Software Development and Update Procedure

MRI Fusion Controller System Software (FusionServer)

SUBJECT:	Software Development and Update Procedure
PROJECT:	MRI Fusion Controller System Software (FusionServer)
REVISION:	DRAFT Rev J <jwa>

Printed on: 4/11/2019 2:31 PM

(See last page for revision history)

SUMMARY:

The following document details the method of developing and releasing production software for the Fusion Controller.

Two repositories will be used, one private and the other public, to develop and create releases of the production level software. Production level software will have obfuscated code and hold none of the source directories that reside within the development repository. Both repositories however must follow the same structure to the letter to minimize confusion when creating a release. That means if the dist folder resides in FusionServer, it should still be on the public side in the same place.

When at a point that the development side is a release candidate, the standard procedure listed in this document must be followed to the letter to create a release of the software on the public release repository.

GITHUB REPO OPERATION:

The GitHub repository supports the following features which are used for updating the Fusion software both by development and by the user standpoint.

- Branches
- Commits
- Tags

Currently there is only one branch of the repository, named “master”, which is where all the source code lives. When development pushes new things to the GitHub they will reside at the top of the master branch (or HEAD).

Each time something is pushed to the GitHub, a new commit ID is attached allowing one to “reset” if necessary back to that specific point in time. A commit ID is essentially a snapshot of the repo at that given point in time.

Tags essentially place a specific name to a given commit so they can be used for release tracking. When a specific version of the master source is ready for release and has been fully tested and approved, a release tag can be placed on that commit providing a snapshot of the repo for all the Fusions in the field to update to. The master source can continue to be worked on, however a new update will not be available *to the user* until a new release tag has been created on the GitHub.

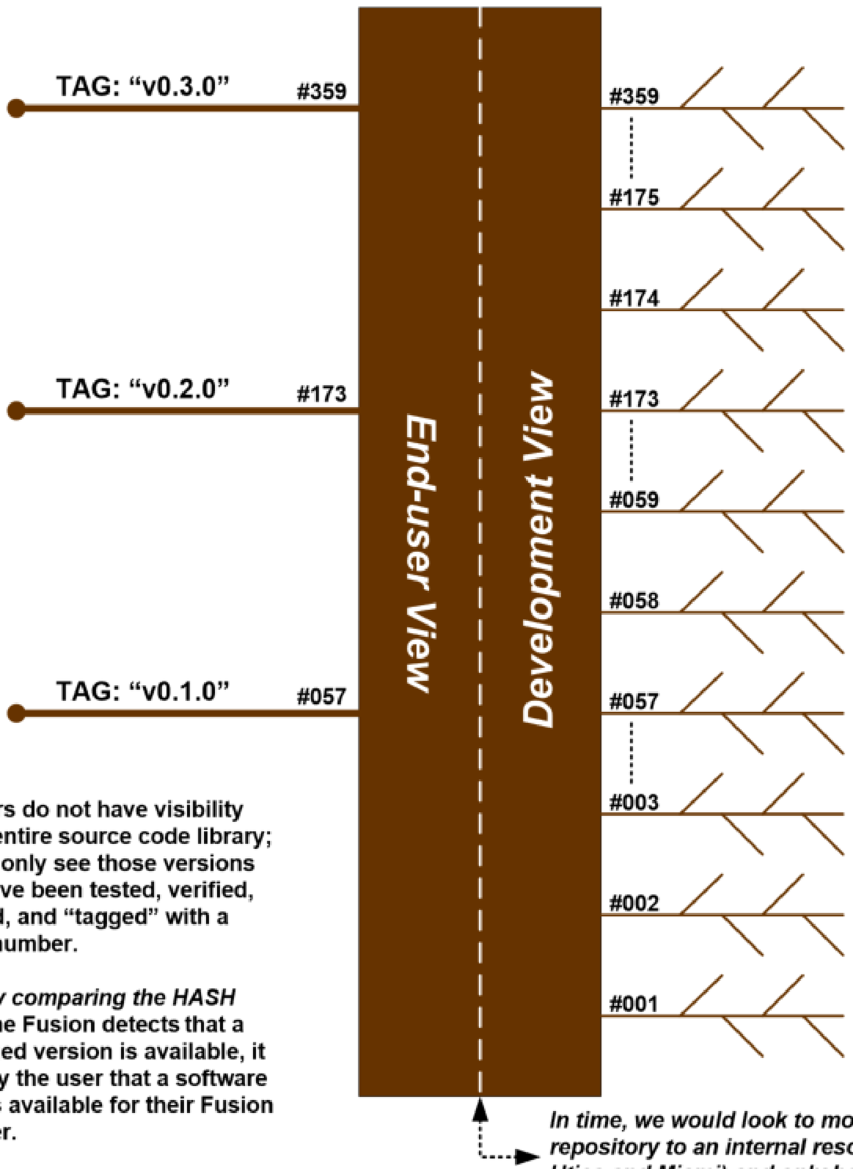
GITHUB REFERENCE:

```
`USER RELATED ===============================================================`
# Fetch brings the most recent source from the GitHub to the temp .git folder
# on the target system but does not apply it to the local directory.
git fetch

# Rev-parse pulls the SHA value of the current commit on the Fusion itself.
git rev-parse HEAD

# A git fetch must be performed first so that any new tags that have been added
# are pulled to the local repo. Using the following options with the rev-list will
# return the SHA value of the most recent tag available.
git rev-list --tags -max-count=1

# When a new tag is seen and an update is issued, the following command will reset
# the local repository to the specific commit the tag points to.
git reset --hard $(git rev-list --tags --max-count=1)

# When displaying the tag information to the user in the "hamburger" menu within
the
# Fusion GUI, the following command should be issued.
git describe --tags $(git rev-parse HEAD)

`DEVELOPMENT ONLY ===========================================================`
# Pull is similar to fetch in that it reaches out and gets the most recent
# files. However, it also performs a merge to the local repo updating all files
# that have been changed.
git pull

# Push is to be used only by development and is for adding code to the GitHub
# repo. A pull must ALWAYS be performed first so that other changes can be added
# along with else they will be lost.
git push

# Cloning a repository is used when performing development tasks as well as
# creating releases of the software to the public library.

# This is used when bringing the development repo into usr/Fusion directory and
# renames the local copy to Fusion so all code will still point to the correct
# folders.
git clone http://github.com/Modern-Robotics/Fusion-dev.git Fusion

# This brings the repo in under the name it was created.
git clone http://github.com/Modern-Robotics/Fusion.git


# The following allows your password to be saved for 8 hours or until restart.
git config --global credential.helper ‘cache --timeout 28800’


# The --orphan option on git-checkout allows the creation of a parentless branch:
git checkout [-q] [-f] [-m] --orphan <new_branch> [<start_point>]
      Per the git documentation: Create a new orphan branch, named <new_branch>,
      started from <start_point> and switch to it. The first commit made on this
```

```
new branch will have no parents and it will be the root of a new history
totally disconnected from all the other branches and commits.

The index and the working tree are adjusted as if you had previously run
"git checkout <start_point>". This allows you to start a new history that
records a set of paths similar to <start_point> by easily running "git
commit -a" to make the root commit.

This can be useful when you want to publish the tree from a commit without
exposing its full history. You might want to do this to publish an open
source branch of a project whose current tree is "clean", but whose full
history contains proprietary or otherwise encumbered bits of code.

If you want to start a disconnected history that records a set of paths that
is totally different from the one of <start_point>, then you should clear
the index and the working tree right after creating the orphan branch by
running "git rm -rf ." from the top level of the working tree. Afterwards
you will be ready to prepare your new files, repopulating the working tree,
by copying them from elsewhere, extracting a tarball, etc.
```

