Reynold Clinton

clintonr@meredith.com

Venus Framework Installation guide

Table of Contents

[Install Git for Mac 2](#_Toc98267930)

[SSH Creation 2](#_Toc98267931)

[Provide SSH Key to Bitbucket 2](#_Toc98267932)

[Install Amazon Corretto Java JDK 8 3](#_Toc98267933)

[Set the JAVA\_HOME variable 3](#_Toc98267934)

[Installing Maven manually 4](#_Toc98267935)

[Set the MAVEN\_HOME variable 4](#_Toc98267936)

[NVM Installation 4](#_Toc98267937)

[Yarn Installation 5](#_Toc98267938)

[Grunt Installation 5](#_Toc98267939)

[Clone Venus Project 5](#_Toc98267940)

[Download and Install IntelliJ IDE (CE Edition) 6](#_Toc98267941)

[Add settings.xml 6](#_Toc98267942)

# Install Git for Mac

* Paste the following command to install Homebrew. This will provide the functionality to install multiple applications just by using “brew” command

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

* Now Install Git using the command:

$ brew install git

* Install Git-GUI (Optional)

$ brew install git-gui

# SSH Creation

* Open terminal and navigate to SSH Directory

$ cd ~/.ssh

* Generate public/private rsa key pair using your Meredith email address that is used to set your Bitbucket account

$ ssh-keygen -t rsa -C "<your\_meredith\_email@meredith.com>"

* Enter the file to save the key, or press Enter for the default location. Default is .ssh/id\_rsa.
* Press enter twice when prompted, or supply a passphrase
* Verify identification and public key files were created like: .ssh/id\_rsa and .ssh/id\_rsa.pub just by entering $ ls command

Text

Description automatically generated

# Provide SSH Key to Bitbucket

* Login to <https://bitbucket.prod.aws.about.com/projects/AUT/repos/venus/browse>
* Navigate to Profile 🡪 Manage account 🡪 SSH Keys
* Click on “Add SSH Keys”
* To open the key file, switch back to terminal and enter the following using your own public key:

$ open -e .ssh/id\_rsa.pub

* Copy the entire key and paste it on the Bitbucket and click on “Add Key”
* The Key gets added to Bitbucket as shown below:

Graphical user interface, application

Description automatically generated

# Install Amazon Corretto Java JDK 8

* Download the Mac .pkg file from <https://docs.aws.amazon.com/corretto/latest/corretto-8-ug/downloads-list.html>

Text

Description automatically generated

* Double click the downloaded file to start the installation wizard. Follow the steps in the wizard.
* Once the wizard completes, Amazon Corretto 8 will be installed in /Library/Java/JavaVirtualMachines/.

You can run the following command in a terminal to get the complete installation path.

/usr/libexec/java\_home –verbose

# Set the JAVA\_HOME variable

* Open terminal and type:

nano ~/.zshenv

* Add the following environment variable to the end of the **~/.zshenv** file

export JAVA\_HOME=/Library/Java/JavaVirtualMachines/amazon-corretto-8.jdk/Contents/Home

* Save and exit nano (ctrl-x 🡪 enter option-Y to save the file 🡪 hit Enter)
* Source the file

source ~/.zshenv

* Enter the following command to print $JAVA\_HOME:

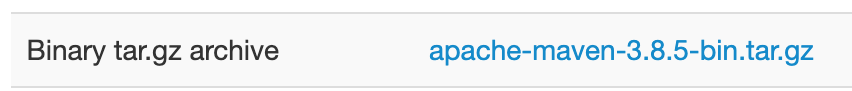
echo $JAVA\_HOME

* If we do see the below command, then we have successfully set your JAVA\_HOME environment variable

/Library/Java/JavaVirtualMachines/amazon-corretto-8.jdk/Contents/Home

# Installing Maven manually

* Download the file from <https://maven.apache.org/download.cgi>



* Type the following command to extract this Maven file. (First switch to the directory in the terminal where you have downloaded this file and then type the command):

tar xzvf apache-maven-3.8.5-bin.tar.gz

* Folder gets extracted to the default location if not move it to (/Users/{user}/ apache-maven-3.8.5)

# Set the MAVEN\_HOME variable

* Open terminal and type:

nano ~/.zshenv

* Add the following environment variable to the end of the **~/.zshenv** file

export MAVEN\_HOME=~/apache-maven-3.8.5

export PATH=$PATH:$MAVEN\_HOME/bin

* Save and exit nano (ctrl-x 🡪 enter option-Y to save the file 🡪 hit Enter)
* Source the file

source ~/.zshenv

* Enter the following command to print the $MAVEN\_HOME:

mvn -version

* If you do see the below command, then you have successfully set your MAVEN\_HOME environment variable

Text

Description automatically generated

# NVM Installation

* Enter any one of the following command in the terminal to install or update NVM.

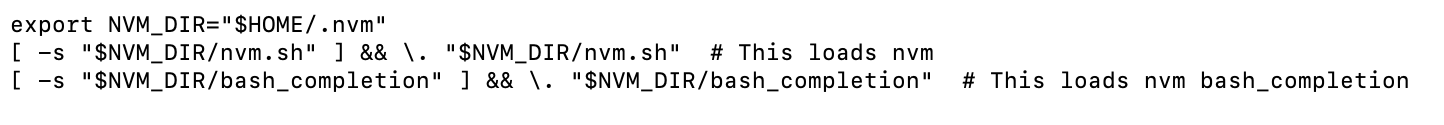
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash

or

wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash

* Running either of the above commands downloads a script and runs it. The script clones the nvm repository to ~/.nvm, and attempts to add the source lines to the correct profile file (~/.zshrc).
* You check any one of these files by again typing the command:

nano ~/.zshrc



* To verify that nvm has been installed, do:

command -v nvm

* Output will be – nvm
* Ensure that nvm was installed correctly with nvm --version, which should return the version of nvm installed.
* Install the version of Node.js you want:
  + Install the latest version with nvm install node
  + Use the latest version with nvm use node
  + Install the latest LTS version with nvm install --lts
  + Use the latest LTS version with nvm use --lts

# Yarn Installation

* Once we have nvm installed we can run the following both to install and upgrade Yarn:

npm install --global yarn

* Check that Yarn is installed by running:

yarn --version

# Grunt Installation

* When using nvm we do not need sudo to globally install a module with npm -g, so instead of doing sudo npm install -g grunt, do instead:

npm install -g grunt grunt-cli to install grunt on the current version of node.

# Clone Venus Project

* At first we must have an access to Dotdash Bitbucket repo <https://bitbucket.prod.aws.about.com/>
* Once the access is granted, go to this Bitbucket repo and click on “Clone”

Graphical user interface, text, application

Description automatically generated

* Copy the http URL <https://bitbucket.prod.aws.about.com/scm/aut/venus.git>
* In the terminal type the below command:

git clone https://bitbucket.prod.aws.about.com/scm/aut/venus.git

* Terminal will ask for the Bitbucket credentials
* If authorization is done, then it will download the Venus Project file in the default folder (/Users/{user}/Venus)

# Download and Install IntelliJ IDE (CE Edition)

* Open IntelliJ IDE and open this cloned Venus folder

File 🡪 Open 🡪 Venus

* Venus project will get imported into IntelliJ IDE

# Add settings.xml

* Right click on the Venus folder 🡪 maven 🡪 create ‘settings.xml’
* Append the following into this settings.xml file and Save the file

<mirrors>

<mirror>

<id>maven-public</id>

<name>About Maven Respository</name>

<url>http://nexus.prod.aws.about.com/repository/maven-public</url>

<mirrorOf>\*</mirrorOf>

</mirror>

</mirrors>

<profiles>

<profile>

<id>snapshots</id>

<repositories>

<repository>

<id>maven-snapshots</id>

<url>http://nexus.prod.aws.about.com/repository/maven-snapshots</url>

<releases><enabled>false</enabled></releases>

<snapshots><enabled>true</enabled></snapshots>

</repository>

</repositories>

</profile>

<profile>

<id>releases</id>

<repositories>

<repository>

<id>maven-releases</id>

<url>http://nexus.prod.aws.about.com/repository/maven-releases</url>

<releases><enabled>true</enabled></releases>

<snapshots><enabled>false</enabled></snapshots>

</repository>

</repositories>

</profile>

</profiles>

<activeProfiles>

<activeProfile>snapshots</activeProfile>

<activeProfile>releases</activeProfile>

</activeProfiles>

* Do a **mvn clean install** and Venus will have no errors there after.

Graphical user interface

Description automatically generated

Automation guidelines

# **Starter kit...**

* Identify an automated solution for setup.
* [Install Git](https://confluence.meredith.com/display/MQA/Install+Git) to your local machine.
* Setup Git security with [SSH Keys](https://confluence.meredith.com/display/MQA/Setup+SSH+Keys)
* Choose your Git interactivity, CLI or a Git Client (provide GitHub with you public SSH Key, and your Git Client with your private. [See SSH Keys Setup](https://confluence.meredith.com/display/MQA/Setup+SSH+Keys))
* Clone a project to your local machine. ([See SSH Keys Setup](https://confluence.meredith.com/display/MQA/Setup+SSH+Keys))

# **Automation Test Planning, Work Management, & Prioritization**

#### Test Planning

Using Confluence, QA Engineers can create a short-hand automation test plan that can help identify automation candidates, scope, gaps, and important product references. QA Engineers can use this as a blue print for creating and linking Jira issue artifacts (Suites, Cases) for tracing purposes.

