CA Release Automation

Token Support

Best Practices Guide

Author : Walter Guerrero

Version: 1.4

Filename: CA-Release-Automation-Token-Support-Best-Practices-GuideV1.4

Date: 3/18/2015

Table of Contents

Overview

Starting with release 5.5.0 of Release Automation, the concept of “tokens” has been introduced. Tokens provide the user with the ability of providing different values for different token values within configuration files based on the environment that is being used.

Tokens work very closely with artifact definitions. Tokens can also be used for replacement of objects that are part of a file.

Tokens provide you with a lot of flexibility when it comes to replacing file contents that have been downloaded by Release Automation to a given server, this is very important when you are dealing with different environments and tokens reduce the number of errors at deployment time.

Setting up a Token

The token administration is accessed via the “Administration→Token Management” option in the ROC. Let us go over how to setup tokens and available options, there are two main areas that you will have to concentrate prior to the creation of a token.

Token Masks

Masks are used to enforce validation on token values. The pre-defined masks that have been created deal with enforcing validation of IP and email addresses, for custom mask you will defined the validation rule.

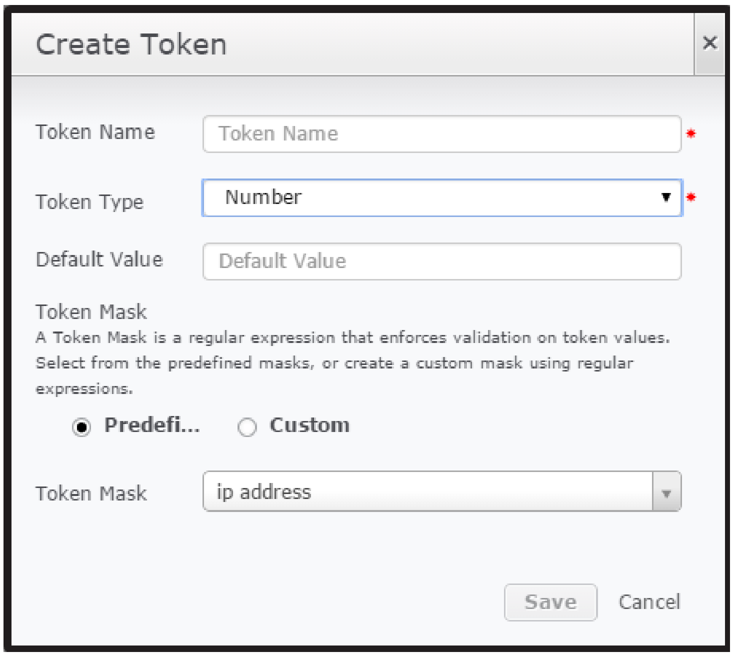
The pre-defined masks apply to String and Number token types.

- Predefined
  - IP address
  - email
- Custom
  - The value that you provide for the token must match the mask

Token Type

There are four available token types that you can use. For the String and Number types, you can use pre-defined masks, whereas for the Boolean type, you can only have “True or False”; for a password type, you get to enter the password, which will be masked.

- Boolean
- Number
- String
- Password



Token Name: Token Name *

Token Type: Number *

Default Value: Default Value

Token Mask

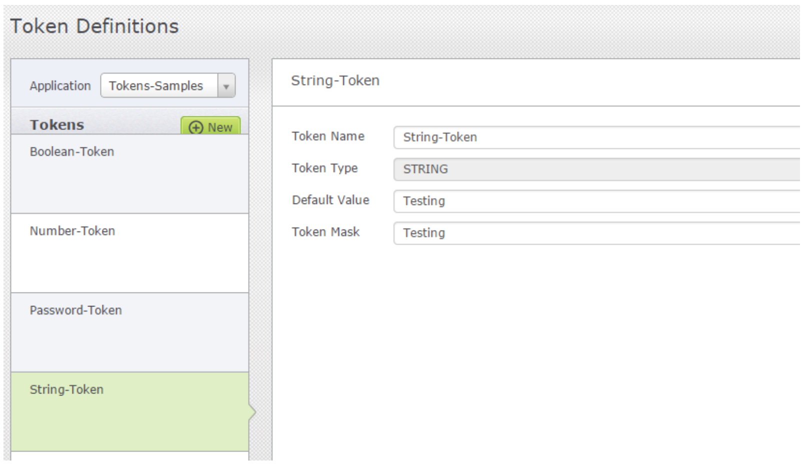
A Token Mask is a regular expression that enforces validation on token values. Select from the predefined masks, or create a custom mask using regular expressions.

