Auto-App-Configure

Documentation and usage

Application Author: Dan Anderson

Documentation Updated: 09-JUN-2017

By: Dan Anderson

Table of Contents

[Application Inventory (Folders and Files): 2](#_Toc484779771)

[App-config-sample 2](#_Toc484779772)

[App-settings 2](#_Toc484779773)

[Modules 2](#_Toc484779774)

[Script-config 2](#_Toc484779775)

[Directory Descriptions: 3](#_Toc484779776)

[App-config-sample 3](#_Toc484779777)

[App-settings 3](#_Toc484779778)

[Modules 3](#_Toc484779779)

[Script-config 3](#_Toc484779780)

[File Descriptions: 4](#_Toc484779781)

[Modules/token-utils.psm1 4](#_Toc484779782)

[Script-config/<your-app-config>.config 4](#_Toc484779783)

[App-settings/<your-settings-file>.csv 6](#_Toc484779784)

[Usage Examples: 7](#_Toc484779785)

# Application Inventory (Folders and Files):

## App-config-sample

* Bin
  + ConnectionStrings.config
  + LoggingConfiguration.config
* Web.config

## App-settings

* QA
  + <Region>
    - Common
    - <app specific>
* TESTING
  + <region>
    - Generic settings files (for testing and examples).

## Modules

* Token-utils.psm1

## Script-config

* <application configurations>

Auto-app-configure.ps1

ReadMe.docx

# Directory Descriptions:

App-config-sample:

* This directory contains sample files for testing.
* Useful for ensuring a key is replaced properly before making a live change.
* Additional files may be added or removed at any point. The application is not dependent on any file in this directory.

App-settings:

* This is where all the key/value pairs live when the application is running.
* The sub-folder structure can be safely re-arranged at any time, as long as the path to the settings files are updated in the: script-config/<your-app-config> file.
* If this directory is renamed or moved, the application script (auto-app-configure.ps1) will need to be updated so it can find the settings files.

Modules:

* This directory contains modules required by the application to function properly.
* If this directory is renamed or moved, the application script (auto-app-configure.ps1) will need to be updated so it can find the modules.

Script-config:

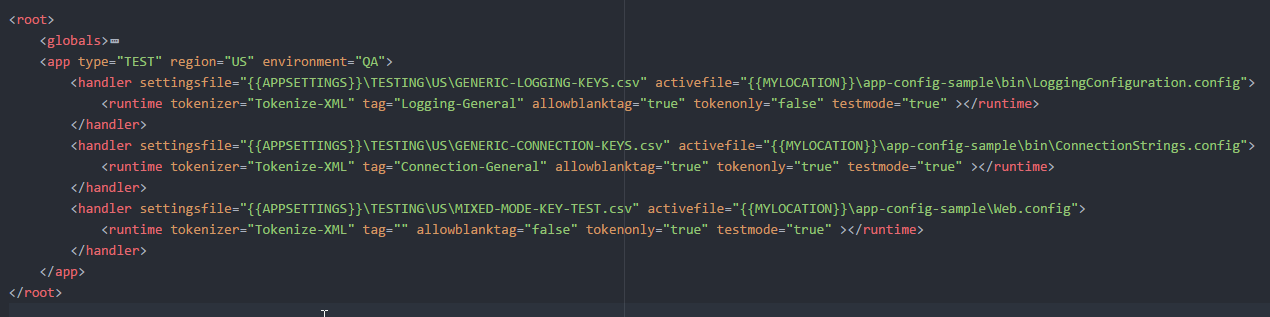
* This is where the main configuration files live.
* The <your-app-config>.config file is critical to the application and it will not function properly without it.
* This folder exists to keep all the settings files organized.
* Sub-folders can be added if required.
* When running the application, a “-ConfigFile” parameter must be passed during runtime. Technically the config file can be anywhere visible to the PowerShell process, however, keeping all configuration files located within this directory is considered best-practice.

