Runbook

for

For NASA Maestro HUD

Version 4.0

**April 26, 2020**

Prepared by Ryan Cohan & Johnny Lockhart

Team Members

Ali Alnaqeeb

Laura Tamshen

Ryan Cohan

Matthew Elliott

Jonathan Johnson

Table of Contents

Table of Contents ii

Revision History iii

1. Overview 1

2. Build 1

2.1 File Structure 1

3. Deployment 2

3.1 Standalone Project Deployment 2

3.2 Docker Deployment 3

4. Common Tasks 4

4.1 File Management 4

5. Error Messages 4

6. Disaster Recovery/Service Level Agreement 5

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Johnny Lockhart | 04/03/2020 | Initial Version | 1.0 |
| Ryan Cohan | 04/04/2020 | Updated all sections in the document | 2.0 |
| Ali Alnaqeeb | 04/04/2020 | Formatted and update document structure | 2.1 |
| Johnny Lockhart | 04/05/2020 | Updated file structure section | 3.0 |
| Ryan Cohan | 04/23/2020 | Updated file structure section | 4.0 |

# Overview

This NASA Maestro Heads Up Display (HUD) project is a feature added to the main NASA Maestro console/GUI application. The feature application is developed, deployed, and administered as a client/server application, through the structure of the Single Page Application (SPA). The primary goal of the app is to produce a collection of procedures, roles, and tasks that is consumable by a browser-based HUD client. This run book targets the System Admin and advanced users to detailed processes to maintain and deploy the HUD application. As such, the target user should be familiar with the following concepts: JavaScript, Web Servers Systems, Git, Github, Docker, and Client-side development.

# Build

The HUD project is comprised of native JavaScript (JS) and the Node/Express framework. An Nginx Server is a comprehensive open-source server system that is utilized by the application to handle the initial delivery of client-side products, which in this package refers to a single HTML page that makes an additional request for necessary assets via the standard HTTP protocol. Also, an ExpressJS back-end system is included to handle file management and server processing tasks for linting, parsing, and serving JSON data to the HUD client.

## File Structure

The current file structure is critical to the proper deployment and execution of the HUD application, as many of the directories are hardcoded in the app. The following is a detailed listing of the necessary folders and files for system operations.

* ./Documentation/ – stores all project-related documents for long term storage
* ./backend/ – location of the back-end code and contains the files for running the ExpressJS web server
  + ./backend/assets/ - contains the files to be served by the application
    - ./backend/assets/images – image files associated with the various tasks
    - ./backend/assets/procedures - contains all procedure files. Admins should load new procedures in this folder, with the appropriate name which is displayed on the HUD
    - ./backend/assets/tasks - contains all tasks files. Admins should load new tasks in this folder, with the appropriate name as they appear in the respective procedure file
  + .backend/controllers/ – contains three file controllers
    - ./filesController.js – reads the procedures directory and returns an array of all filenames. Also, it performs linting on procedures YAML files to ensure they are properly formatted.
    - ./rolesController.js – when given a procedure filename, reads the related procedure file, parses the containing roles, and returns an array of strings.
    - ./taskController.js - when given a procedure filename and role, reads the related procedure file, parses the containing tasks, and returns an array of task objects.
  + ./backend/routes/ – contains one file, index.js to specify to the ExpressJS server how handle API calls
  + ./backend/test/ - contains unit tests that are ran via the $ npm run test command
    - ./backend/test/filesController.spec.js – unit tests for the filesController
    - ./backend/test/rolesController.spec.js – unit tests for the rolesController
    - ./backend/test/tasksController.spec.js – unit tests for the tasksController
  + ./backend/app.js – starts and configures the ExpressJS server
* ./frontend/ - location of the frontend code
  + ./frontend/index.html – the single page application main file
  + ./frontend/js/ - contains scripts file to support
    - ./frontend/js/anycontrol.js – vendor file for voice control features
    - ./ frontend/js/bootstrap.bundle.js – vendor file for bootstrap support
    - ./ frontend/js/jquery.js – vendor file for jQuery support
    - ./ frontend/js/mainWindow.js – main application logic and controller for view changes
  + ./frontend/css/ - contains vendor bootstrap designs CSS file
  + ./frontend/test/ - contains the unit tests that are ran via the $ npm run test command
    - ./frontend/test/ - unit tests for the mainWindow class

# Deployment

There are two options for the deployment of the HUD application, either as a standalone project or a Docker container application.