[✓ Predefi...] Custom

Token Mask: ip address

Figure 1: Typical token creation

Once the tokens that you will be using have been defined as shown below, we are going to have to create

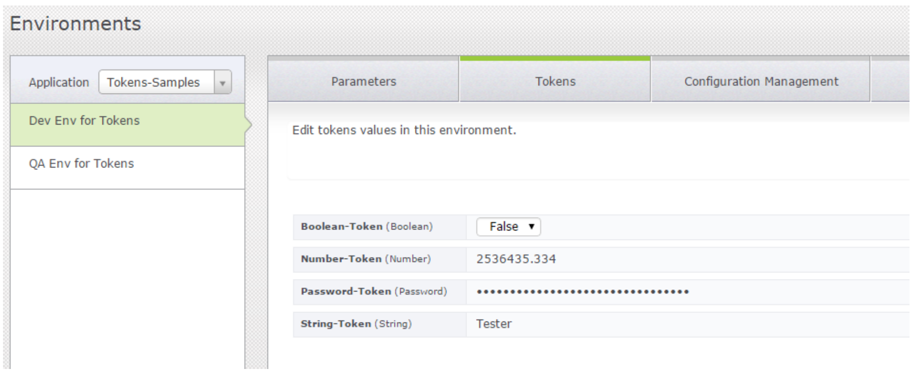


String-Token

Token Name	String-Token
Token Type	STRING
Default Value	Testing
Token Mask	Testing

Figure 2: Token definitions as shown in the RA ROC

After the tokens have been defined, you can setup the necessary values for the environments that you have already defined. This is accomplished by selecting "Environment→Parameter Configuration→Tokens".



Edit tokens values in this environment.

Boolean-Token (Boolean)	False
Number-Token (Number)	2536435.334
Password-Token (Password)	••••••••••••••••••••••••••••••
String-Token (String)	Tester

Figure 3: Token values at the environment level

Any environment level changes can be saved and be reflected for the given token.

Token Custom Masks

The custom masks for the Release Automation token are based off the JavaScript regular expressions (regex), below is the list of the regular expression syntax

Metacharacters supported

These are special characters that will be used for to control what type of character, how to search for a word, and other conditions.

Metacharacter	**Description**
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character

\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word
\0	Find a NUL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\uxxxx	Find the Unicode character specified by a hexadecimal number xxxx

Quantifiers supported

Quantifiers are used to match the desired number of characters.

Quantifier	**Description**
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*

n?	Matches any string that contains zero or one occurrences of *n*
n{X}	Matches any string that contains a sequence of *X n*'s
n{X,Y}	Matches any string that contains a sequence of X to Y *n*'s
n{X,}	Matches any string that contains a sequence of at least X *n*'s
n$	Matches any string with *n* at the end of it
^n	Matches any string with *n* at the beginning of it
?=n	Matches any string that is followed by a specific string *n*
?!n	Matches any string that is not followed by a specific string *n*

Brackets supported

The brackets are used to find the range of characters.