Fusion Software Source-Code Repository

Currently hosted in the cloud using GitHub



Developers can access any source code revision in the file repository by using the unique HASH code (#xxx) generated when the files were committed.

The most current commit can be referenced using the keyword HEAD.

Internally, our developers have visibility to the entire development repository and can receive notification anytime a file is revised and committed.

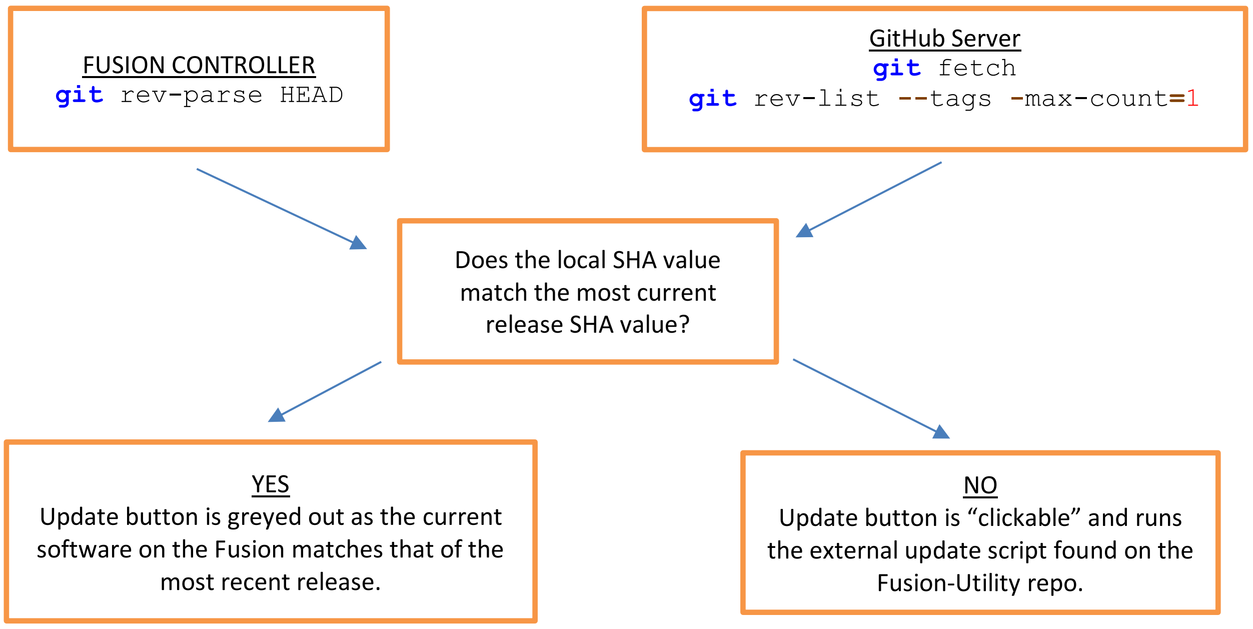
End-Users do not have visibility into the entire source code library; they can only see those versions which have been tested, verified, approved, and “tagged” with a version number.