# File Descriptions:

Modules/token-utils.psm1:

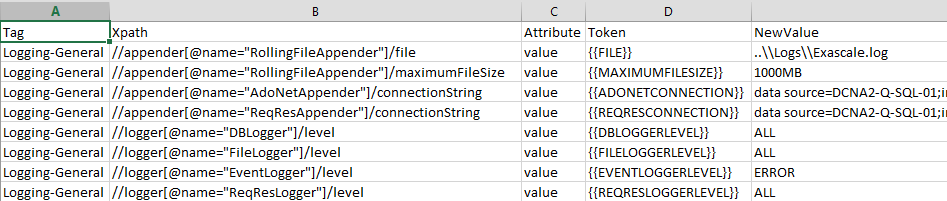
* This PowerShell module contains functions for interacting with configuration files.
* The script-config/<your-app-config>.config file has properties for instructing the main application (auto-app-configure.ps1) to call one of the functions in this module.
* Functions can be added to this module at any point, however alterations to existing functions should be taken with care. They are designed to be very generic and operate over a wide set of use cases (generally type specific, like XML files, JSON files, standard text files, etc...).
* File specific (vs. type specific) additions should be addressed with their own function, or handled outside of these module functions.

Script-config/<your-app-config>.config:

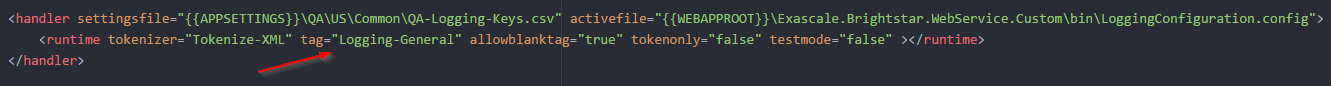


* This file can be considered the “glue” for the application and controls its behavior.
* It’s a standard XML file with a custom structure:
* **<root>**
  + The root element. Doesn’t do anything significant, but is required for xml parsing. All other elements live inside <root>.
  + **<globals>**
    - Any script-wide configuration items (like a log location) can live here.
    - As of the last update, there are no active configurations. It exists as a placeholder for future updates.
  + **</globals>**
  + **<app type=”” region=”” environment=””>**
    - This serves as the 1st level container.
    - It can accept as many attributes as required, but they do not serve any logical function.
    - Type, Region, and Environment attributes are included by default to show what application the configuration belongs to, but again, they don’t do anything in the code.
    - We can specify as many <app>’s as we need. The application is designed to operate on every <app> element inside the configuration file, however it is recommended to just include one <app> element and create a new configuration file for additional <app>’s.
  + **<handler settingsfile=”” activefile=””>**
    - The <handler> element accepts 2 attributes:
      * **Settingsfile**: the path to the settings file we will use.
      * **Activefile**: the path to the configuration file on the target server
    - <handler>’s are 1:1. They accept a single settings file, and single active file.
    - There can be as many handlers as we need for a given <app>
    - A new <handler> should be created for every **activefile** we need to work with.
    - Settings files do not have to be unique and can be reused over multiple active files.
  + **<runtime tokenizer=”” tag=”” allowblanktag=”” tokenonly=”” testmode=””>**
    - <runtime> lives inside <handler> and is a 1:1. There can only be a single runtime element for each handler. It accepts the following attributes:
      * **Tokenizer**: the tokenizer function to run from modules/token-utils.psm1
      * **Tag**: can be an empty string, or can contain a value. If this attribute contains a value, the tokenizer will only apply the settings with this specific tag inside the settings file. When left as an empty string, the tokenizer will attempt to set all values inside the settings file regardless of the tag.
      * **Allowblanktag**: can be true or false. This setting adds a little more flexibility to the tag attribute. When this setting is “true” the tokenizer will also include all keys in the settings file which have a blank tag. When “false” the tokenizer will only apply the keys in the settings file with a matching tag. Note: If the tag attribute is an empty string, this setting does nothing.
      * **Tokenonly**: can be true or false. If true, the tokenizer will write out the values in the “token” column of the settings file rather than replacing the values. Useful for troubleshooting, or testing which values in the activefile will get replaced.
      * **Testmode**: can be true or false. When set to true, the original activefile will not be replaced. Instead, the tokenizer will create a new file (in the same file path) with a “.test” extension. Useful for validating changes and ensuring values are safe before making the full replacement.
  + **</runtime>**
  + **</handler>**
  + **</app>**
* **</root>**

App-settings/<your-settings-file>.csv:



* These files are how the tokenizer functions know what to replace in a given activefile (see Script-config/<your-app-config>.config section).
* This is just a standard .csv file with a specific column structure understood by the application.
* A settings file must have the following columns:
  + **Tag**: this column works with the <your-app-config>.config file. It corresponds directly to the “tag” attribute in the config file. This column can be blank.