Expression	**Description**
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any digit between the brackets
[^0-9]	Find any digit NOT between the brackets
(x|y)	Find any of the alternatives specified

The most common JavaScript regular expressions that you will see being used are:

Expression	Description
.	Matches any character
^regex	Finds regex that must match at the beginning of the line.
regex$	Finds regex that must match at the end of the line.
[abc]	Set definition, can match the letter a or b or c.
[abc][vz]	Set definition, can match a or b or c followed by either v or z.
[^abc]	When a caret appears as the first character inside square brackets, it negates the pattern. This ccontent/an match any character except a or b or c.
[a-d1-7]	Ranges: matches a letter between a and d and figures from 1 to 7, but not d1.
X|Z	Finds X or Z.
XZ	Finds X directly followed by Z.
$	Checks if a line end follows.

Based on the JAVA regular expressions above, we are going to provide some example to get you going.

Custom mask - number token

The following are examples of how to create the different number masks

- 999999.99 → Enter up to that value with two digits for the suffix value, i.e. 128474.45
- \d{6} → A number with a fixed value of 6 digits, i.e. 123456

- ^[-0-9].* → Enter a number (positive or negative) with a floating suffix, i.e. 234, -204.02, 93.394

Custom mask – string token

The following examples are for string masks

- ^#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})$ → Hex values with a minimum of 3 characters and a maximum of 6 characters, i.e. #3F0, #FFA
- (0?[1-9]|[12][0-9]|3[01])/(0?[1-9]|1[012])/((19|20)\d\d) → Date format in the form of DD/MM/YYYY, i.e. 10/03/2015
- (0?[1-9]|1[012])/(0?[1-9]|[12][0-9]|3[01])/((19|20)\d\d) → Date format in the form of MM/DD/YYYY, i.e. 03/10/2015
- ([01]?[0-9]|2[0-3]):[0-5][0-9] → Time in a 24 hour format, i.e. 18:20
- ^([0-9]|([1][0-2])):[0-5][0-9][[ ]]?([ap][m]?|[AP][M]?) → Time in a 12 hour format, i.e. 7:30 am
- ([^\s]+(\.?([jpg|png|gif|bmp]?|[JPG|PNG|GIH|BMP]?))$) → Image file extension, i.e. tester.jpg
- ^[a-z0-9_-]{3,15}$ → Username, i.e. rauser
- ^[_A-Za-z0-9-]+(\.[_A-Za-z0-9-]+)*@[A-Za-z0-9]+(\.[A-Za-z0-9]+)*(\.[A-Za-z]{2,})$ → email address, i.e. demouser@anycompany.com
- ^([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])$ → IP address, where you could just set this up to look at a specific subnet, if so desired

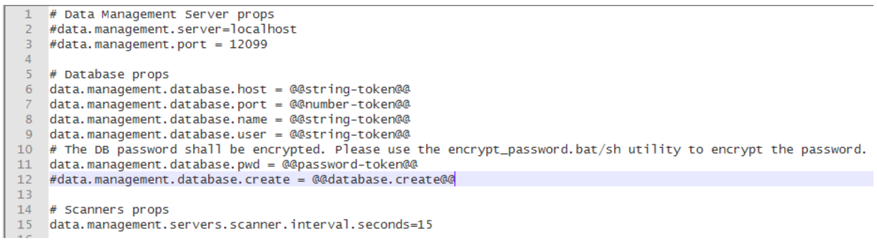
As it has been demonstrated above, the JAVA based regular expression are a powerful mechanism to setup the desired Release Automation token's custom mask.

Artifacts Using Tokens

Once you have defined the tokens, you can either modify existing files or create new files that will contain the token objects. A token object is defined as follows:

- @@TOKEN@@
- Given the token samples that I have created above, these values would represented as follows:
  - @@Boolean-Token@@
  - @@Number-Token@@
  - @@Password-Token@@
  - @@String-Token@@

An example of what the modified/created file that will handle Release Automation's tokens correctly is presented below.