## Standalone Project Deployment

As a standalone project, admin personnel should copy the entire application’s project folder (except for Documentation and project-docs) to any platform that has and supports NodeJS (v12). Refer to the nodejs.org website for installation instructions. Additionally, the user-provided platform will have the ability to serve the initial HTML and related asset files via industry-standard HTTP communications. In the Docker deployment, the server of choice is the previously mentioned Nginx platform. While this free platform is both open-source and well-maintained, setup and management of the system are out-of-scope for this document.

1. On the target machine, open a terminal/command-line console in the ${project root}/backend
2. Install project dependencies, execute the command
   1. npm install
   2. node downloads all dependent packages for the ExpressJS server components
3. Start the server, run the following command
   1. npm start
   2. the server will start listening for requests on local port 3000
4. Utilizing user-provided server, navigate a browser window to ${server}/frontend/index.html
5. The application is ready for execution per instructions located in the User Guide documentation.

## Docker Deployment

A docker image of the project is automatically assembled and deployed to Docker Hub under the organization of “umgccapstonebot.” This packaging happens as a part of the code repository’s built-in Continuous Integration and Deployment. To update the latest available package on Docker Hub’s repository, initiate a pull request to the release branch of the project GitHub repository. Refer to the DevOps Guidelines document for more details. The admins should ensure a system containing an installed version of the official Docker Engine is operational to execute a docker deployment.

1. On the target machine, open a terminal/command-line console in any desired directory
2. Execute the command
   1. docker pull ${docker hub url/frontend container name}:latest (or any previous build)
   2. docker pull ${docker hub url/back-end container name}:latest (or any previous build)
   3. docker run -d -p xxxx:3000 -n “nasa.hud.backend” ${docker hub url/backend container name}:latest
   4. docker run -d -p xxxx:3001 -n “nasa.hud.frontend” ${docker hub url/frontend container name}:latest
   5. -p maps a local machine port to the docker container
   6. -d runs the system in background mode
   7. -n provides an alias to reference the container in future docker commands
3. Navigate a browser window to (server address/DNS name):3001
4. The application is ready for execution per instructions located in the User Guide

# Common Tasks

As currently structured, the HUD application is a typical client/server web application. Admins have no administrative tasks required to perform while the application is running. However, a System Admin or Developer will need to execute the following tasks as needed to ensure the system is operational per specified requirements.

## File Management

As detailed in Section 2.1, the HUD application employs several different files to discover and decompose the necessary tasks to display on the target viewport. The only mechanism for adding, changing, or removing any file to or from the system is to load the corresponding file in its pre-designated directory manually. After making file changes, the System Admin or Deployer will need to rebuild and deploy the application as specified in Sections 2 and 3.

# Error Messages

The System Admin should be aware of the following HUD system responses and their probable causes.

* /hud/lint
  + 200 – message success
  + 404 – file not found
  + 400 – request malformed
  + 422 – invalid or missing query parameter
  + 5xx – internal server error
* /hud/getfiles
  + 200 – message success
  + 400 – request malformed
  + 422 – invalid or missing query parameter
  + 5xx – internal server error
* /hud/roles
  + 200 – message success
  + 404 – file not found
  + 400 – request malformed
  + 422 – invalid or missing query parameter
  + 5xx – internal server error
* /hud/tasks
  + 200 – message success
  + 404 – file/role not found
  + 400 – request malformed
  + 422 – invalid or missing query parameter
  + 5xx – internal server error

# Disaster Recovery/Service Level Agreement

The HUD application is developed as a prototype-level application. Production-level infrastructure and access controls are out-of-scope for this development effort. While any deployed version of the application is maintained in both a pre-built Git repository and Docker image, there exist no guarantees as to the availability, accessibility, or scalability of the application beyond those offered by the respective host systems.