When (*by comparing the HASH codes*) the Fusion detects that a new tagged version is available, it can notify the user that a software update is available for their Fusion Controller.

In time, we would look to move the development repository to an internal resource (available to both Utica and Miami) and only have the tagged (release) versions available via the internet.

END-USER UPDATE PROCEDURE:

The user update procedure relies on a comparison of the commit number of the local software and the commit of the most recent release tag on the public Fusion repository.


FUSION CONTROLLER
git rev-parse HEAD
GitHub Server
git fetch
git rev-list --tags -max-count=1
Does the local SHA value match the most current release SHA value?
YES
Update button is greyed out as the current software on the Fusion matches that of the most recent release.
NO
Update button is "clickable" and runs the external update script found on the Fusion-Utility repo.

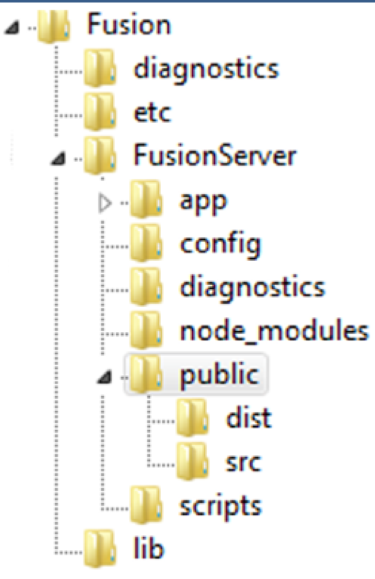
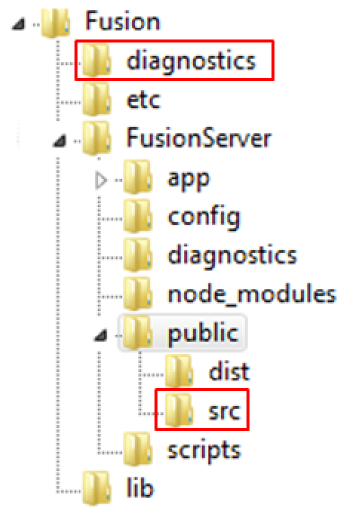
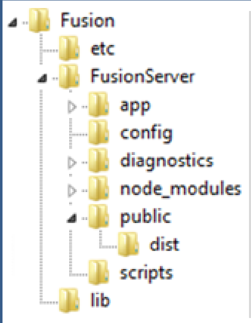
When the update script is executed, the following commands will be issued to ensure no changes that may have been made by the end-user in any way will throw off the local repo and not allow the reset to the most recent release to occur.

```
# Get the most recent source from the GitHub with all available tags
sudo git fetch

# Reads what the most recent release tag is and resets the local repository
# to that point in time.
sudo git reset --hard $(sudo git rev-list --tags --max-count=1)
```

DEVELOPMENT PROCEDURE OVERVIEW:

When developing new source on the Fusion controller, engineering will utilize the current GitHub pull/push scheme for updating the Fusion-dev development master source, leaving the public repository alone until a release candidate is prepared.

Note: When developing, the commit number on the Fusion will differ from the commit of the most recent release tag on the production release repository. The update button should now be accessible from within the GUI and pressing the button should revert or bring the Fusion back to the most recent release tag. Even though the update button is not to be used for development, it is good practice to have it enabled when the commit number differs so that if a device commit is off in any way, it can be brought back to the most current release.

Fusion-dev (Private)		Fusion (Public)
master	release	master
 Fusion diagnostics etc FusionServer app config diagnostics node_modules public dist src scripts lib	 Fusion diagnostics etc FusionServer app config diagnostics node_modules public dist src scripts lib	 Fusion etc FusionServer app config diagnostics node_modules public dist scripts lib Pre-release Commit v0.4.0 -1 –abcdef v0.4.0 V0.3.1
Master source resides on the private Fusion-dev repo and all development work and commits will be performed here.	When a release candidate is ready, a new branch will be created locally and all source content deleted. This branch should remain local, be created each time a release is proposed and never be pushed to the Fusion-dev repo.	Once the source files are removed the branch is then pushed and merged with the master of the Fusion production Github. An approved OS image will be made with this proposed release and all testing will be performed on that image. The end-user will not see this proposed release until it has been fully tested and a release Tag has been created at which time their Fusions will auto detect a new version.