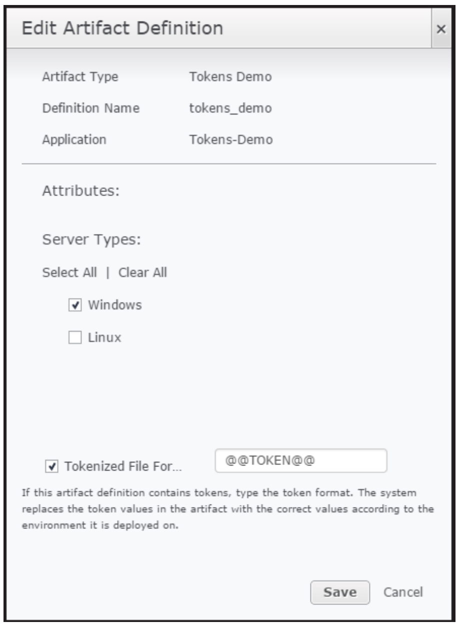
```
# Data Management Server props
#data.management.server=localhost
#data.management.port = 12099

# Database props
data.management.database.host = @@string-token@@
data.management.database.port = @@number-token@@
data.management.database.name = @@string-token@@
data.management.database.user = @@string-token@@
# The DB password shall be encrypted. Please use the encrypt_password.bat/sh utility to encrypt the password.
data.management.database.pwd = @@password-token@@
#data.management.database.create = @@database.create@@

# Scanners props
data.management.servers.scanner.interval.seconds=15
```

Figure 4: A file containing RA defined tokens

This represents how to setup the Release Automation's tokens in a file that has been added to the software package repository and it will be deployed and executed by Release Automation based on the defined tokens.

Tokenizing artifact type

For the artifact type that you have defined, you will need to "tokenized" for the artifact definition to perform the replacement of the token with the token contents for the given environment. Below is an example of how to "tokenized" the artifact type.



Edit Artifact Definition

Artifact Type	Tokens Demo
Definition Name	tokens_demo
Application	Tokens-Demo

Attributes:

Server Types:

Select All | Clear All

- [x] Windows
- [ ] Linux

- [x] Tokenized File For... @@TOKEN@@

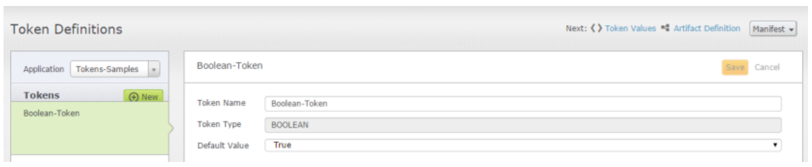
If this artifact definition contains tokens, type the token format. The system replaces the token values in the artifact with the correct values according to the environment it is deployed on.

Figure 5: Setting Tokenization for artifact type

By default the tokens have been setup to "@@TOKEN@@", which correlates to how the token tag has been defined in the artifact file. This can be changed to reflect your coding standards.

Tokens and Manifest Files

In the Release Automation's token facilities can also create the necessary manifest files, which can be used to setup the necessary token values at deployment time. Manifest files can be downloaded or uploaded via the Manifest drop-down option in the Token Definitions or Environment Parameter configuration.



Application: Tokens-Samples

Boolean-Token

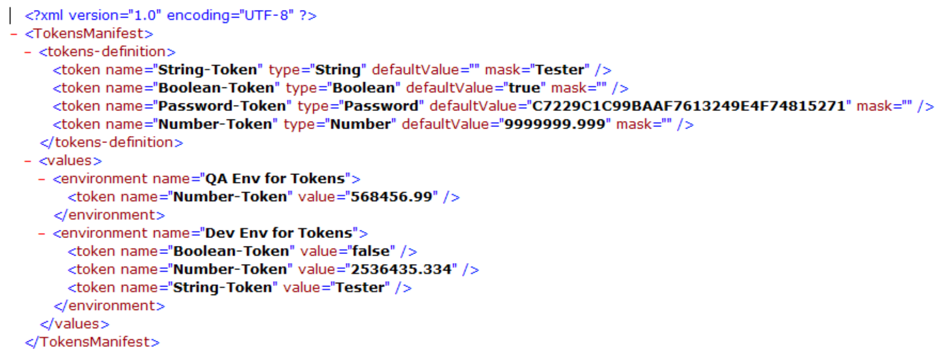
Token Name: Boolean-Token

Token Type: BOOLEAN

Default Value: True

Figure 5: Manifest drop-down on right-hand corner

Here is a sample of the tokens manifest that I downloaded for demonstration purposes.