* Confluence [Test Plans](https://confluence.meredith.com/display/MQA/Test+Plans) can be created to capture the scope of your automation. Creating a single page test plan provides insight for stakeholders and QA colleagues into what is automated, what is being worked on, and what needs automated from a bird's eye view. This document may be fluid and changing often.
* After a test plan is developed, Jira test suites (epics) and test scripts issue types can be created in the DQA project to match the test plan. You can even trace your Jira artifacts into your Confluence document.
* **Artifact naming practices (Jira Issues/Tickets):**
  + Keep it short
  + Place the most important words at the beginning
  + Eliminate filler words
  + Be clear and specific about the topic of the artifact (Plan, Suite, Case, Script, etc.)
  + Keep it simple and focused
  + Don't use redundancy of information that can be found in the details or description

#### Manage Work

* + QA Engineers can manage and prioritize their automation work using the [DQA JIRA backlog](https://jira.meredith.com/secure/RapidBoard.jspa?rapidView=7168&projectKey=DQA&view=planning.nodetail&epics=visible&issueLimit=100).
  + The backlog can allow QA Engineers organize their workload and goals.
  + Each sprint is setup by month, allowing QA Engineers to move their work where feasible.
    - Example: Known automation work for a product can be a few months away, however the QA Engineer can proactively establish an automation strategy by developing a Test Plan, Suite, Cases, etc... These can be planned for the expected month and moved according to when the work is priortized to begin development or completion.
  + The active board is the current months workload.

#### Prioritizing

* + The DQA project uses the MoSCoW methodology to indicate level of importance for this to become automated.
  + **MUST** Critical to the current implentation timebox in order for it to be a success. (note: scripts can be downgraded from MUST; for example, when new scripts are deemed more important).
  + **SHOULD** Important but not necessary for implementation in the current timebox. While SHOULD scripts can be as important as MUST, they are often not as time-critical or there may be another way to satisfy the requirement, so that it can be held back until a future implementation timebox.
  + **COULD** Desirable but not necessary, and could improve quality for little development cost. These will typically be included if time and resources permit.
  + **WOULD** The least-critical, lowest-payback items, or not appropriate at that time. As a result, WOULD/WAIT scripts are not planned into the schedule for the delivery timebox. WOULD/WAIT scripts are either dropped or reconsidered for inclusion in later timeboxes.

#### Story Pointing

* + This is used for a reference point for the QA Engineers to help plan their work loads.
  + There is no impact if the work takes more or less time than estimated.
  + Simple Story Pointing: (1-5) = (1-5) days or less to complete ticket.

#### JIRA: Test Suite (epic)

Within the DQA project, Epics are used as Test Suites. A TEST SUITE is a collection of test cases. In automated testing, it can mean a collection of test scripts.

**NOTE: Soon to be XRay Test Set**

* + Have the epic name match the summary (Desktop will be assumed as the default, so adding 'Mobile' is only really needed.)
  + Select the proper Digital group &  software Component the test suite belongs to.
    - DQA Maintenance → Work impacting all automation groups or projects.
    - Commerce → Work impacting only Commerce products and development.
    - Consumer Revenue → Work impacting Consumer Revenue products and development.
    - Content Platform → Work impacting Meredith back office solutions that are not public facing.
    - Digital Sites → Work impacting Meredith public facing products.
  + Provide a description of the test suite (epic) objective and scope.
  + Provide a test script priority (see Work Management & Prioritization)
  + Once automated development begins, make sure to set your Test Suite status to "In Progress"
  + **Issue links** can be used to associate any Tasks, Research & Development, Documentation, or Requirements. Refactoring existing code, creating a custom keyword, building a modular script, data file or source work. No need to associate DQA Automation Scripts or Test Cases as these are assumed as part of the 'Issues in Epic' section.
  + **Issues in Epic**are reserved for the Test Cases & Automated Test Scripts. An automation script "helper/common" created and used within another script would fall under Task and should show within the Issue Links. Issues in Epic is to depict the collection of tests being executed as part of the Test Suite.

#### JIRA: DQA Automation Script

**NOTE: Soon to be XRay Test**

* + Create these tickets within the associated Jira Test Suite (epic) or make sure to associate these tickets to the appropriate epic link (Test Suite/epic).
  + Provide a test script/case name, using artifact naming practices.
  + Provide a test script priority (see Work Management & Prioritization)
  + Select the proper Digital group &  software Component.
  + Set Story points by roughly estimating how long it will take to complete the automation required for the script. (see Work Management & Prioritization)
  + Provide a description of the test script objective, scope and any details of importance to this case.
  + Set the Sprint to the appropriate sprint Month this work is either planned to be worked on. There may be scripts that are planned that may not be ready or possible to complete in the current month, these tickets can be planned and set in advance based on the situation.

#### JIRA: Task

* + Create these tickets to address any existing automation updates, modifications, refactoring, etc. (Scripts, Data, Keywords, Folders...)
    - Tasks can be non-automation related as well.
    - If a new test case/script then this should be a DQA Automation Script.
  + Linking these tasks to an automation DQA epic is not required, but can be added optionally.
  + Provide a task name, using artifact naming practices.
  + Provide a task priority (see Work Management & Prioritization)
  + Select the proper Digital group &  software Component.
  + Set Story points by roughly estimating how long it will take to complete the task. (see [Work Management & Prioritization](https://confluence.meredith.com/display/MQA/Automation+Guidelines#AutomationGuidelines-WorkManagement&Prioritization))
  + Provide a description of the task objective, scope and any details of importance to this case.
  + Set the Sprint to the appropriate sprint Month this work is planned to be worked on. There may be tasks that are planned that may not be ready or possible to complete in the current month, these tickets can be planned and set in advance based on the situation.

#### JIRA: Issue Workflow

QAE Test Case

QAE Standard

HOURS LOGGING WILL MOVE TO A DIFFERENT PAGE OR SECTION?

* When logging hours for a task utilize the appropriate [DQA timesheet ticket](https://confluence.meredith.com/display/MQA/Timesheet) to log these hours instead of logging them to the task ticket. Examples: Proof of Concept task ([DQA-6207](https://jira.meredith.com/browse/DQA-6207)) , Automation Engineering task ([DQA-6211](https://jira.meredith.com/browse/DQA-6211)), etc.

HOURS LOGGING WILL MOVE TO A DIFFERENT PAGE OR SECTION?

# **Git Branching**

#### Git Happens

* Local Git is your local machine copies of Git branches.
* Remote Git are our copies of Git branches on GitHub.
* Remote 'Main' is our production environment, what automation points to and runs.
* To "checkout" a remote branch, is essentially to clone a copy of that branch to your local machine.
  + e.g., I want a copy of remote 'Main' locally, I would checkout remote 'Main'.
  + I want a copy of remote 'FEAT-1234', so I can help another QA Engineer, I would checkout remote 'FEAT-1234'.

#### Git Recommendations

* Try to keep your local 'Main' as in sync with remote 'Main' as possible. This requires checking out local 'Main', fetching any changes and pulling those changes from remote 'Main' to your local 'Main' branch.
* Avoid developing while local 'Main' branch is checked out. This branch is simply used to create new feature branches off of, and any changes made on local 'Main' will need to be discarded or removed.
* When creating a new feature branch, engineers should create from an updated local 'Main' and follow the Jira key from the project the test scripts exists prior to pushing remotely. e.g., COMMQA-123, DQA-456
  + There are instances where a feature branch may be created from another feature branch in order to replicate that branch, but for new development creating an updated local 'Main' is recommended.
  + "Test" feature branches can be created all the time, renamed, or deleted. This is often useful when troubleshooting a failure, proof of concept, or practicing.
  + Versioned feature branch names are useful in the event an update needs to be made to an existing script. e.g., a script is developed for ticket DQA-123, a pull-request is opened approved and merged. After merge, QA Engineer realizes they missed an assertion, instead of creating a new ticket, create a new feature branch DQA-123v2, a new pull-request, and merge. This work is still all under DQA-123 and traces back to this ticket allowing for better tracing and history in GitHub.
* New local feature branches will not be seen remotely (in GitHub) until after you push your commits.
* It is okay to push commits even if your work is not completed. All a Git push does is makes your code available by providing a copy to GitHub. This does not declare you're work is done by any means. So do not be shy about pushing. This will also allow other contributors help if you need it.
* Development changes on 'Main' cannot be pushed to directly to 'Main'. 'Main' is protected and requires a pull request with CODEOWNER reviews prior to merge. If you accidentally commit changes to your local 'Main', you will need to either revert the changes or delete your local 'Main' and checkout remote 'Main' to get another instance locally.
* When committing changes, take care to only commit changes from one Jira ticket per branch. Including work from multiple Jira tickets on the same branch makes it more difficult to review, and track work.
* Stay pruned! Try to keep your remotes cleaned up by only showing what actual remote branches are on GitHub. To do this, you will need to Git remote prune, or use your UI/Git Client (SourceTree has a check box when you fetch.)
* Delete unused or stale local feature branches. Once you are done with a feature branch locally (it has already merged to remote 'Main', or is no longer relevant), it is perfectly fine to delete it. In order to delete a local feature branch you cannot be checked out on the branch. So you will need to checkout another branch, and perform a git force delete on the local feature branch no longer in need.
* Commit messages should always begin with the Jira key name (e.g., COMMQA-123 Created new execution profile for stage environments.), followed with a meaningful message about the commit. Doing this traces the code change to the Jira ticket. Inside the Code section in Jira, you then can see any Git events regarding this ticket.
  + Commit messages should allude to what the QA member is adding, modifying, or deleting.