Some Useful Linux Incantations to Aid in Development:

Here are a few command recommendations to make developing on the Fusion a little easier:

```
# Gives super user access until logging out
sudo su


# Enable login as the root user via ssh
nano /etc/ssh/sshd_config
    # Change PermitRootLogin without-password to the following
    PermitRootLogin yes
    # Save the file

# Set a new root password
passwd root
    # New password: admin
    # Verify: admin

# Restart the ssh service
service ssh restart


# Enable colorization of the command prompt and ls output for the root user
    # Primarily, this involves editing the /root/.bashrc file to enable and
    # customized the command prompt:
```

```
# ~/.bashrc: executed by bash(1) for non-login shells.

# Note: PS1 and umask are already set in /etc/profile. You should not
# need this unless you want different defaults for root.
PS1='${debian_chroot:+($debian_chroot)}\h:\w\$ '
# umask 022

# You may uncomment the following lines if you want `ls' to be colorized:
export LS_OPTIONS='--color=auto'
eval "`dircolors`"
alias ls='ls $LS_OPTIONS'
alias ll='ls $LS_OPTIONS -l'
alias l='ls $LS_OPTIONS -lA'
#
# Some more alias to avoid making mistakes:
# alias rm='rm -i'
# alias cp='cp -i'
# alias mv='mv -i'
```

(Note: A very nice colorized command prompt generator can be found at: http://www.ezprompt.net)

DEVELOPMENT, BUILD, AND RELEASE PROCEDURE:

***NB**: The bulk of the Revision J changes begin at this point in the document. The preceding text may not match perfectly with detailed the procedure that follows. In any case of discrepancy, the procedure that follows takes precedence and is the correct methodology.*

CLONING THE FUSION-DEV REPOSITORY:

Accessing the proper/latest source code base is the first step. The code is available via the upstream repository (currently github). Since, during operation, the code will be running from the `/usr/Fusion` directory, it is suggested that the development branch be placed on the development target into `/usr/Fusion` so that an environment more like the actual run-time deployment is used for development.

It is recommended that you create a new branch from the current master to use while you are doing your development work. It can be merged back in with the master when you are ready.

```
sudo su                    # Gives super user access (root)
cd /usr                    # Change to the /usr directory
rm -r Fusion               # Removes the Fusion directory

# Perform a git clone to bring your branch of the Fusion-dev repository into /usr
under the name Fusion

git clone http://github.com/Modern-Robotics/Fusion-dev.git --branch <YourBranchName> Fusion
```

MAKING UPDATES TO THE FUSION-DEV REPOSITORY:

Development practices are beyond the scope of this document other than to recommend that you keep other developers aware of your efforts and make sure that, if multiple developers are working on the same branch, you keep your collective work in-sync with each other using `git-pull`, `git-add`, `git-commit`, and `git-push` as appropriate:

```
# Perform a git pull to bring in any work that might have been pushed
# by other developers. This should be done regularly as to avoid conflicts.
git pull

# Perform another git pull to be sure all other files from other developers are in
# your local repo and push your changes.
git pull                                # Pulls updates
git add --all                           # Adds all untracked files
git commit -m "New commit description"  # Give the new commit a description
git push -u origin                      # Push your local repo to the GitHub
```

VERSIONING AND LABELING:

To achieve some level of consistency, the following guidelines should be used in labeling revisions and naming repository branches and tags

IMPORTANT: Please be sure to update the following:

- ✓ *The Release Notes in* ***~/Fusion-dev/README.md***
- ✓ *The Version and Build information in* ***~/Fusion-dev/version.txt***
- ✓ *The Version/Build info in* ***~/Fusion-dev/update.sh***
- ✓ *The Fusion Library Version Number in* ***~/Fusion-dev/lib/python/src*** *(if necessary)*

This should be done ***before*** *committing your final changes and pushing them to the upstream repository in preparation for a release build.*

Changes to these files during the actual build process are relative to a temporary 'orphan' branch that will be lost following the build. Since these values identify the build version, they should be part of the master repository at the same version as the code being built.

NOMENCLATURE:

~/Fusion-dev/README.md

Remember that this file is publically visible. Do not put anything in this file that is not suitable public knowledge. The version information in this file generally shows the product name, the version, and the target release date.:

```
Fusion Server version v1.2.0, 12 December 2018
```

~/Fusion-dev/version.txt

This file may ultimately be used to show the user the software version, but is used by the build and update processes. It should contain information that fully identifies down to the build attempt. For example, this is the FusionRT (Fusion RunTime), version v1.2.0, Release Candidate 3, build 2:

```
FusionRT v1.2.0 RC3 Build-2
```

~/Fusion-dev/update.sh

This file is used to update the software installed on a Fusion and pulls the version information directly from the version.txt file. If new files, flags, or other changes occur to the overall architecture and layout of the directory structure, this file must be updated and should be tested on production Fusions running all prior releases of software for validation.