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <TokensManifest>
  - <tokens-definition>
      <token name="String-Token" type="String" defaultValue="" mask="Tester" />
      <token name="Boolean-Token" type="Boolean" defaultValue="true" mask="" />
      <token name="Password-Token" type="Password" defaultValue="C7229C1C99BAAF7613249E4F74815271" mask="" />
      <token name="Number-Token" type="Number" defaultValue="9999999.999" mask="" />
    </tokens-definition>
  - <values>
    - <environment name="QA Env for Tokens">
        <token name="Number-Token" value="568456.99" />
      </environment>
    - <environment name="Dev Env for Tokens">
        <token name="Boolean-Token" value="false" />
        <token name="Number-Token" value="2536435.334" />
        <token name="String-Token" value="Tester" />
      </environment>
    </values>
  </TokensManifest>
```

Figure 6: Sample Token Manifest File

Manifest files provide you with the flexibility of determining the correct values for the token values that will be passed to the different artifacts for processing.

Best Practices

The following best practices will help you in determining the best method to follow in the usage of tokens.

Number of tokens to use

One of the first steps that you have to take prior to starting the creation of the tokens that will be used for a given application:

- Number of tokens needed
- How generic are these tokens?
- Are the tokens only specific to one environment?
- Do you need to modify the token mask from “@@TOKEN@@” to your coding standards?

Number of token manifest files to use

The number of token manifest files to be maintained, how many different manifest files are going to be utilizing for a given Release Automation application.

Artifacts affected by tokens

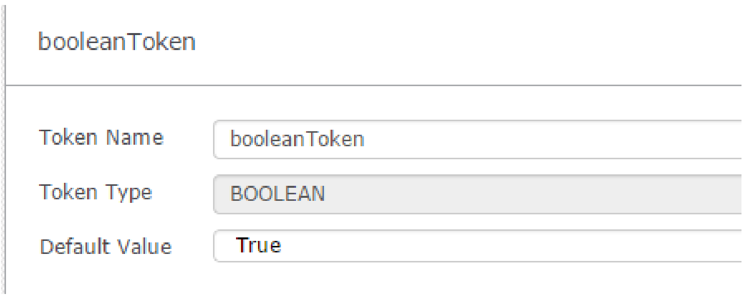