# **Coding Guidelines**

#### Code Optimization Recommendations

Utilize a **'Fail First'** approach to test case development to limit false failures. Find areas to improve our failure messages, making them helpful and descriptive. When automating a test case, purposely invoke the failure scenarios to verify the failure messages are satisfactory and fail as expected. This may involve running the automated test multiple times, modifying the test case to throw different errors that are commonly encountered.

* **Web Element not found**: Supply an incorrect object locator to verify the test fails, and verify the message clearly indicates what object was not found.
* **Value Comparison**: If the test is checking that some values are equivalent, or some String is contained in another String, supply an incorrect value and verify the test output is sufficient. For most cases it's preferred to have some output that indicates what the expected value is, and what the actual value was. Example:
  + Actual text 'LESLIE GRACE Diosa del Caribe' and expected text 'leslie grace diosa del' are not matched using regular expression
* **Custom Failure output**: If using KeywordUtil.markFailed(), make sure the custom error message adds detail to why something failed, and any values that were used to determine failures.
  + Limited example :  Fail to click on element
  + Descriptive example: Failed to click on element using object: Object Repository/Promocodesforyou/Home/caroselArea
  + Confirm that relevant test output is shown for the failure, and consider adding additional logging to make the failure more descriptive.
  + Explore the situations that make the test case fail, and modify the test case as needed.
  + Identify any risky areas that are more prone to errors/exceptions, and consider wrapping them in try/catch blocks that log any errors or exceptions thrown.
* **Delays:** Try to remove the unwanted delays in the script, to improve the efficiency of your scripts.
* **Test Local:** Test out the new feature before opening a Pull Request, Run the modified or new test cases on Katalon Studio locally to confirm expected test behavior.
* **Code Comments:** Commented out code should not be committed. While it is nice to be able to refer to some previous code, git can and should be used instead to reference previous code that is no longer needed. Some exceptions may apply to auto-generated files.
  + Comments in the code files should be added to help describe obscure or unclear functionality. Simple and short descriptions can also be nice for organization, but an overabundance of long comments may indicate that a piece of code needs to be broken into multiple functions. For code readability’s sake and maintenance, it is far preferred to separate logic into smaller well named functions instead of including more comments in the code.
* **White space:**White space should be limited, with unnecessary blank lines removed. In most cases, one blank line can be used to separate statements when needed, but should be consistent in a file.
* **Less is more:** Find ways to simplify code, or make it shorter. As long as it doesn't hinder the readability or functionality, less code is always easier to maintain.
* **Duplication:** Check that we aren't re-inventing any functions that are already available. Keep an eye out for duplication.
  + For test input files, check that a similar or identical file does not already exist.
  + For test objects, check that we don't already have a object in the Object Repository for that same element.
* **Groovy Camels:** Naming conventions should follow camelCase for Groovy/Java.
* **Def**can be used instead of an explicit type where appropriate, but mixing of def and type can be redundant and confusing.
* **Classes and methods** do not need the ‘public’ control modifier. Leaving no access control modifier will default to public.
* No **Semicolons**for groovy scripts. They are unnecessary to add.
* **Modularity!**Write the actions or execution steps that are to be performed on any webelement in a Method under keywords or any Script file which can be reused again.
  + Create a page specific or Content specific folders and create subfolders based on the targeted page for which we are going to write scripts similar to that of Page Object model structure for segregation.
  + Granularizing the scripts will help in easy maintenance of framework.
  + Maintain the hierarchy of TestSuites → Testcases → called Testcases ( Script methods/Common Scripts ) → Testscript.  If followed these steps it will be easy to debug and maintain the script or even for migration.

#### Custom Keywords

Creating Custom Keywords in Katalon Studio can be a helpful way to make code more reusable. Creating new custom keywords does however increase the size of our projects since packages and .cache files also need to be managed. It can also impact the speed in which the IDE opens.

Before creating new keywords, check that we don’t have any built in keywords that can satisfy the same purpose. These can be found in the documentation at <https://docs.katalon.com/katalon-studio/docs/index.html> -> Katalon Studio Enterprise -> Keywords. It’s also a good idea to check if a similar keyword already exists in the project, so we aren’t duplicating code.

Katalon specifies that we can create custom keywords for the following purposes:

* Creating reusable scripts
  + We want to make code reusable whenever possible, so keywords can be used for this purpose.
  + Keywords should generally take some input, otherwise we should consider if the keyword can be a reusable test script instead.
* Extending testing capability
  + Normal test scripts in Katalon Studio do not allow us to leverage all the object oriented principles that the groovy language offers, like Inheritance. If we need to implement object oriented principles that are not possible in the test scripts, we can leverage keywords for this purpose.
* Setting up testing projects with a specific pattern
  + Keywords can be used if we need to implement patterns that are not possible in normal test scripts. Generally we like to follow the patterns provided by Katalon Studio, but should the need arise we can use keywords for other patterns.

#### Package Names

To create new Custom Keywords, we will need to add a package if one is not already created: <https://docs.katalon.com/katalon-studio/docs/introduction-to-custom-keywords.html> . When naming packages, there are a few things to keep in mind:

* Packages should be all lower case to avoid conflict with the names of classes or interfaces.
* Generally, packages should avoid using hyphens or other special characters. If needed, underscores can be used.
* A common practice is to use the company's reversed domain name to begin the package. Example: com.example.mypackage
  + For our purposes, we can follow something similar using the site or project like com.element.mypackage, or com.onecms.mypackage

# **IDE Development**

#### Katalon Studio Enterprise

* Start KSE and open the desired project at the "repos > katalon.studio > projects" folder location.
* If the product under automation is not already in the project, create a folder with the product name in proper format within the Test Cases section of the Test Explorer.
  + When creating folders, please observe the folder hierarchy and structure to identify a proper location. Make sure the product or folder does not already exist.
* To import a Jira test script select the Jira Integration icon > select Import Test Case from JIRA JQL, and enter the Jira JQL (uncheck the BDD, see KSE BDD) and click OK.
  + To import a Jira BDD Gherkin syntax to a feature file, select the Jira Integration icon > select Import Test Case from JIRA JQL, and enter the Jira JQL and keep the include BDD feature check box checked. This will import a Katalon test script and a BDD feature file in the 'Includes' folder structure. ← **SD-354354**
  + Learn more about [Katalon and BDD](https://docs.katalon.com/katalon-studio/docs/cucumber-features-file.html#add-feature-files).
* Ready for KSE development

#### KSE Development:

* Only test scripts with the Jira key naming convention should be considered the test case. e.g., DQA-123 Search Product would be the test case, while LOGIN may be a global script used or called by other test cases to run. The LOGIN script on its own should not be pulled into a test suite.
* Build scripts to be dynamic and modular. Do not copy and paste a snippet of steps from script to script, instead create a "Common or Helper Script". This script does not need to be imported and can be created within KSE directly. Call this script to any imported Jira scripts. Now you can maintain this snippet in one file instead of multiple.
* Create and organize reusable and modular element objects in the object repository. e.g., Element search form field is used on all or most brands. Instead of creating an search field object for each brand and storing in each individual brand folder file, create a single search field object that is identifiable on each brand, and store in a global or common folder location like Element > Search, Global, or Common. Now any engineer looking for this object should easily be able to locate the object and maintain any element attribute changes.
* Keep data files consolidated as much as possible to avoid duplicate test data. When constructing test data, create in a way that is efficient and maintainable. If a product has variety of test data this can be broken up by Excel tabs and named appropriately. i.e., Element test data may include content types with various URLs, and advertising data, etc. The engineer can create an Element data source, with a tab containing a url for each brands content type on one tab, and another tab can contain advertising data. When the engineer wants to use this data, multiple data files can be created off a single data source. So the script for content type can map to one data file, and the advertising script can map to the other. A many to one relationship, as much as possible.

**Hybrid Testing Framework (Modular, Lib. Arc., & Data-Driven)**

#### Postman

* Start Postman and click on the Collections section on the left side. If starting a new API test suite, click the 'New' button on top the explorer and select 'Collection'. Name the collection after the Test Suite name, and use underscores instead of spaces for JS purposes. When creating folders, please observe existing Postman collection folder structures, and make sure the service or API collection does not already exist.
* If API or Service under automation already has a collection, to work within an existing collection, click the 'Import' button, then click the 'Upload Files' button, and navigate to your postman.newman repo folder path and find the desired Postman collection to import.
  + Please make sure prior to importing that you have the appropriate or desired GitHub branch checked out locally.
* Ready for Postman development.

#### Postman Development

* API tests that require multiple calls, commonly for setting up a data, please create a folder named after the test case (spaces allowed here), e.g., DQA-5206 Create Comment
  + Only provide the "DQA-1234" naming scheme to the actual request that you're asserting your test. Any setup scripts or post request scripts should be named to their purpose,
    1. Create Content
    2. Delay 3 seconds
    3. DQA-5206 Create Comment
    4. Read Comment
  + Recommendation for setup scripts is to create a "Common Setup" folder to locate these requests inside the test case folder.
    1. DQA-5206 Create Comment/Common Setup/Create Content
    2. DQA-5206 Create Comment/Common Setup/Delay 3 second
    3. DQA-5206 Create Comment/DQA-5206 Create Comment
    4. DQA-5206 Create Comment/Read Comment
  + Environment variables should be named using camelCase formatting.
    1. Keep the variable names small, simple, and intuitive. Make sure the variable name is not already used.
    2. Environment variables should be service or API specific.
  + Global variables should be named using CAPS\_AND\_UNDERSCORE\_SPACING formatting.
    1. Global variables should be invoking url paths, x-api-key headers, or other authentications.
    2. Please follow the existing naming conventions as current with other global variables.
  + To recognize your changes on Git, click on the Postman collection ellipses (three dots ...) or right-click on the collection, and select 'Export'.
    1. Keep the recommended Postman collection versioning and click the Export button.
    2. Locate the proper repository, and postman collection folder location, save.