~/Fusion-dev/lib/python/src

If the Fusion board libraries are modified, the Version information in this directory must be updated. At some point, we will make the Fusion Libraries their own separate entity and the burden of insuring this is properly updated will be placed on the library developer.

Branches in the Fusion-dev Repository

Developers can use whatever (hopefully meaningful) name they want for project development tasks, but when the code is ready to be built for testing and eventual deployment, the following format showing product name, version, and Release Candidate Status shall be used:

```
FusionServer-v1.2.0-RC3
```

Build Candidate Branches in the Fusion Repository

The branches in the Fusion Repository are currently only used temporarily during the validation process. When validation in complete, the branch will be merged to the master and a tag created to mark the release. We will use the following format for these build branches, incrementing the build each time a new version is built:

```
FRT-v1.2.0RC3b1
```

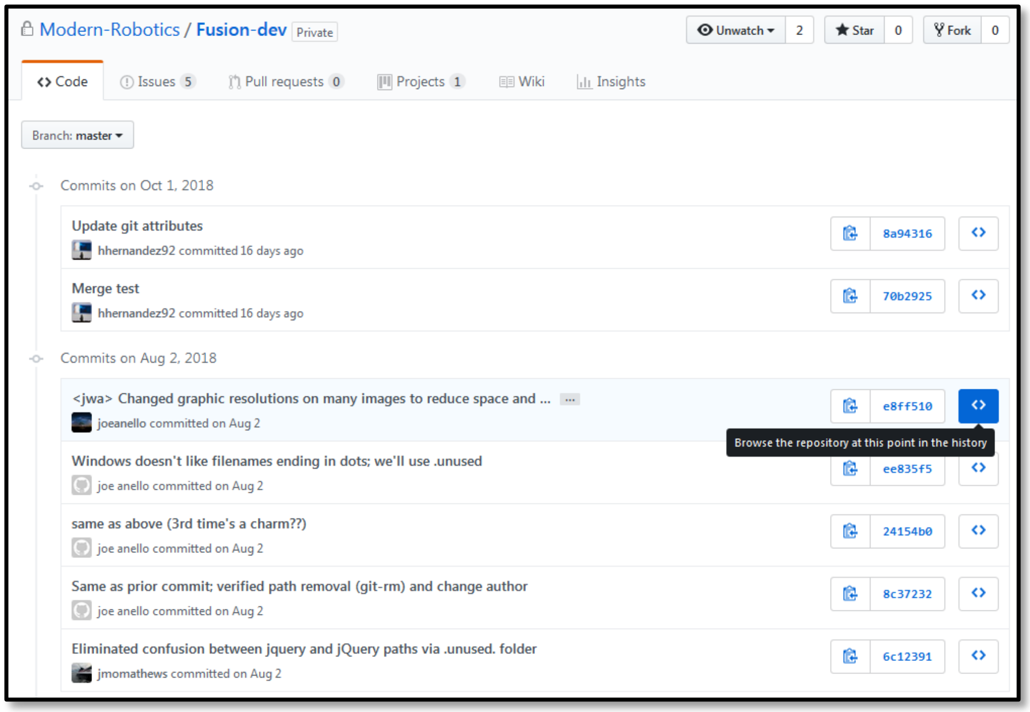
RELEASE STAGING PROCEDURE:

Once the development repository is at a point and decided to become a release candidate, the following procedure should be used for staging a release within the public repository. This staged release will not be tagged as an official release on either repository until it has been thoroughly tested via the entire test procedure outlined for release and signed off by the proper authority.

NOTE: Because this staged commit does not have a tag id, users will only be able to see it if they performed a git pull until the release tag is created. This poses no visual threat to the source code as the proposed release pushed to the public repo for pre-release in-house testing will have all source removed and should reflect the exact code that we wish tag for final release.

PREPARING FOR THE RELEASE:

On GitHub, go to the Fusion-dev repository and view the commit list for the 'master' branch. Select the commit which you want to base the release on and 'browse the repository at this point in the history' as shown:



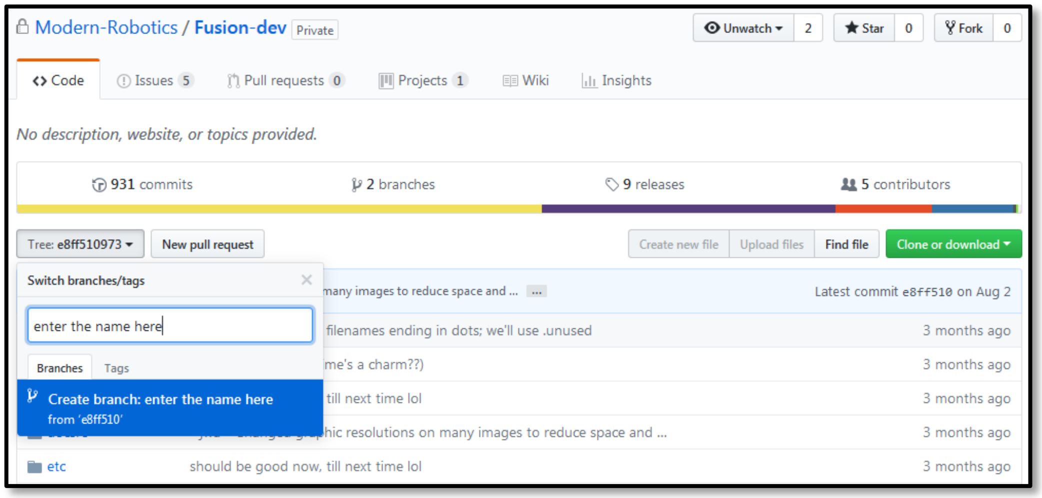
Commits on Oct 1, 2018

- Update git attributes — hhernandez92 committed 16 days ago — 8a94316
- Merge test — hhernandez92 committed 16 days ago — 70b2925

Commits on Aug 2, 2018

- <jwa> Changed graphic resolutions on many images to reduce space and ... — joeanello committed on Aug 2 — e8ff510
- Windows doesn't like filenames ending in dots; we'll use .unused — joe anello committed on Aug 2 — ee835f5
- same as above (3rd time's a charm??) — joe anello committed on Aug 2 — 24154b0
- Same as prior commit; verified path removal (git-rm) and change author — joe anello committed on Aug 2 — 8c37232
- Eliminated confusion between jquery and jQuery paths via .unused. folder — jmomathews committed on Aug 2 — 6c12391

GitHub will show the repository as requested. Select the "Tree" dropdown and create a new branch to identify the planned release. This creates a branch based on the repo at that point in history and does not include any changes, additions, or deletions which may have occurred since that point. It is recommended to use RC-vX.Y.Z in the branch name to identify the planned release.



You are now ready to use that branch to build the FusionServer for release.

MAKEFUSIONRT – AN AUTOMATED SCRIPT TO BUILD THE RELEASE:

A script has been placed in the **ref** directory of the Fusion-dev repository called makeFusionRT.sh. This is an automated build handler that runs through all of the build steps which follow. To use the script, you need to create a directory where the build will be staged, place a copy of the script from the **ref** directory in this staging directory. Then ***modify the script with the versioning information*** (see below) and execute the script.

```
# Make a new directory called staging to perform all work within
mkdir staging
cd staging
```

```
# Place a copy of the makeFusionRT.sh script into this staging directory
```

There are a variety of ways to accomplish this depending on your connection and the originating location of the script.

```
# Edit/Modify the script with the versioning information
```

There are several variables that can be set in the makeFusionRT.sh script:

```
#--------------------------------------------------------------------------------
# Please set these values before running the script since they determine what
# branches to use during the build. These names should be formatted per the
# build procedure document.
#
DEV_BRANCH="FusionServer-RC3-v1.2.0"
RT_BUILDID="FRT-v1.2.0RC3b2"

#--------------------------------------------------------------------------------
# Please set these values before running the script. These allow control over
# whether the the script should fetch either of the repos from the upstream
# server, or if they have already been cloned and to just build the code already
# in the directories.
#
# *** REMEMBER that this script will ultimately commit and push the completed
# *** builds to the upstream server.
#
GIT-DEV="no"              # yes or no -> determines if the Dev repo should be cloned
GIT-RT="yes"              # yes or no -> determines if the RT repo should be cloned
```

```
# Run the script
./makeFusionRT.sh
```

The script will provide informative output regarding progress and watch for errors which may occur during the build.

MANUAL STEP 1) GETTING THE SOURCE CODE ONTO THE BUILD SYSTEM:

Log into the build system as the superuser and create a directory to hold all of the work. In this process, we will use a unique directory name based on the current date.

```
# Make a new directory called staging to perform all work within
mkdir staging
cd staging

# Clone the dev and release repositories
git clone http://github.com/Modern-Robotics/Fusion-dev.git --branch <branchname>
git clone http://github.com/Modern-Robotics/Fusion.git

# Move into the Fusion-dev directory
cd Fusion-dev


# Create a new temporary (orphan) branch for this release candidate.
git checkout --orphan <frt-rc#>
```