At the same time that you are determining the number of tokens to use for a given application, you need to pay attention to the number of software package artifacts that will be affected by these Release Automation tokens

It is recommended that you add in the comment for the artifact definition that this artifact makes use of Release Automation tokens for its successful processing.

Simple Use case

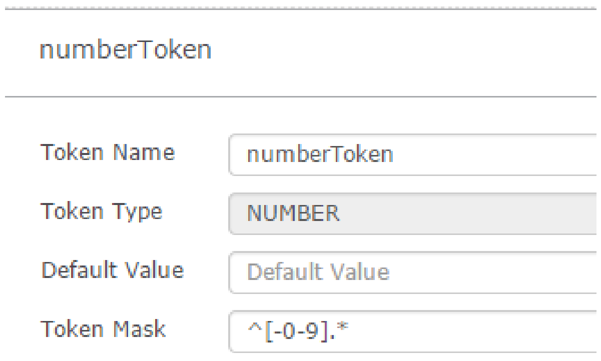
To help you in the understanding of how to use the Release Automation token functionality, we are going to create a very simple that performs a series of delay actions lasting about 30 seconds each and adding a user input action to the flow and processes to see how the token values will be updated.

- Create a Release Automation application
- Setup the corresponding server types, architectures, and environments
- Access the "Administration→Token Management" in the ROC (Release Operations Center)
- Create the following tokens



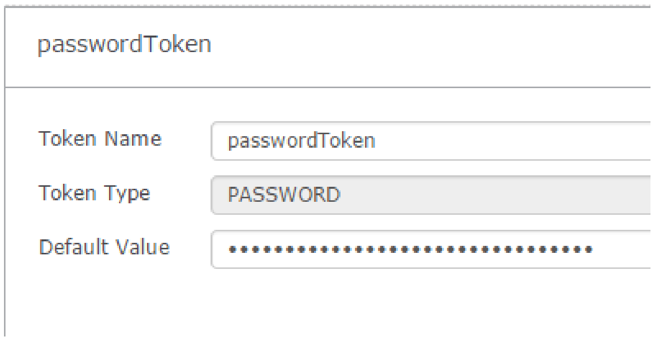
booleanToken

Token Name	booleanToken
Token Type	BOOLEAN
Default Value	True

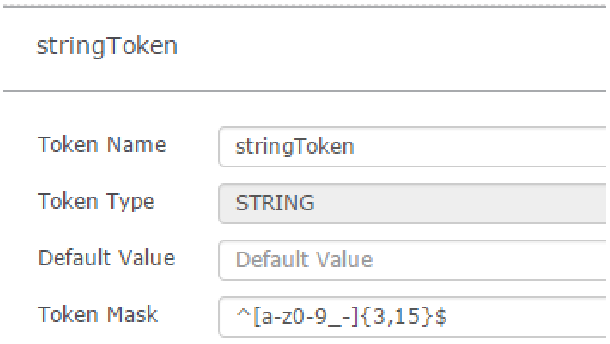


numberToken

Token Name	numberToken
Token Type	NUMBER
Default Value	Default Value
Token Mask	^[-0-9].*

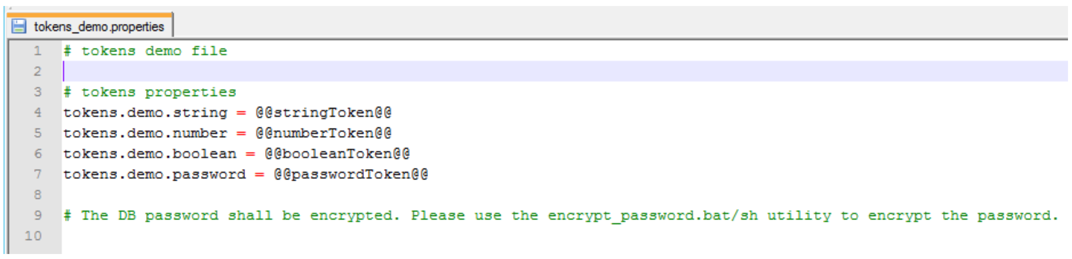

passwordToken

Token Name	passwordToken
Token Type	PASSWORD
Default Value	••••••••••••••••••••••••••••••••


stringToken

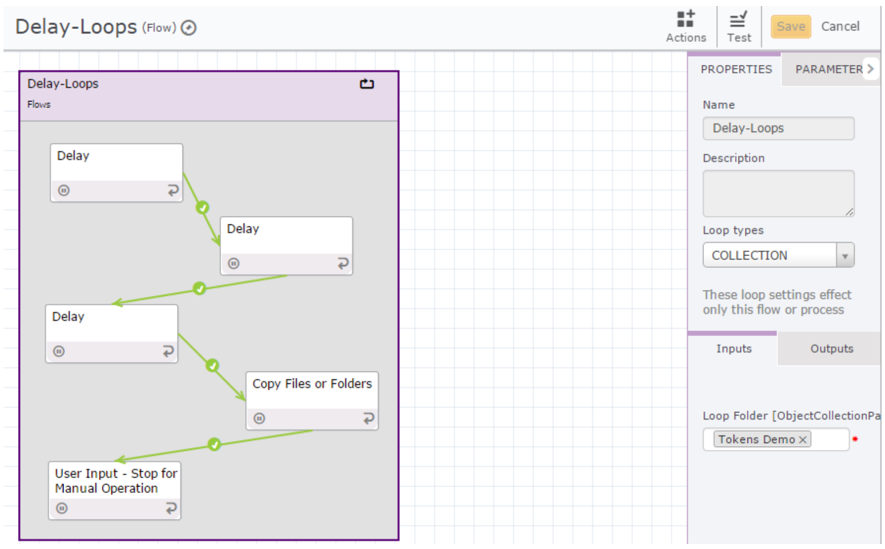
Token Name	stringToken
Token Type	STRING
Default Value	Default Value
Token Mask	^[a-z0-9_-]{3,15}$

- Now we need to add entries to the tokens for the different environments, this can be performed at the “Environment→Parameter Configuration→Tokens”
- Now move to Artifacts
- Create a new artifact type and call it “Tokens_Demo.properties” as shown below.


tokens_demo.properties