# **Code Reviews**

* Code Reviews are conducted using GitHub pull requests. Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub.
* Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.
  + The person that opened the PR is responsible for getting their changes merged in a timely manner
    - Ping the dqa-engineering channel again if your PR has not been reviewed within 3 days
    - Merge the PR immediately after acquiring approvals
  + Merge the main branch into your branch daily and resolve any conflicts
  + Pull requests will alert the [#dqa](https://meredith.slack.com/archives/C3VQ8K3QF) Slack channel & CODEOWNERS of the repository
  + We use a topic or feature branch for our pull request. While you can send pull requests from any branch or commit, with a topic or feature branch you can push follow-up commits if you need to update your proposed changes.
  + When pushing commits to a pull request, don't force push. Force pushing can corrupt your pull request.
  + The pull request naming should follow the artifact naming practices & also using the Jira id ticket at the beginning (can include versioning) for traceability, e.g., "DQA-1234v3 Dinner Spinner Updates".
  + Inform the reviewer to use the Jira ticket (you may also provide a direct link) as a reference to what should be verified
  + Provide simple and useful information of what changes were made, purpose of changes, and other important information in the description.
  + Set an appropriate label to your pull request. The label should reflect the product or technical space to which changes were made to.
* Pull requests require 2 approvals prior to merge into 'main'. One (1) of the approvals must be from a CODEOWNER, and a second approval can come from any QA engineer or CODEOWNER.
* Conversations, Checks, and Files changed should all be reviewed.
  + Conversations provides a thread of branch changes, comments, conflicts, and other potential requests.
  + If the repository has checks, such as builds status, it's important to review whether the changes made the branch of this pull request are passing automation execution or building properly.
  + Files changed is the most important section, as this allows reviewers to look at the files changed. If there are deep-rooted file changes such as KSE .classpath, internal, or .project that appear to alter referencing paths, deletion, they should be question and marked with a 'request change' to prevent merge until addressed.
* Comments opened by reviewers should be responded to, or fixed accordingly on the feature branch. If needed, have a discussion with the reviewers to address the concerns, or provide explanation for why a concern is not an issue. If additional changes are needed to your topic or feature branch before you can merge, make the changes locally, test, commit, and push the changes.
  + After updates have been made to a pull request, use the **'Re-request review'** button  located to the right of each reviewer in GitHub. This will directly notify that reviewer to re-review the pull request. No need to message the reviewer directly unless time urgent.
  + Comments should be resolved before approval is complete
* Help identify misspellings in comments and code, or names that might be confusing
* Merge conflicts happen when you merge branches that have competing commits, and Git needs your help to decide which changes to incorporate in the final merge.
  + If your changes have merge conflicts with the base branch, you must address the merge conflicts before you can merge your pull request's changes.
  + To resolve a merge conflict, with your local topic or feature branch checked out, you need to merge local 'main' into your local topic or feature branch. Next, you need to locate the file(s) in conflict and resolve by using 'your' copy or 'their' copy. Finally, once resolved you can push the updated conflict to your remote topic or feature branch.
* For larger or more complex changes, consider checking out the branch being reviewed to your local machine, and run the tests being reviewed. Invoking failures on the test may also be helpful to check there is sufficient failure output. This is not mandatory but this extra step may be helpful in understanding how the test works and what is being tested, for someone that may not be as familiar.

CODEOWNERS are individuals or teams that are responsible for code in a repository, however are not soley responsible for code reviews. All QA Engineers should participate in Code Reviews providing comments, requests, or approvals.

# **Execution**

#### Katalon Studio Command Line Syntax

* Upon creating or updating your Test Suite to include your recent development changes, KSE allows you to generate a command line that can be fed into our automation building solutions.
  + [More on KSE Command Builder and line syntax](https://docs.katalon.com/katalon-studio/docs/console-mode-execution.html)
  + Not all arguments from the KSE generated command line are used, so please reference existing commands in use.
  + apiKey values are stored as repository secrets. Please reach out to [Jeff Kemble](https://confluence.meredith.com/display/~jkemble) or use your managers secret in examples below.

**Example: Scheduled**

|  |
| --- |
| katalonc -noSplash -runMode=console -projectPath="${{ github.workspace }}\katalon.studio\projects\Meredith Digital Site\Meredith Digital Site.prj" -retry=0 -testSuitePath="Test Suites/Ad Product/Karma/DQA-4239 Element Homepage Ads Mobile" -executionProfile="default" -browserType="iphone 8 plus" -apiKey= ${{ secrets.KATALON\_API\_KEY\_JTAYLOR }} --config -webui.autoUpdateDrivers=true |

**Example: Git Trigger (pull\_request)**

|  |
| --- |
| katalonc -noSplash -runMode=console -projectPath="${{ github.workspace }}\katalon.studio\projects\Meredith Digital Site\Meredith Digital Site.prj" -retry=0 -testSuitePath="Test Suites/Ad Product/Karma/DQA-4239 Element Homepage Ads Mobile" -executionProfile="default" -browserType="iphone 8 plus" -apiKey= ${{ secrets.KATALON\_API\_KEY\_JKEMBLE }} --config -webui.autoUpdateDrivers=true |

**Example: On-Demand**

|  |
| --- |
| katalonc -noSplash -runMode=console -projectPath="${{ github.workspace }}\katalon.studio\projects\Meredith Digital Site\Meredith Digital Site.prj" -retry=0 -testSuitePath="Test Suites/Ad Product/Karma/DQA-4239 Element Homepage Ads Mobile" -executionProfile="${{ github.event.inputs.environment }}" -browserType="${{ github.event.inputs.browserType }}" -apiKey= ${{ secrets.KATALON\_API\_KEY\_CKENNEDY }} -g\_mdpBrand="${{ github.event.inputs.brand }}" -g\_parameters="${{ github.event.inputs.parameters }}" -g\_path="${{ github.event.inputs.path\_override }}" --config -webui.autoUpdateDrivers=true |

#### GitHub Actions

* Automation can run on a scheduled cron, invoked by Git events, and can be triggered on demand at [G](http://9010-7658hx1:8080/)[itHub Actions](https://github.com/MeredithCorp/katalon.studio/actions) from the UI.
  + On demand runs are configurable, by branch, execution profile (environments), and device.
  + Scheduled runs are configured using cron, with the GitHub time zone set to UTC / GMT 0 (Iceland), e.g., 22h = 5 PM CST

* + Git event triggered runs are configured within the GitHub Action workflow by setting the Git event, paths to impacted files, and branch.
    - * Currently pull\_request on 'main' branch is set with path to files that are altered or modified.
* GitHub Action workflow file naming should not contain any spaces, instead use underscores. e.g., ads\_homepage\_adverstising.yml
  + Naming should begin with the component or feature under tests, and should be all lower case.
* Workflow name should begin with the component or feature under tests in all caps and tagged with an "@" sign followed by the test suite name.
* Jobs should not have any spaces, instead should user hyphens. e.g., ELEMENT-Homepage-Ads-Mobile-Regression
  + Naming should begin with the component or feature under test in all caps.
* Jobs should run on "self-hosted"; Using "self-hosted" will identified any of are self-hosted runners configured to this repository. Otherwise will pick up a GitHub runner.
  + Currently we have 8 VMs with 8 Katalon Runtime Engine (KRE) licenses that have been configured as GitHub self-hosted runners.
* Several steps are needed prior to executing KSE automation on a KRE.
  + Cloning of the katalon.studio repository to the runner.
  + Branch identification: helps the runner know what branch of code from GitHub you're wanting to execute your tests against.
  + Branch checkout: checks out the identified branch to execute code against.
  + Trigger identification: helps the runner establish which triggered step to run on (scheduled, git trigger, or on-demand). Conditions within these steps identify the trigger.
* Several steps are need after execution to clean up the runners.
  + Chromedriver check: looks to see if ChromeDriver is still active, and if so closes. This is needed as if ChromeDriver is not shutdown, it will prevent future runs.
  + Workspace cleanup: looks to remove recently cloned repositories on the runner. This prevents resource swelling and branch mix ups for future runs.

# **Monitoring & Troubleshooting**

Automation execution should be monitored closely. If a failure occurs, these should be addressed immediately prior to new automation development. The integrity of our automation is the most important component of our quality control. If executions are false failing constantly or failures are not addressed, our automation can be perceived as flaky, unreliable and we as a group lose trust in this service we provide not only to our group but to stakeholders.

#### Monitoring & Alerts

* **Alerts:** We have integrated Katalon Studio with Slack using a webhook. This allows us to alert specified channel/s for execution statuses.
  + All automation execution status alerts occur in the [#dqa](https://meredith.slack.com/archives/C3VQ8K3QF) Slack channel.
  + Links should be provided to the execution builds.
* **Monitoring:** We have several locations for monitoring automation execution results.
  + Katalon TestOps: Reports & Analytics → Test Runs provides almost real-time results.
  + GitHub: Readme page provides status badges of test suite executions and can be further reviewed within the Actions page.
  + Jira: Dashboard and Scripts, Katalon Studio integrations with Jira allowing execution results to display within Jira tickets.

