build will be run

4.6.5.2 file

File to load; default ReleaseConfig.xml

4.6.5.3 process

Boolean attribute that defaults to false that will create the archive.

4.6.5.4 nobuild

Defaults to parent nobuild; whether or not to run the build steps

5 Command Line Usage

5.1 Switches

Each command line switch has a short form (-) and a long form (--).

Usually the target type (-t, --target) is specified

Full switch	short	Type	Description	Default	Example
config	c	File	Config file to use	ReleaseConfig.xml	--config ReleaseConf
nobuild	n	Switch	Do not perform the <build> steps in the config. This option is usually only used rarely to re-release an existing build and must be used with care		
root	r	Directory	Folder to use	Current Directory	
target	t	String	Target type	live	
toolsdir	p	Directory	path to search for tools (used in exec tags). Can occur more than once to specify multiple directories		
verbose	v	Switch	Verbose output. Can be specified twice to have more output		

6 Examples

6.1 Angular

This will run node to execute ng to build using the config that was previously copied into the source tree. The result will be the contents of the **dist** folder

```
<?xml version="1.0" encoding="utf-8"?>
<ReleaseConfig p1:noNamespaceSchemaLocation="ReleaseConfig.xsd" xmlns:p1="http://www.w3.org/2001/XMLSchema-
instance">
    <Name>MyApp-WebUI</Name>
    <Target name="test" path="$SYNC_MYAPP_TEST"/>
    <Target name="live" path="$SYNC_MYAPP_LIVE"/>
    <Target name="demo" path="$SYNC_MYAPP_DEMO"/>

    <Artefacts>
        <build>
            <create file="src\app\appVersionDetails.ts">
                export const appVersionDetails =
                ~GITVERSION.JSON~
                ;
            </create>
            <copy from="env-~TYPE~" match="*.*" to="src\environments"/>

            <exec app="node.exe" args="node_modules\@angular\cli\bin\ng build --env=prod"
folder="~PUBLISHROOT~" log-stdout="true"/>
        </build>
        <folder>dist</folder>
    </Artefacts>
</ReleaseConfig>
```

6.2 UWP App

Prior to running this the store packages must be created and this uses the Folder version to locate the built version based on the latest (date) folder

The version number of the archive will use the version number from the folder. To be consistent this should match the gitversion.

```
<?xml version="1.0" encoding="utf-8"?>
<ReleaseConfig p1:noNamespaceSchemaLocation="/xml-schemas/ReleaseConfig.xsd"
xmlns:p1="http://www.w3.org/2001/XMLSchema-instance">
    <Name>MyAppApp</Name>
    <Folder name="APPPATH" name-version="APPPATH-VERSION" path="MYAPP.UWP_*" version="latest" />
    <Target name="test" path="$SYNC_MYAPP_TEST" archive-version="GetVersion,~APPPATH~">
        <Set name="InstallURL" value="https://MyApp.beta5.lab.brightservecloud.com/"/>
    </Target>
    <Target name="live" path="$SYNC_MYAPP_LIVE" archive-version="GetVersion,~APPPATH~">
        <Set name="InstallURL" value="https://MyApp.example.com/"/>
    </Target>
    <Target name="demo" path="$SYNC_MYAPP_DEMO" archive-version="GetVersion,~APPPATH~">
        <Set name="InstallURL" value="https://demo.example.com/"/>
    </Target>

    <Artefacts folder="release-~TYPE~">
        <build>
            <clean folder="files/"/>
            <copy from="MYAPP.UWP.appinstaller" to="files">
                <transform-content transform='replace,Version="~APPPATH-VERSION~",Version="~TargetVersion~"'/>
                <transform-content transform='replace,https://MyApp.example.com/MyApp/app/,~InstallURL~MyApp/app/'/>
            </copy>
            <copy from="index.html" to="files">
                <transform-content transform='replace,Version="~APPPATH-VERSION~",Version="~TargetVersion~"'/>
                <transform-content transform='replace,https://MyApp.example.com/MyApp/app/,~InstallURL~MyApp/app/'/>
            </copy>
            <copy from="~APPPATH~"  to="files" match="*.appxbundle" />
            <copy from="~APPPATH~/Dependencies/x64/"  to="files/Dependencies/x64/" />
        </build>
        <file skip-directories-front="2" folder="files">*.*</file>
        <file skip-directories-front="2" folder="files/Dependencies/x64/">*.*</file>
    </Artefacts>
</ReleaseConfig>
```