```
# tokens demo file

# tokens properties
tokens.demo.string = @@stringToken@@
tokens.demo.number = @@numberToken@@
tokens.demo.boolean = @@booleanToken@@
tokens.demo.password = @@passwordToken@@

# The DB password shall be encrypted. Please use the encrypt_password.bat/sh utility to encrypt the password.

```

- Make sure that this resides in the “C:\temp” of the Release Automation server prior to creating an artifact definition for it
- Map the components that you have created in this application to this artifact type, and make sure that you select the “Tokenized File for… @@TOKEN@@”
- Add an artifact definition based on the file name above
- Add a version to this artifact definition, remember where the file presently resides in the Release Automation server, since it will be a local file
- Now, we need to switch to the “Designer→Process Design” and add the following actions: delay, copy files or folders, and User Input – Stop for Manual Operation
- Create a flow and call it “delay-loops”



Name: Delay-Loops

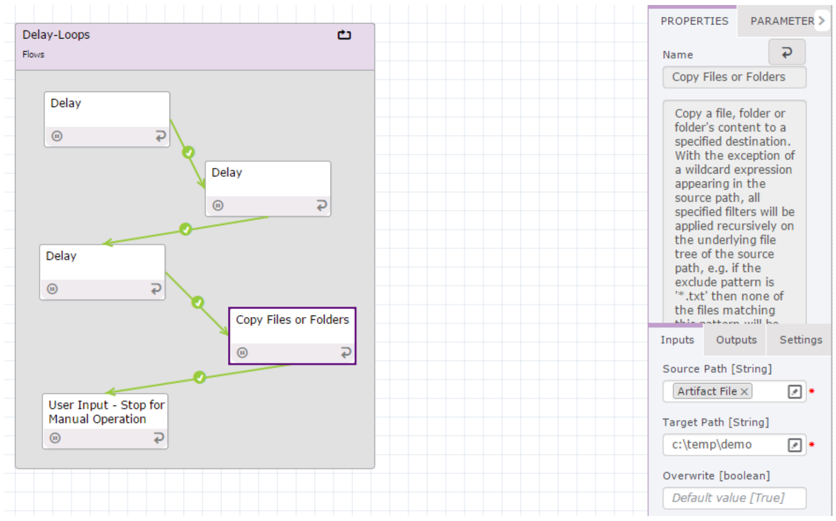
Description:

Loop types: COLLECTION

These loop settings effect only this flow or process

Loop Folder [ObjectCollectionPa: Tokens Demo

- Setup the flow to utilize a loop type: COLLECTION and the input is going to be the "Tokens Demo" artifact type



Name: Copy Files or Folders

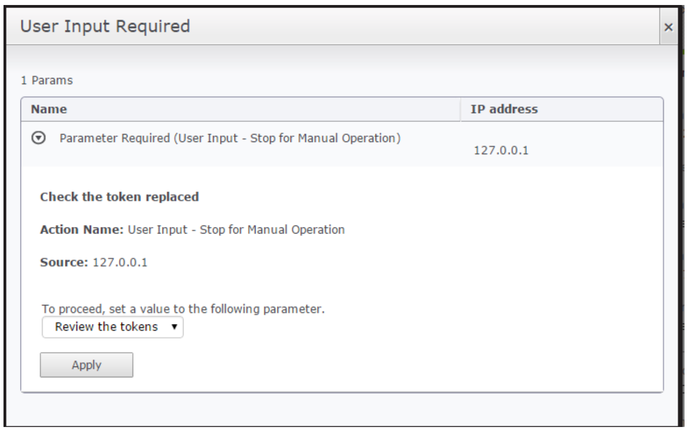
Copy a file, folder or folder's content to a specified destination. With the exception of a wildcard expression appearing in the source path, all specified filters will be applied recursively on the underlying file tree of the source path, e.g. if the exclude pattern is '*.txt' then none of the files matching

Source Path [String]: Artifact File

Target Path [String]: c:\temp\demo

Overwrite [boolean]: Default value [True]