* + MS Group Email: Executions using KSE will send emails on failure to [DigitalQualityAssurance@meredith.onmicrosoft.com](mailto:DigitalQualityAssurance@meredith.onmicrosoft.com)

#### Troubleshooting

* Troubleshooting failed automation, required identifying if the failure is due to a "True Fail" or a "False Fail".
  + **True Fail** occurs when our automation detects an actual bug or defect introduced.
  + **False Fail** means there may be no defect and the system/product may be working as expected. The test may be unclear whether the test case has passed or failed.
    - *False Fails* can occur for many reasons: script interaction with browser, locator changed, introduction of feature or behavior, dynamic behavior of application, etc.
* Katalon TestOps makes troubleshooting simple from the Reports & Analytics → Test Runs view.
  + Locate your test suite execution results by using filters.
  + Select a failed test execution to view the failing test results.
  + QA Engineers can collect valuable information when viewing a failed test.
    - Execution **environmental information**.
    - **Error** Message/s & Failed Assertions
    - **Screenshot** of the state of the application during failure
    - **Descriptive information** used during testing.
    - **Full logs**
    - **Similiar failures** allows further dive to determine if the failure is occuring in other scripts or is localised.
    - **Jira Defects** can be associated and/or logged from within the failed tests.
  + **API Performance** can be reviewed.
  + Review execution history can help pinpoint when failures occurred.
* Katalon TestOps can assist on identifying flakiness of tests, using a flakiness % metric.
  + The higher the flakiness %, the more unreliable the test/s. View the flakiness reports [here](https://testops.katalon.io/team/7198/project/17940/test-reports/test-maintenance/flakiness-report)(may require selecting the proper Katalon project).

Testplan

## **Smoketest -**[**DQA-6597**](https://jira.meredith.com/browse/DQA-6597)**- OneCMS Smoketest**IN PROGRESS

Percent completed as of 29 Nov 2021 -

#### Phase 1 - Content Types Smoketest -  [DQA-6499](https://jira.meredith.com/browse/DQA-6499) - OneCMS Content Type Smoketest - Phase 1 IN PROGRESS

Percent completed as of 29 Nov 2021 - 80% 12 completed/15 total

##### Scope of test:

##### Can successfully create and publish content type with bare minimum data (only required fields). (Only in DEV and TEST environments)

* Gallery -  [~~DQA-6459~~](https://jira.meredith.com/browse/DQA-6459) - OneCMS Gallery | Smoke Test **AUTOMATED**
* Block Article -  [~~DQA-6506~~](https://jira.meredith.com/browse/DQA-6506) - OneCMS Block Article | Smoke Test **AUTOMATED**
* Sweepstakes -  [~~DQA-6508~~](https://jira.meredith.com/browse/DQA-6508) - OneCMS Sweepstakes | Smoke Test **AUTOMATED**
* Author Page -  [~~DQA-6522~~](https://jira.meredith.com/browse/DQA-6522) - OneCMS Author Page | Smoke Test **AUTOMATED**
* Longform -  [~~DQA-6507~~](https://jira.meredith.com/browse/DQA-6507) - OneCMS Longform | Smoke Test **AUTOMATED**
* Projects -  [~~DQA-6533~~](https://jira.meredith.com/browse/DQA-6533) - OneCMS Gutenberg Projects | Smoke Test | Martha Stewart brand **AUTOMATED**   [~~DQA-6534~~](https://jira.meredith.com/browse/DQA-6534) - OneCMS CMB2 Projects | Smoke Test | BHG brand **AUTOMATED**
* Recipe -  [DQA-6514](https://jira.meredith.com/browse/DQA-6514) - OneCMS Recipe | Smoke Test | Allrecipes brand **QA CODE REVIEW**
* Category term -  [~~DQA-6516~~](https://jira.meredith.com/browse/DQA-6516) - OneCMS Category Term | Smoke Test **AUTOMATED**
* Category page -  [~~DQA-6517~~](https://jira.meredith.com/browse/DQA-6517) - OneCMS Category Page | Smoke Test **AUTOMATED**   [~~DQA-6518~~](https://jira.meredith.com/browse/DQA-6518) - OneCMS Food Aggregate | Smoke Test | Allrecipes brand **AUTOMATED**
* Creative work page -  [~~DQA-6520~~](https://jira.meredith.com/browse/DQA-6520) - OneCMS Creative Work Page | Smoke Test **AUTOMATED**
* Affiliate -  [~~DQA-6513~~](https://jira.meredith.com/browse/DQA-6513) - OneCMS Affiliate | Smoke Test **AUTOMATED**
* Newsletter -  [~~DQA-6515~~](https://jira.meredith.com/browse/DQA-6515) - OneCMS Newsletters | Smoke Test | People brand **AUTOMATED**
* Homepage (edit and update only - DO NOT PUBLISH!) -  [DQA-6535](https://jira.meredith.com/browse/DQA-6535) - OneCMS Homepage | Smoke Test **QA DEVELOPMENT**
* Merchant - [~~DQA-7043~~](https://jira.meredith.com/browse/DQA-7043) - OneCMS Merchants | Smoke Test | RealSimple brand **OBSOLETE**

**Phase 2 - Entities -**[**DQA-6596**](https://jira.meredith.com/browse/DQA-6596)**- OneCMS Entity Smoketest - Phase 2 IN PROGRESS**

Percent completed as of 29 Nov 2021 - 95% 18 completed/19 total

Scope of test:

Can successfully create and publish entity with bare minimum data (only required fields).

* Listicles -  [~~DQA-6537~~](https://jira.meredith.com/browse/DQA-6537) - OneCMS Listicle | Smoke Test **AUTOMATED**
* Site wide alerts -  [~~DQA-6538~~](https://jira.meredith.com/browse/DQA-6538) - OneCMS Site Wide Alerts | Smoke Test **AUTOMATED**
* Upload media -  [~~DQA-6539~~](https://jira.meredith.com/browse/DQA-6539) - OneCMS Upload Media | Smoke Test **AUTOMATED**
* Menus -  [~~DQA-6548~~](https://jira.meredith.com/browse/DQA-6548) - OneCMS Menu | Smoke Test **AUTOMATED**
* Plants -  [~~DQA-6540~~](https://jira.meredith.com/browse/DQA-6540) - OneCMS Plants | Smoke Test | BHG brand **AUTOMATED**
* Rules -  [~~DQA-6512~~](https://jira.meredith.com/browse/DQA-6512) - OneCMS Sweepstakes Rule | Smoke Test | Root level **AUTOMATED**
* Sponsorship -  [~~DQA-6511~~](https://jira.meredith.com/browse/DQA-6511) - OneCMS Sweepstakes Sponsorship | Smoke Test | Root level **AUTOMATED**
* Sponsor entity -  [~~DQA-6541~~](https://jira.meredith.com/browse/DQA-6541) - OneCMS Sponsor entity | Smoke Test | Root level **AUTOMATED**
* Checklists -  [~~DQA-6542~~](https://jira.meredith.com/browse/DQA-6542) - OneCMS Checklist | Smoke Test **AUTOMATED**
* Polls -  [~~DQA-6549~~](https://jira.meredith.com/browse/DQA-6549) - OneCMS Poll | Smoke Test **AUTOMATED**
* Quizzes -  [~~DQA-6550~~](https://jira.meredith.com/browse/DQA-6550) - OneCMS Trivia | Smoke Test **AUTOMATED** [~~DQA-6551~~](https://jira.meredith.com/browse/DQA-6551) - OneCMS Quiz | Smoke Test **AUTOMATED**
* Interactive Content -  [~~DQA-6552~~](https://jira.meredith.com/browse/DQA-6552) - OneCMS Interactive Content | Smoke Test **AUTOMATED**
* Celebrity Entity -  [~~DQA-6545~~](https://jira.meredith.com/browse/DQA-6545) - OneCMS Celebrity entity | Smoke Test | People brand **AUTOMATED**
* Dogs -  [~~DQA-6554~~](https://jira.meredith.com/browse/DQA-6554) - OneCMS Dogs | Smoke Test | Dailypaws brand **AUTOMATED**
* Cats -  [~~DQA-6553~~](https://jira.meredith.com/browse/DQA-6553) - OneCMS Cats | Smoke Test | Dailypaws brand **AUTOMATED**
* Baby Names -  [~~DQA-6544~~](https://jira.meredith.com/browse/DQA-6544) - OneCMS Baby name | Smoke Test | Parents brand **AUTOMATED**
* Author -  [~~DQA-6521~~](https://jira.meredith.com/browse/DQA-6521) - OneCMS Author | Smoke Test **AUTOMATED**
* Creative Work -  [~~DQA-6519~~](https://jira.meredith.com/browse/DQA-6519) - OneCMS Creative Work | Smoke Test | EW brand **AUTOMATED**

**Phase 3 - Tools -**[**DQA-6598**](https://jira.meredith.com/browse/DQA-6598)**- OneCMS Tools Smoketest - Phase 3 IN PROGRESS**

Percent completed as of 07 Jul 2021 - 0% 0 completed/5 total

Scope of Test:

Can perform basic functionality of the tool

* Edit search  [~~DQA-6605~~](https://jira.meredith.com/browse/DQA-6605) - OneCMS | EDIT SEARCH | SMOKE TEST **AUTOMATED**
* Bulk edit [DQA-6630](https://jira.meredith.com/browse/DQA-6630) - OneCMS | Bulk Classify | SMOKE TEST **QA CODE REVIEW**
* Recipe moderation dashboard [DQA-6631](https://jira.meredith.com/browse/DQA-6631) - OneCMS | Recipe Moderation | Smoke Test **QA CODE REVIEW**
* Image search -  [DQA-6904](https://jira.meredith.com/browse/DQA-6904) - OneCMS | Image Search | Smoke Test **QA CODE REVIEW**
* Subnavigation -  [DQA-6905](https://jira.meredith.com/browse/DQA-6905) - OneCMS | Subnavigation | Smoke Test **QA CODE REVIEW**

### Production Smoketest **-**[DQA-6599](https://jira.meredith.com/browse/DQA-6599) - OneCMS Prod Smoketest NEW

Scope of Test :

Percent completed as of 07 Jul 2021 - 0% 0 completed/7 total

##### Can successfully validate UDF, Content graphs for Published and Draft contents ( in PROD environment only)

* Gallery
* Block Article
* Galleries
* Longform
* Recipe (dependent on selected store)
* Category page
* Homepage (edit and update only - DO NOT PUBLISH!)