MANUAL STEP 2) BUILDING THE RUN-TIME VERSION FOR RELEASE:

```
#-------------------------------------------------------------------------------
# Compile all necessary files and remove all source files at this time that are
# not intended to be in the final release

# 1) Modify .gitignore file to include Build/ and node_modules/
cd ./staging/Fusion-dev
nano .gitignore
    # ((( Remove Build/ and node_modules/ from the file )))


# 2a) Build the Documentation
# productionBuild automatically places doc files into FusionServer/Src directories
cd ./staging/Fusion-dev/docsrc
./productionBuild.sh

# 2b) Delete the Document Source folder and the build script
cd ./staging/Fusion-dev
rm -r docsrc


# 3a) Build the Fusion Server
# The automated serverBuild.sh script runs the entire build process
cd ./staging/Fusion-dev/FusionServer
./serverBuild.sh

# 3b) Delete the Source folder and the build script
rm -r Src
rm –r serverBuild.sh
```

```
# 4a) Build the Diagnostic Utiltiy
# The automated compile.sh script runs the entire process
cd ./staging/Fusion-dev/diagnostics
./compile.sh

# 4b) Retain the following files and folders
    diagnosticGui.py
    psutil/
    res/
    runRemi.sh
    src.so
```

```
rm -r  compile.sh
       README.md
       rpi-benchmark.sh
       setup.py
       src.py
```

```
# 5a) Build the data Logging Utility
# The automated compile.sh script runs the entire process
cd ./staging/Fusion-dev/logging
./compile.sh

# 5b) Retain the following files and folders
    dataLogger.py
    res/
    runRemi.sh
    src.so
```

```
rm -r  compile.sh
       setup.py
       src.py
```

```
# 6) Build procedure for Libraries
# The automated compile.sh script runs the entire process except for
#   a few housekeeping issues...

# 6a) Clean out any old compiled libs
cd ./staging/Fusion-dev/lib
rm -r Fusion*

# 6b) Run the build process
cd ./staging/Fusion-dev/lib/python
./compile.sh
    # The final production .tar.gz is placed in the parent directory
    # (ie:./staging/Fusion-dev/lib)

# 6c) Move up to the parent directory and get rid of the source files
cd ..
# Retain the following files and folders
    Fusion-x.x.x.tar.gz
    *.deb
    *.tar.gz
    noVNC/
```

```
rm -r  c++
       python
       README.md
```

```
# 7) Final cleanup of the main Fusion-dev directory
cd ./staging/Fusion-dev

# Retain the following files in the main directory
```

```
diagnostics/
etc/                   rm -r  docsrc
FusionServer/                 ref
.git*                         temp
lib/                          .unused
logging/
README.md        # This file should have been updated earlier with notes
update.sh
```

Manual Step 3) Moving the Built Version to the Repositories:

The current 'lay of the land' is this:

- The **staging/Fusion-dev** directory contains the "FusionServer Binary". This is the built and obfuscated version of the Fusion Server code. All source modules have been removed.
- The **staging/Fusion** directory is currently unchanged from what was cloned earlier. It is the currently released FusionServer Binary.
- Remember that, at this point, although the files in the various sub-directories of **Fusion-dev** have been modified, the local repository (aka: **Fusion-dev/.git**) still contains the clone of the original branch we are building to release.

```
#-------------------------------------------------------------------------------
# 1) Add and commit all the new changes in the branch
git add --all
git commit -m "Proposed release from <commit_ID>"
```

This adds all of the new files we just generated, moved, copied, etc in the working directories to the files that git is keeping track of, then commits them to the local repository.

The automated script uses the following to build the <commit_ID>:
Commit_ID="Release Candidate: $(cat version.txt) [Dev: $<branchname> RT: $<frt-rc#>]"

```
# 2) Add the public release repository as a remote repo
git remote add FRT-Binary http://github.com/Modern-Robotics/Fusion.git
```

This links the public release repository **Fusion.git** to our current FusionServer Runtime (binary) local repository, calling that link **FRT-Binary**.

```
# 3) Fetch all info from the remote repository and push the new branch to the remote.
#    >>> Do not push anything to the source origin, keep all this local as this folder
#        is deleted in future steps keeping any branches and changes from reaching the
#        online source repository.
git fetch FRT-Binary
git push FRT-Binary <frt-rc#>
```