* + **XPath**: this column instructs the tokenizer how to find the setting in the activefile we would like to replace. It just uses standard xpath commands.
  + **Attribute**: Once the tokenizer finds the element(s) in the activefile, this column tells the tokenizer which attribute we should replace.
  + **Token**: this column is only used when tokenonly=”true” in the configuration file. The tokenizer will insert this value instead of the “NewValue” column. Useful for testing and ensuring the xpath is finding the correct elements in the active file.
  + **NewValue**: this is what the tokenizer will insert into the element/attribute found by the xpath if tokenonly=”false” in the configuration file.

# Usage Examples:

Executing this application is very simple. Everything is contained within the <your-app-config>.config and the settings files. In the simplest case, all you need to do is run the application with a config file:

.\auto-app-configure.ps1 –ConfigFile “<path-to-config-file>”

The script also accepts the following parameters at runtime. NOTE: any combination of the following parameters can be passed in during runtime.

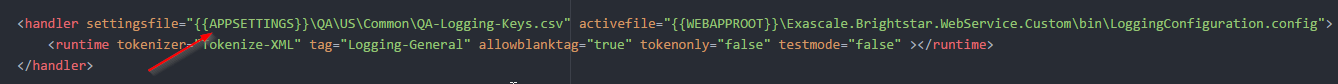
* -**ConfigFIle**: the path to the configuration file. This is the only required parameter. Usually located in the “script-confg” directory. NOTE: If running this application from an automated process, relative file paths don’t work very well. This is more of a PowerShell limitation vs. a code limitation.

.\auto-app-configure.ps1 –ConfigFile “<path-to-config-file>”

* -**ModuleDir**: if you relocate the directory of the required modules (not recommended), the script will need this filled in with the new location. By default, the script knows to look inside its own directory for the modules. This parameter does not need to be passed in if you don’t relocate the modules.

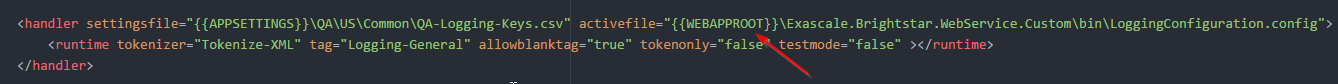
.\auto-app-configure.ps1 –ConfigFile “<some-config-file>”–ModuleDir “<path-to-the-modules>”

* -**AppSettingsDir**: This is a convenience parameter for the configuration files. Whatever is passed into this parameter will update the {{APPSETTINGS}} token in the configuration file. This is useful for settingsfile paths in the handlers. Example, if you have all settings files in “C:\Settings-Files” you can set the path inside the configuration file to {{APPSETTINGS}}, and pass – AppSettingsDir “C:\Settings-Files” during runtime. This parameter does not need to be passed in, and by default the application assumes you will be using the “app-settings” directory located within the default folder structure.



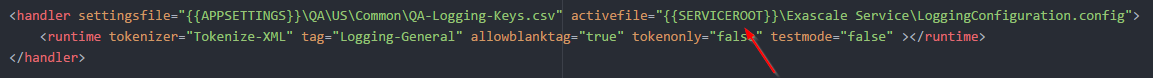
.\auto-app-configure.ps1 –ConfigFile “<some-config-file>”–AppSettingsDir “<path-to-the-app-settings>”

* -**WebAppRoot**: This is a convenience parameter for the configuration files. Whatever is passed into this parameter will update the {{WEBAPPROOT}} token in the configuration file. This is useful for activefile paths in the handlers. Example, if you have all components installed to “C:\inetpub\wwwroot” on the server, you can set the path inside the configuration file to {{WEBAPPROOT}}, and pass –WebAppRoot “C:\intetpub\wwwroot” during runtime. Where this functionality shines is if locations change on the server, you don’t need to update all the paths inside the configuration files.



.\auto-app-configure.ps1 –ConfigFile “<some-config-file>” – WebAppRoot “C:\inetpub\wwwroot”

* -**ServiceRoot**: This is a convenience parameter for the configuration files. Whatever is passed into this parameter will update the {{SERVICEROOT}} token in the configuration file. This is useful for activefile paths in the handlers. Example, if you have all components installed to “E:\Services” on the server, you can set the path inside the configuration file to {{SERVICEROOT}}, and pass – ServiceRoot “E:\Services” during runtime. Where this functionality shines is if locations change on the server, you don’t need to update all the paths inside the configuration files.



.\auto-app-configure.ps1 –ConfigFile “<some-config-file>” – ServiceRoot “E:\Services”