## **Regression**

### Gallery (Siva) [DQA-3245](https://jira.meredith.com/browse/DQA-3245) - OneCMS Gallery IN PROGRESS

Percent completed as of 07 Jul 2021 -

1. Publish Gallery using all slides  [~~DQA-3246~~](https://jira.meredith.com/browse/DQA-3246) - OneCMS Gallery Publish **AUTOMATED**
   * + Add Title, add test-dek, add image
     + Fill out the content and input image for each slides below  
       - Add image slide
       - Add recipe slide
       - Add instagram slide
       - Add shop slide
     + Set a primary, secondary category, and editorial program.
     + Select Meredith Taxonomy
     + Publish the Gallery
     + Capture Content Graph Log data.
     + Verify UDF + CG has no errors
2. Edit Gallery  [~~DQA-6436~~](https://jira.meredith.com/browse/DQA-6436) - OneCMS Gallery | Edit **AUTOMATED**
   * + Edit Title, edit test-dek, choose new image
     + Edit the content and input new image for each slides below  
       - Edit image slide
       - Edit recipe slide
       - Edit instagram slide
       - Edit shop slide
     + Set new primary, secondary category, and editorial program.
     + Select new Meredith Taxonomy
     + Save and Re-Publish the Gallery
     + Capture Content Graph Log data.
     + Verify UDF + CG has no errors
3. Gallery, Status and Visibility Tab Management  [DQA-6452](https://jira.meredith.com/browse/DQA-6452) - Gallery | Status & Visibility Tab Management **QA DEVELOPMENT**
   * Internal Metadata Feature
     + Create gallery with internal headline metadata.
   * Tags Feature
     + Select any Term.
   * URL Override Feature
     + Select any URL override.
   * Ad Tags Feature
     + Select ad tag term.
   * Clear Duplicates  
     + Click Clear Dupes button

       5.  Verify all Plugins work within OneCMS Gallery Component   [DQA-6434](https://jira.meredith.com/browse/DQA-6434) - OneCMS Gallery | Plugins **QA DEVELOPMENT**

* + - Bylines
    - Categories
    - Date & Time
    - Meredith Metadata
    - SEO Fields
    - Social Fields
    - Syndications
    - Tout Attributes
    - Taxonomy
    - UUID Fields

         8. Modify Permalink of the Gallery   [~~DQA-6489~~](https://jira.meredith.com/browse/DQA-6489) - OneCMS Gallery | Edit Permalink **AUTOMATED**

* + Edit the Permalink
  + Save and Update the Gallery
  + Verify UDF data is correct
  + Verify CG data is correct
  + Verify redirect rule is being created in the Redirect Manager

### Recipes (DQA-6421) -  [DQA-6421](https://jira.meredith.com/browse/DQA-6421) - OneCMS Recipe NEW

1. Add and Publish a new Recipe  [~~DQA-6422~~](https://jira.meredith.com/browse/DQA-6422) - OneCMS Recipe Publish **AUTOMATED**
   * Add Description, Recipe Title, Primary Media (Image or Video), Intro paragraph, and Recipe Media
   * Select Servings, Select Times, Select Ingredients (3-4 items is preferred)
   * Add Nutrition Header, Click Calculate Nutrition (which should pre-populated the values)
   * Add Instructions, Add Notes
   * Select MTAX terms, fill out all available fields
   * Select Primary Category, fill out all available fields
   * Save and Publish the draft
   * Verify UDF data is correct
   * Verify CG data is correct

       3. Edit Recipe  [~~DQA-6423~~](https://jira.meredith.com/browse/DQA-6423) - OneCMS Recipe Edit **AUTOMATED**

* + Edit Description, Recipe Title, Primary Media (Image or Video), Intro paragraph, and Recipe Media
  + Edit Servings, Edit Times, Edit and Select new Ingredients (3-4 items is preferred)
  + Edit Nutrition Header, Click Calculate Nutrition (which should pre-populated the values)
  + Edit Instructions, Edit Notes
  + Select new MTAX terms, fill out all available fields
  + Select new Primary Category, fill out all available fields
  + Save and Publish the draft
  + Verify UDF data is correct
  + Verify CG data is correct

       5. Modify Permalink of the Recipe  [~~DQA-6486~~](https://jira.meredith.com/browse/DQA-6486) - OneCMS Recipe | Edit Permalink **AUTOMATED**

* + Edit the Permalink
  + Save and Update the Recipe
  + Verify UDF data is correct
  + Verify CG data is correct
  + Verify redirect rule is being created in the Redirect Manager

      6. Verify all Plugins work within OneCMS Recipe Component  [DQA-6426](https://jira.meredith.com/browse/DQA-6426) - OneCMS Recipe | Verify Plugins **QA CODE REVIEW**

* + Bylines
  + Categories
  + Date & Time
  + Meredith Metadata
  + SEO Fields
  + Social Fields
  + Syndications
  + Tout Attributes
  + Taxonomy
  + UUID Fields
  + Verify the Recipe can be published without errors
  + Verify UDF data is correct
  + Verify CG data is correct

### Homepage (Siva) -  [DQA-7015](https://jira.meredith.com/browse/DQA-7015) - OneCMS Homepage Regression NEW

1. Edit Homepage - [DQA-7016](https://jira.meredith.com/browse/DQA-7016) - OneCMS | Regression | Edit Homepage **QA CODE REVIEW**
   * Edit existing homepage and update.
   * Capture Content Graph Log data.
   * Verify UDF + CG has no errors
2. Preview Homepage -  [~~DQA-7018~~](https://jira.meredith.com/browse/DQA-7018) - OneCMS | Regression | Preview homepage **AUTOMATED**
   * Preview homepage
   * Open with permalink

### Block Article (Lakshmi) [DQA-6916](https://jira.meredith.com/browse/DQA-6916) - OneCMS BlockArticle NEW

1. Publish Block Article  [~~DQA-6988~~](https://jira.meredith.com/browse/DQA-6988) - OneCMS | Publishing Block Article **AUTOMATED**  
   * Create new Block Article and publish.
   * Edit existing draft version of Block Article and publish.
   * Move to Trash feature
   * Preview after publishing
   * Create Revisions of existing Block Article
   * Add New Custom Fields
2. Edit block article

### Longform (Manoj)  [DQA-7000](https://jira.meredith.com/browse/DQA-7000) - OneCMS Longform NEW

1. Create Content
   1. From Addnew on left pane and From all longform
      * Assert the mandatory errors at initial launch
      * Redirection with error links
      * Title and Bylines
      * Product block creation
      * Image addition
      * Paragraph addition
      * Heading, Button, pull quote insertions
      * Browse all
   2. Settings and Configuration
      * Settings bar
      * Affiliate fields
      * OneCMS brand content
      * Bylines
      * Categories
      * Date and Time Information
      * Meredith Metadata
      * OneCMS sagas
      * SEO Fields
      * Social Fields
      * Syndication
      * Redirects
      * Touts
      * Taxonomies
      * UUID
   3. Validation of Content
      * Save and Update the Longform
      * Verify UDF data is correct
      * Verify CG data is correct
      * Verify redirect rule is being created in the Redirect Manager
2. Edit Existing Content
   * + Assert title and Dates
     + Title, Bylines modifications
     + Edit the Permalink
     + Edit Categories
     + Edit taxonomies
     + Edit the rest of configurations
     + Save and Update the Longform
     + Alter visibility  and assignment status
     + Verify UDF data is correct
     + Verify CG data is correct

### Sweepstakes, Rules and Sponsorships -  [DQA-7075](https://jira.meredith.com/browse/DQA-7075) - OneCMS Sweepstakes, Rules and Sponsorships NEW

Sponsorships

1. Create sponsorships -  [DQA-7054](https://jira.meredith.com/browse/DQA-7054) - OneCMS | Regression | Create Sponsorship **NEW**
2. Edit sponsorships -  [DQA-7057](https://jira.meredith.com/browse/DQA-7057) - OneCMS | Regression | Edit sponsorship **NEW**