- The source path field in the "Copy Files and Folders" action that is part of the flow needs to have the "Artifact File" parameter needs to be added here
- Create the "c:\temp\demo" folder in the Release Automation server
- Create a process "Tokens Demo" and publish it
- Now we need to switch to "Releases→Template Categories" and create the following
  - Template category
  - Deployment template
  - Add the process that you have already published, and add it as deployment step
  - Deployment plan
  - Create a project
  - Create a new deployment off the deployment template
  - Add an artifact package based off the artifact type that you created previously
  - Create a deployment
- Once the deployment is running and you get to the user input action



Parameter Required (User Input - Stop for Manual Operation) 127.0.0.1

Check the token replaced

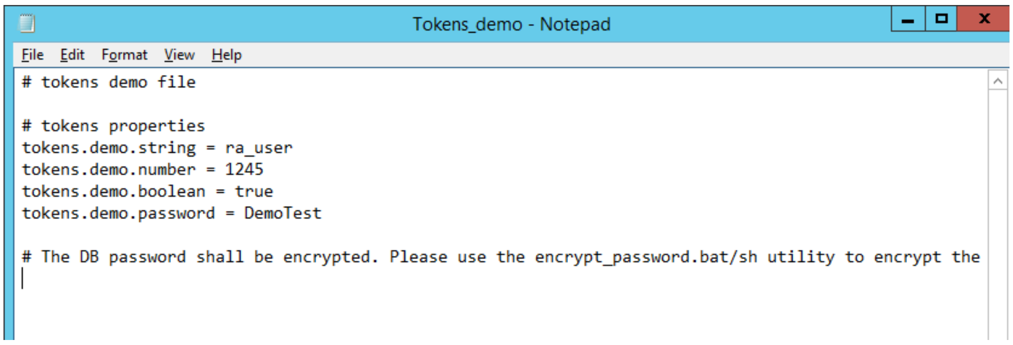
Action Name: User Input - Stop for Manual Operation

Source: 127.0.0.1

To proceed, set a value to the following parameter.

Review the tokens

- You can switch to the RA server and check the status of the file that has been placed in the "c:\temp\demo" folder
- The contents of the downloaded file will be as shown below



```
# tokens demo file

# tokens properties
tokens.demo.string = ra_user
tokens.demo.number = 1245
tokens.demo.boolean = true
tokens.demo.password = DemoTest

# The DB password shall be encrypted. Please use the encrypt_password.bat/sh utility to encrypt the
```

This demonstrate the power of the tokenization of the different values being passed in the different environments that you have setup for your Release Automation application.

Copyright Notice

Useful Links

https://wiki.ca.com/display/RA55/Tokens

https://wiki.ca.com/display/RA55/CA+Release+Automation+Home

Mozilla JavaScript Guide Regular Expressions