6.3 Squirrel App

A number of steps is required as the files have to be copied, signed, built and signed again prior to the archive being created

```
<?xml version="1.0" encoding="utf-8"?>
<ReleaseConfig p1:noNamespaceSchemaLocation="ReleaseConfig.xsd" xmlns:p1="http://www.w3.org/2001/XMLSchema-instance">
    <Name>MyAppUpdater</Name>
    <Target name="test" path="$SYNC_MYAPP_TEST"/>
    <Target name="live" path="$SYNC_MYAPP_LIVE"/>
    <Target name="demo" path="$SYNC_MYAPP_DEMO"/>

    <Artefacts folder="release-~TYPE~">
        <build>
            <clean folder="files/"/>
            <copy from="bin/Release/" match="*.*" to="files"/>
            <exec app="signtool.exe" args="sign /sha1 9925D37880B5CED926E108911ABA5F6274A7D0FC files\
MyAppUpdater.exe"/>
            <exec app="nuget.exe" args=" pack MyAppUpdater.nuspec -properties version=~VERSION~"/>
            <exec app="Squirrel.com" args='-n "/a /f ..\..\certificates\MYAPP-spc.pfx /p bb /fd sha256 /tr
http://timestamp.digicert.com /td sha256" --no-msi --releasify MyAppUpdater.~VERSION~.nupkg -g ..\splashscreen.gif'/>
            <copy from="Releases\setup.exe" to="Releases" name="Setup.~VERSION~.exe"/>
            <copy from="Releases\RELEASES" to="Releases" name="RELEASES~VERSION~"/>
        </build>
        <file folder="Releases">MyAppUpdater-~VERSION~-delta.nupkg</file>
        <file folder="Releases">MyAppUpdater-~VERSION~-full.nupkg</file>
        <file folder="Releases">SetupUpdate.exe</file>
        <file folder="Releases">RELEASES</file>
    </Artefacts>

</ReleaseConfig>
```

6.4 WEB API

This copies in the required files from the publish web foder and then copies the config files for the target type.

```
<?xml version="1.0" encoding="utf-8"?>
<ReleaseConfig p1:noNamespaceSchemaLocation="ReleaseConfig.xsd" xmlns:p1="http://www.w3.org/2001/XMLSchema-instance">
    <Name>MYAPP.WEBAPI</Name>
    <Target name="test" path="$SYNC_MYAPP_TEST"/>
    <Target name="live" path="$SYNC_MYAPP_LIVE"/>
    <Target name="demo" path="$SYNC_MYAPP_DEMO"/>

    <Artefacts folder="release-web-~TYPE~">
        <build>
            <clean folder="files/"/>
            <copy from="publish-web/" match="*.dll*" to="files"/>
            <copy from="publish-web/" match="*.exe" to="files"/>
            <copy from="publish-web/" match="*.xml" to="files"/>
            <copy from="publish-web/" match="*.config" to="files"/>
            <copy from="publish-web/" match="appsettings.json" to="files"/>
            <copy from="MYAPP.API.exe.config" to="files"/>
            <copy from="appsettings.json" to="files"/>
            <copy from="web.~TYPE~.config" to="files" name="web.config"/>
        </build>
        <file folder="files">*.*</file>
    </Artefacts>
</ReleaseConfig>
```

6.5 Top Level Build all

To build a number of XML files contained in subfolders; just include them.

e.g. for a project with angular and aspnet-core with a db migration tool the following is used

```
<?xml version="1.0" encoding="utf-8"?>
<ReleaseConfig p1:noNamespaceSchemaLocation="file:///c:/xml-schemas/ReleaseConfig.xsd"
xmlns:p1="http://www.w3.org/2001/XMLSchema-instance">
    <Name>Project</Name>
    <Target name="test" path="$SYNC_SERVET_TEST"/>

    <ReleaseBuilder folder="angular" process="true" />
    <ReleaseBuilder folder="aspnet-core"  process="true" />
    <ReleaseBuilder folder="aspnet-core" file="ReleaseConfigDbMigrator.xml"  process="true" />
</ReleaseConfig>
```