Rules

1. Create rules -  [~~DQA-7063~~](https://jira.meredith.com/browse/DQA-7063) - OneCMS | Regression | Create rules **AUTOMATED**
2. Edit rules -  [~~DQA-7066~~](https://jira.meredith.com/browse/DQA-7066) - OneCMS | Regression | Edit rules **AUTOMATED**

Sweepstakes (Laxman)

1. Create sweepstakes
2. Edit Sweepstakes   [DQA-6818](https://jira.meredith.com/browse/DQA-6818) - OneCMS Sweepstakes Edit **QA CODE REVIEW**
   * Edit sweepstake title, Start date, expiration date
   * Edit entry page title, description
   * Edit primary media under Media block
   * Edit tout headline, tout description
   * Edit prize information title, prize value, prize limit, gift card value
   * Save and Publish the draft
   * Verify UDF data is correct
   * Verify CG data is correct
3. Create bulk daily sweepstakes
4. Verify all Plugins work within OneCMS Sweepstakes Component
   * UUID fields
5. Modify Permalink of the Sweepstakes [DQA-6820](https://jira.meredith.com/browse/DQA-6820) - OneCMS Modify Permalink of Sweepstakes **QA CODE REVIEW**
   * Edit the Permalink
   * Save and Update the sweepstake
   * Verify UDF data is correct
   * Verify CG data is correct
   * Verify redirect rule is being created in the Redirect Manager

### Category term and Category pages [DQA-6995](https://jira.meredith.com/browse/DQA-6995) - OneCMS Category NEW

1. Create category term -  [DQA-7025](https://jira.meredith.com/browse/DQA-7025) - OneCMS | Regression | Create category term **QA DEVELOPMENT**
2. Edit category term -  [DQA-7026](https://jira.meredith.com/browse/DQA-7026) - OneCMS | Regression | Edit category term **QA DEVELOPMENT**
3. Edit category variant/sub-variant -  [DQA-7027](https://jira.meredith.com/browse/DQA-7027) - OneCMS | Regression | Edit category variant **AUTOMATION CANDIDATE**
4. Edit Category Page -  [DQA-7019](https://jira.meredith.com/browse/DQA-7019) - OneCMS | Regression | Edit Category page **NEW**  
   * Edit the title
   * Edit the primary image
   * Edit the content for content block one
   * Edit the content for Related Content block
   * Edit tout headline, tout description
   * Edit the OneCMS social fields like Social Title, Social description, Facebook, Pinterest, Twitter
   * Click on Update
5. Bulk edit Category Page -  [DQA-7020](https://jira.meredith.com/browse/DQA-7020) - OneCMS | Regression | Bulk edit category page **NEW**  
   * Choose multiple category titles from All category pages
   * From the dropdown just beside the Apply button , Choose Edit and apply
   * BULK EDIT should appear, add the Tags value and click on update
6. Verify all plugins of Category Page -  [DQA-7021](https://jira.meredith.com/browse/DQA-7021) - OneCMS | Regression | Category pages | Plugins **NEW**  
   * Click on the three dots present at the rightmost corner of the page
   * Verify Plugins section is present under the three dots pop up window
   * Verify clicking on the plugins redirects to the desired plugin section
7. Verify the permalink of Category Page -  [DQA-7022](https://jira.meredith.com/browse/DQA-7022) - OneCMS | Regression | Category page | Permalink **NEW**  
   * Click on the view Category Page of Permalink section, verify it redirects to the correct Category Page
   * Verify UDF data is correct
   * Verify the CG data is correct

### Image features

### Scheduling and revisions -  [DQA-7028](https://jira.meredith.com/browse/DQA-7028) - OneCMS Scheduling and Revisions Regression IN PROGRESS

1. Schedule Gallery    [~~DQA-6487~~](https://jira.meredith.com/browse/DQA-6487) - OneCMS Gallery | Scheduling **AUTOMATED**
   * + Select future date on the right navigation pane
     + Save and Schedule the Gallery
     + Verify UDF data is correct
     + Verify CG data is correct
2. Create Revision Gallery  [DQA-6488](https://jira.meredith.com/browse/DQA-6488) - OneCMS Gallery | Regression | Revision **QA DEVELOPMENT**
   * Click + Create Revision on the right hand navigation tab
   * Edit any of the elements on the recipe
   * Save and publish the revision
   * Verify the revision saved properly by locating the new edits
3. Schedule revision gallery -  [DQA-7029](https://jira.meredith.com/browse/DQA-7029) - OneCMS Gallery | Regression | Schedule revision **NEW**
   * Click + Create Revision on the right hand navigation tab
   * Edit any of the data on the gallery
   * Edit date and select a future date from the publish panel
   * Schedule the revision
   * Verify the revision scheduled properly by locating the new edits
4. Schedule Recipe  [DQA-6485](https://jira.meredith.com/browse/DQA-6485) - OneCMS Recipe | Scheduling **QA CODE REVIEW**
   * + Select future date on the right navigation pane
     + Save and Schedule the Recipe
     + Verify UDF data is correct
     + Verify CG data is correct
5. Schedule Recipe of the day [DQA-6424](https://jira.meredith.com/browse/DQA-6424) - OneCMS Recipe | Recipe of the Day **NEW**
   * Select Recipe of the Day checkbox
   * Scheduled Entity box should appear (select starting and expiration dates)
   * Save and Publish the Recipe
   * Verify UDF data is correct
   * Verify CG data is correct
6. Create Revision Recipe [DQA-6425](https://jira.meredith.com/browse/DQA-6425) - OneCMS Recipe | Create Revision **NEW**  
   * Click + Create Revision on the right hand navigation tab
   * Edit any of the elements on the recipe
   * Save and publish the revision
   * Verify the revision saved properly by locating the new edits
7. Schedule revision recipe -  [DQA-7030](https://jira.meredith.com/browse/DQA-7030) - OneCMS Recipe | Regression | Schedule revision **NEW**
   * Click + Create Revision on the right hand navigation tab
   * Edit any of the data on the recipe
   * Edit date and select a future date from the publish panel
   * Schedule the revision
   * Verify the revision scheduled properly by locating the new edits
8. Schedule block articles -  [DQA-7033](https://jira.meredith.com/browse/DQA-7033) - OneCMS Block Articles | Regression | Scheduling **QA CODE REVIEW**
   * Select future date on the right navigation pane
   * Save and Schedule the block article
   * Verify UDF data is correct
   * Verify CG data is correct
9. Create revision block articles -  [DQA-7034](https://jira.meredith.com/browse/DQA-7034) - OneCMS Block Article | Regression | Revision **NEW**
   * Click + Create Revision on the right hand navigation tab
   * Edit any of the elements on the block article
   * Save and publish the revision
   * Verify the revision saved properly by locating the new edits
10. Schedule revision block articles -  [DQA-7031](https://jira.meredith.com/browse/DQA-7031) - OneCMS Block Articles | Regression | Schedule revision **NEW**
    * Click + Create Revision on the right hand navigation tab
    * Edit any of the data on the block articles
    * Edit date and select a future date from the publish panel
    * Schedule the revision
    * Verify the revision scheduled properly by locating the new edits
11. Schedule revision homepage -  [DQA-7032](https://jira.meredith.com/browse/DQA-7032) - OneCMS Homepage | Regression | Schedule revision **NEW**
    * Click + Create Revision on the right hand navigation tab
    * Edit any of the data on the homepage
    * Edit date and select a future date from the publish panel
    * Schedule the revision
    * Verify the revision scheduled properly by locating the new edits

### Syndication

### Redirects -  [DQA-7068](https://jira.meredith.com/browse/DQA-7068) - OneCMS Redirects NEW

1. Add New Permanant Redirect - [DQA-3603](https://jira.meredith.com/browse/DQA-3603) - OneCMS Add New Permanant Redirect **NEW**
   1. HTTP Status Code is set to 301. Successfully publish a new redirect.
2. Add New Temporary Redirect -  [DQA-3604](https://jira.meredith.com/browse/DQA-3604) - OneCMS Add New Temporary Redirect **NEW**    
   1. - HTTP Status Code is set to 302. Successfully publish a new redirect.
   2. - All 302 redirects require an expiration date when the redirect with expire.
3. Add Redirect with Tracking Parameters -  [DQA-3605](https://jira.meredith.com/browse/DQA-3605) - OneCMS Add Redirect with Tracking Parameters **NEW**    
   1. - Verify ability to add tracking parameters to the redirect URL.
4. Redirect Publish Dates -  [DQA-3606](https://jira.meredith.com/browse/DQA-3606) - OneCMS Redirect Publish Dates **NEW**    
   1. - Validate redirect publishing dates functionality.
   2. - Schedule a redirect data, and retro-actively set a redirect date.