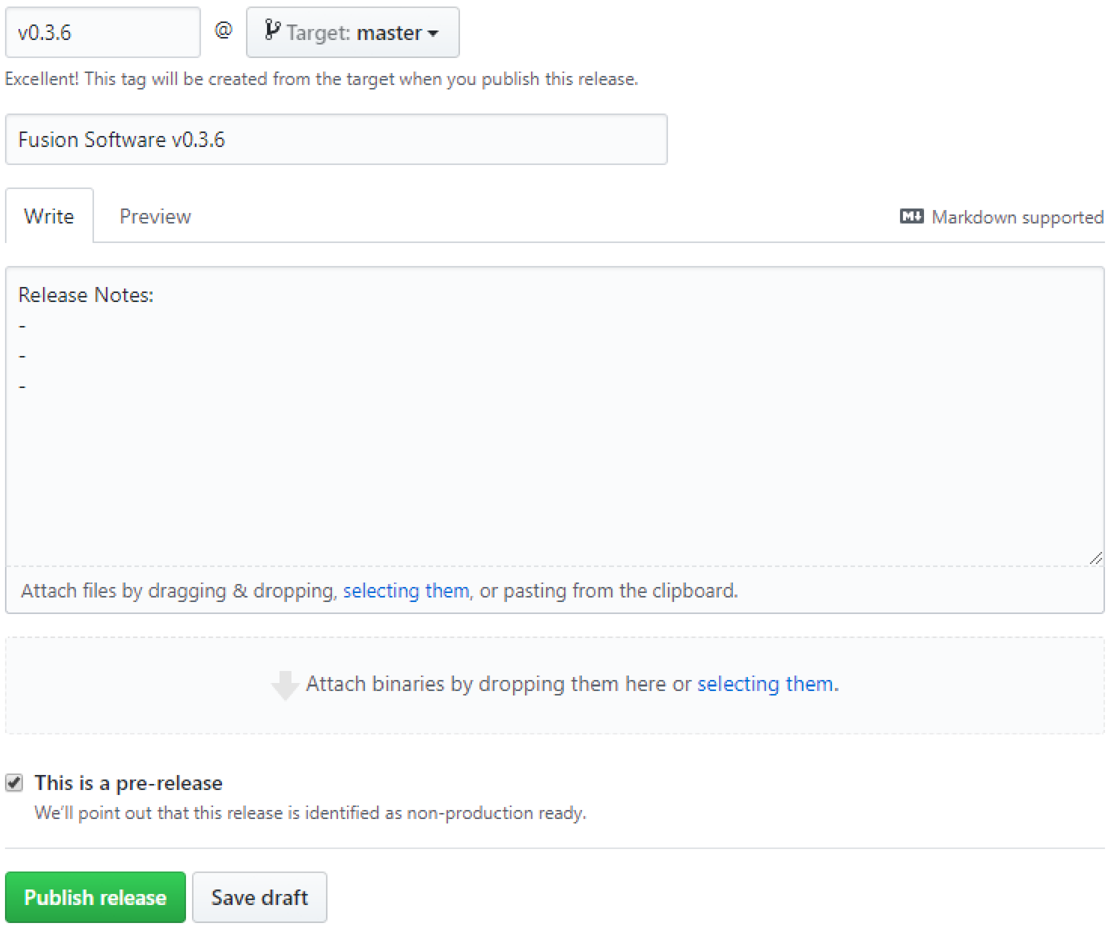
Manual Step 4) Pulling the Newly Built Runtime into the Fusion Directory:

```
# cd into the Release folder and perform a git pull to bring the new branch
# down to the local.
cd ../Fusion
git pull


# Checkout the release branch that was recently pushed from the source and
# perform a merge from master. "-s ours" takes the merge strategy that all files
# added and removed take precedence over the master.
git checkout <frt-rc#>
git merge -s ours master

# Checkout the master and merge with the release. This overwrites the master with
# an exact copy of what the branch is at.
git checkout master
git merge <frt-rc#>

# Push the new master to the github repo. This will place the master two commits
# past the previous tag meaning people will still be updating to the most recent
# good release until this commit is marked with a release tag.
git push -u origin

# Delete the release branch from the github online repo.
git push origin --delete <frt-rc#>


# cd back to the main directory and delete the staging folder.
cd ../..
rm -r staging/
```

CREATING A RELEASE TAG:

The following steps should be taken when creating a new release for the Fusion software. Redundancy checks will be performed by no less than two individuals and both must sign off before a release can be created.

1. A new compile of the OS should be performed with the new proposed release commit ID and all subsequent testing should be done on a burned SD with this image.
2. Proper vetting of the software should be performed by at least two individuals with both following the specified criteria listed in the release form.
3. During this testing phase, a "known bug list" will be generated by both individuals and reviewed at the end of the phase as to whether the release is worth or certain bugs are critical before continuing.
4. Once the software has been fully tested and both have signed off, a release tag can be created. A release tag should not be created via the 'git' command but rather by logging into the MRI GitHub account and using the "Draft a new release" function. The same release tag should be created on the Fusion-dev for the commit ID the proposed release was born.
5. Fill in the required fields as seen below following the **EXACT** naming convention as described here. Be sure to check the box marked "pre-release" if the version number is below 1.0.0


v0.3.6 @ Target: master

Excellent! This tag will be created from the target when you publish this release.

Fusion Software v0.3.6

Release Notes:
-
-
-

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Attach binaries by dropping them here or selecting them.

☑ This is a pre-release
We'll point out that this release is identified as non-production ready.

6. Publish the release, perform an update via the Fusion GUI, and verify either by command line or log that the current local commit SHA matches that of the most recent release just published.

REVISION HISTORY			
DATE	**DESCRIPTION OF CHANGE**	**REVISION**	**MODIFIED BY**
3/21/18	Original Document	A	JCM
3/21/18	Fixes within staging	B	JCM
3/29/18	Add git log to build process	C	JCM
7/25/18	Update build process with specific compile details	D	JCM
7/26/18	Fix to procedure, removing build and node_modules from .gitignore	E	JCM
30-Jul-18	Updated procedure to include notes made during verification; put the list of “files to retain” in alphabetical order to minimize errors	F	JWA
17-Oct-18	Updated procedure to use Branching in the Fusion-dev repository.	G	JWA
12-Dec-18	Update formatting and process steps	H	jwa
14-Dec-18	Incorporated automated build script and naming convention for the different branches	J	jwa