### Content actions - search, filter, order, bulk/quick edit - [DQA-7035](https://jira.meredith.com/browse/DQA-7035) - OneCMS Content Actions NEW

1. Block Article  
   a. Content Search [DQA-6915](https://jira.meredith.com/browse/DQA-6915) - OneCMS Block Article Content Search **AUTOMATION CANDIDATE**  
   * Search content by name.
   * Search content by category.
   * Search content by keyword.

b. Content Filter [~~DQA-6917~~](https://jira.meredith.com/browse/DQA-6917) - OneCMS Block Article Content Filter **AUTOMATED**

* + Filter content by publishing status.
  + Filter content by Content Graph status.
  + Filter content by dates.
  + Filter by Content Creator

c. Content Order 

* + Order content by title. [~~DQA-6925~~](https://jira.meredith.com/browse/DQA-6925) - OneCMS Block Article Content Order- Title **AUTOMATED**
  + Order content by date. [~~DQA-6999~~](https://jira.meredith.com/browse/DQA-6999) - OneCMS Block Article Content Order- Date **AUTOMATED**

d. Content Action   [~~DQA-6926~~](https://jira.meredith.com/browse/DQA-6926) - OneCMS Block Article Content Action **AUTOMATED**

* + Quick Edit Content
  + Edit Content
  + Preview Content

1. Longform
   * Content Search [DQA-7001](https://jira.meredith.com/browse/DQA-7001) - OneCMS Longform Content Search **QA CODE REVIEW**
     + Count verification of articles
     + Search content by name.
     + Search content by category.
     + Search content by keyword.
   * Content Filter [DQA-7002](https://jira.meredith.com/browse/DQA-7002) - OneCMS Longform Content Filter **NEW**
     + Filter content by publishing status.
     + Filter content by Content Graph status.
     + Filter content by dates.
     + Filter by Content Creator.
   * Content Order
     + Order content by title. [DQA-7003](https://jira.meredith.com/browse/DQA-7003) - OneCMS Longform Content Order- Title **NEW**
     + Order content by date. [DQA-7004](https://jira.meredith.com/browse/DQA-7004) - OneCMS Longform Content Order- Date **NEW**
   * Content Actions [DQA-7005](https://jira.meredith.com/browse/DQA-7005) - OneCMS Longform Content Action **NEW**
     + Edit content with Bulk Actions feature.
     + Quick edit content.
     + Trash content.
     + View Content
2. Sweepstakes
   * Content Search
     + Search content by name.
     + Search content by category.
     + Search content by keyword.
   * Content Filter
     + Filter content by publishing status.
     + Filter content by Content Graph status.
     + Filter content by dates.
     + Filter by Content Creator.
   * Content Order
     + Order content by title.
     + Order content by date.
   * Content Actions
     + Bulk edit sweepstakes [DQA-6821](https://jira.meredith.com/browse/DQA-6821) - OneCMS Bulk edit Sweepstakes **AUTOMATION CANDIDATE**
     + Quick edit content.
     + Trash content.
     + Full edit and view content.
3. Category Pages
   * Content Search  [DQA-6994](https://jira.meredith.com/browse/DQA-6994) - OneCMS Category Page Content Search **NEW**  
     + Search content by name.
     + Search content by category.
     + Search content by keyword.
   * Content Filter [DQA-6996](https://jira.meredith.com/browse/DQA-6996) - OneCMS Category Page Content Filter **NEW**
     + Filter content by publishing status.
     + Filter content by Content Graph status.
     + Filter content by dates.
     + Filter by Content Creator.
   * Content Order
     + Order content by title.  [DQA-7012](https://jira.meredith.com/browse/DQA-7012) - OneCMS Category Content Order- Title **NEW**
     + Order content by date. [DQA-7013](https://jira.meredith.com/browse/DQA-7013) - OneCMS Category Content Order- Date **NEW**
   * Content Actions [DQA-7014](https://jira.meredith.com/browse/DQA-7014) - OneCMS Category Content Action **NEW**
     + Edit content with Bulk Actions feature.
     + Quick edit content.
     + Trash content.
     + View Content

### CMB2 Article

### Author Page [DQA-7125](https://jira.meredith.com/browse/DQA-7125) - OneCMS Author Page Works NEW

Publish Author Pages [~~DQA-7126~~](https://jira.meredith.com/browse/DQA-7126) - OneCMS | Regression | Publish Author Page **AUTOMATED**

1. Create Author Page
2. Add values to all the fields
3. Publish Author Pages

Edit Author Page [DQA-7128](https://jira.meredith.com/browse/DQA-7128) - OneCMS | Regression | Edit Author Page **NEW**

1. Edit Author Page
2. Edit the values of all fields
3. Update the Author Page

Delete Author Page [DQA-7129](https://jira.meredith.com/browse/DQA-7129) - OneCMS | Regression | Delete Author Page **NEW**

1. Delete Author Page
2. Verify that Author Page is deleted from All Author Page

### Project

1. Publish Project with all fields   [DQA-7134](https://jira.meredith.com/browse/DQA-7134) - OneCMS Project | Publish Project **NEW**
   * Tout Image
   * Bylines
   * Editorial Program
   * Meta Title/Description
   * Project Details
   * Project Tools
   * Project Materials
   * Project Cut List
   * Project Parts
   * Project Steps
   * Categories
   * Content Property
   * Sagas
   * Redirects
   * Social Media
   * Tout Attributes
   * Meredith Taxonomy
   * Meredith Metadata
   * SEO Metadata
   * Syndication
2. Edit Project along with editing all the fields in it [DQA-7135](https://jira.meredith.com/browse/DQA-7135) - OneCMS Project | Edit Project **NEW**
   * Tout Image
   * Bylines
   * Editorial Program
   * Meta Title/Description
   * Project Details
   * Project Tools
   * Project Materials
   * Project Cut List
   * Project Parts
   * Project Steps
   * Categories
   * Content Property
   * Sagas
   * Redirects
   * Social Media
   * Tout Attributes
   * Meredith Taxonomy
   * Meredith Metadata
   * SEO Metadata
   * Syndication
3. Verify all Plugins work within OneCMS Project Component   [DQA-7132](https://jira.meredith.com/browse/DQA-7132) - OneCMS Project | Plugins **NEW**
   * + - Bylines
       - Categories
       - Date & Time
       - Meredith Metadata
       - SEO Fields
       - Social Fields
       - Syndications
       - Tout Attributes
       - Taxonomy
       - UUID Fields

### Creative Work and Creative Works Pages -  [DQA-7045](https://jira.meredith.com/browse/DQA-7045) - OneCMS Creative Work NEW

Publish Creative Work

1. Create creative work, type = TV Show -  [DQA-7046](https://jira.meredith.com/browse/DQA-7046) - OneCMS | Regression | Create Creative Work - TV Show **NEW**
2. Create creative work, type = Movie -   [DQA-7049](https://jira.meredith.com/browse/DQA-7049) - OneCMS | Regression | Create Creative Work - Movie **NEW**
3. Create creative work, type = Music -  [DQA-7052](https://jira.meredith.com/browse/DQA-7052) - OneCMS | Regression | Create Creative Work - Music **NEW**
4. Create creative work, type = Book -  [DQA-7055](https://jira.meredith.com/browse/DQA-7055) - OneCMS | Regression | Create Creative Work - Book **NEW**
5. Create creative work, type = Stage -  [DQA-7058](https://jira.meredith.com/browse/DQA-7058) - OneCMS | Regression | Create Creative Work - Stage **NEW**
6. Create creative work, type = Video games -  [DQA-7061](https://jira.meredith.com/browse/DQA-7061) - OneCMS | Regression | Create Creative Work - Video Games **NEW**
7. Create creative work, type = Web series -  [DQA-7064](https://jira.meredith.com/browse/DQA-7064) - OneCMS | Regression | Create Creative Work - Web Series **NEW**

Edit Creative Work

1. Edit creative work, type = TV Show -  [DQA-7047](https://jira.meredith.com/browse/DQA-7047) - OneCMS | Regression | Edit Creative Work - TV Show **NEW**
2. Edit creative work, type = Movie -  [DQA-7050](https://jira.meredith.com/browse/DQA-7050) - OneCMS | Regression | Edit Creative Work - Movie **NEW**
3. Edit creative work, type = Music -  [DQA-7053](https://jira.meredith.com/browse/DQA-7053) - OneCMS | Regression | Edit Creative Work - Music **NEW**
4. Edit creative work, type = Book -  [DQA-7056](https://jira.meredith.com/browse/DQA-7056) - OneCMS | Regression | Edit Creative Work - Book **NEW**
5. Edit creative work, type = Stage -  [DQA-7059](https://jira.meredith.com/browse/DQA-7059) - OneCMS | Regression | Edit Creative Work - Stage **NEW**
6. Edit creative work, type = Video games -  [DQA-7062](https://jira.meredith.com/browse/DQA-7062) - OneCMS | Regression | Edit Creative Work - Video Games **NEW**
7. Edit creative work, type = Web series -  [DQA-7065](https://jira.meredith.com/browse/DQA-7065) - OneCMS | Regression | Edit Creative Work - Web Series **NEW**

Edit Creative Work Page

1. Edit creative work page -  [DQA-7048](https://jira.meredith.com/browse/DQA-7048) - OneCMS | Regression | Edit Creative Work Page **NEW**

Delete Creative Work

1. Delete creative work -  [DQA-7051](https://jira.meredith.com/browse/DQA-7051) - OneCMS | Regression | Delete Creative Work **NEW**

### Newsletter [DQA-7150](https://jira.meredith.com/browse/DQA-7150) - OneCMS Newsletter NEW

1. Publish Newsletter with all fields [DQA-7151](https://jira.meredith.com/browse/DQA-7151) - OneCMS Newsletter | Publish Newsletter **NEW**
   1. Title
   2. Newsletter Content
2. Edit Newsletter along with editing all the fields and update it [DQA-7152](https://jira.meredith.com/browse/DQA-7152) - OneCMS Newsletter | Edit Newsletter & Update **NEW**
   1. Title
   2. Newsletter Content
3. Delete Newsletter [DQA-7153](https://jira.meredith.com/browse/DQA-7153) - OneCMS Newsletter | Delete Newsletter **NEW**