Software Requirements Specification

for

NASA Maestro Format Test Tool

**Version 4.0**

**Prepared by:**

**NASA TEAM 1**

**University of Maryland Global Campus**

**23 February 2020**

Table of Contents

[1.](#_heading=h.3tbugp1) Introduction 1

[1.1](#_heading=h.28h4qwu) Purpose 1

[1.2](#_heading=h.nmf14n) Intended Audience and Reading Suggestions 1

[1.3](#_heading=h.37m2jsg) Product Scope 1

[1.4](#_heading=h.1mrcu09) Definitions and Abbreviations 2

[1.5](#_heading=h.46r0co2) References 4

[2.](#_heading=h.2lwamvv) Overall Description 4

[2.1](#_heading=h.111kx3o) Product Perspective 4

[2.2](#_heading=h.3l18frh) Product Features 5

[2.3](#_heading=h.206ipza) Operating Environment 6

[2.4](#_heading=h.4k668n3) Design and Implementation Constraints 6

[2.5](#_heading=h.2zbgiuw) User Documentation in each Sprint cycle 7

[3.](#_heading=h.1egqt2p) System Features 7

[3.1](#_heading=h.3ygebqi) Check DOCX Validity 7

[3.2](#_heading=h.2dlolyb) Create Document Screenshot 8

[3.3](#_heading=h.sqyw64) Compare Image Files 8

[3.4](#_heading=h.3cqmetx) Output Results 8

[3.5](#_heading=h.1rvwp1q) Save New Format 8

[4.](#_heading=h.4bvk7pj) External Interface Requirements 9

[4.1](#_heading=h.gjzse8nblguc) User Interfaces Overview 9

[The UI for the new program extends Maestro’s command-line interface. 9](#_heading=h.gjdgxs)

[4.2](#_heading=h.o01hqa69eeqz) Use-Case Model survey 9

[4.2.1 Use-Case Reports 9](#_heading=h.n1kndgs8yyq3)

[4.2](#_heading=h.2r0uhxc) Hardware Interfaces 12

[4.3](#_heading=h.1664s55) Software Interfaces 12

**Revision History**

|  |  |  |
| --- | --- | --- |
| Date | Reason for Changes | Version |
| 02/23/2020 | Initial Document | 1.0 |
| 03/12/2020 | Response to instructor’s comments | 2.0 |
| 3/22/2020 | Updated Version | 3.0 |
| 4/25/2020 | Added Use Cases | 4.0 |

# Introduction

## Purpose

This software will be designed to verify that the .docx files, created by Maestro, are valid and compare the expected formatting of the provided YAML (A [human-readable](https://en.wikipedia.org/wiki/Human-readable) [data-serialization language](https://en.wikipedia.org/wiki/Serialization)) to a desired or expected output as a picture file. The software will maintain a record of the actual output and compare it to both the new expected and actual outputs, to maintain a record of changes in YAML formatting.

## Intended Audience and Reading Suggestions

This document is intended for members of the development team, to confirm requirements and basic architecture. Additionally, the document is meant for the customer to confirm requirements and initial specifications before serious development.

## Product Scope

This product will be developed to test NASA’s Maestro software and provide additional capability following the completion of document conversion. This includes the ability of the program to discover and highlight changes in the formatting and to display those changes made by Maestro developers.

## Definitions and Abbreviations

There are several terms used in this SRS relating to the software being designed. The related terms are:

* **.NET Core**: .NET Core is an open-source, general-purpose development platform maintained by Microsoft and the .NET community on GitHub. The cross-platform can be used to build the device, cloud, and IT applications.
* **API**: A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or another service.
* **CICD**: Refers to the combined practices of continuous integration and either continuous delivery or continuous deployment.
* **DOCX**: A DOCX file is a document created by Microsoft Word or another [word processing](https://techterms.com/definition/wordprocessor) program, such as OpenOffice Writer or Apple Pages. It contains formatted text but may also include images, drawn objects, and other document elements.
* **Express.js**: is a minimal and flexible Node.*js* web application framework that provides a robust set of features for web and mobile applications.
* **Gitlab**: is a web-based DevOps lifecycle tool that provides a Git-repository manager providing wiki, issue-tracking, and CI/CD pipeline features, using an open-source license, developed by GitLab Inc.
* **GNU**: An operating system that is [free software](https://www.gnu.org/philosophy/free-sw.html). The GNU operating system consists of GNU packages (programs specifically released by the GNU Project) as well as free software released by third parties.
* **HTML**: Hypertext Markup Language.
* **MAESTRO Software**: An open-source program released by [NASA](https://en.wikipedia.org/wiki/NASA) to facilitate the creation of EVA and IVA procedures.
* **NASA**: National Aeronautics and Space Administration.
* **Node.js**: Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser.
* **PDF**: The Portable Document Format (PDF) (redundantly: PDF format) is a file format developed by Adobe in the 1990s to present documents, including text formatting and images, in a manner independent of application software, hardware, and operating systems.
* **PNG**: A raster-graphics file-format that supports lossless data compression. PNG was developed as an improved, non-patented replacement for Graphics Interchange Format (GIF).
* **REST**: Representational State Transfer. It describes an architecture that is used for web APIs for data communication. It also supports some of the common HTTP methods to make the interaction between the machines or applications.
* **Runkit**: Runkit notebooks are interactive JavaScript playgrounds connected to a complete node environment right in your browser. Every npm module is pre-installed.
* **UI**: User Interface
* **Vector**: A point that has a definite position on the x- and y-axis of the work plane and determines the direction of the path; further, each path may have various properties including values for stroke color, shape, curve, thickness, and fill.
* **XML**: Extensible Markup Language (XML) is a [markup language](https://en.wikipedia.org/wiki/Markup_language) that defines a set of rules for encoding [documents](https://en.wikipedia.org/wiki/Electronic_document) in a [format](https://en.wikipedia.org/wiki/File_format) that is both [human-readable](https://en.wikipedia.org/wiki/Human-readable_medium) and [machine-readable](https://en.wikipedia.org/wiki/Machine-readable_data).
* **YAML**: A [human-readable](https://en.wikipedia.org/wiki/Human-readable) [data-serialization language](https://en.wikipedia.org/wiki/Serialization). It is commonly used for [configuration files](https://en.wikipedia.org/wiki/Configuration_file) and in applications where data is being stored or transmitted.

## References

* Sample EVA procedure available at <https://www.nasa.gov/centers/johnson/pdf/539922main_EVA_134_F_A.pdf>.
* The current NASA Maestro source code is available at <https://github.com/xOPERATIONS/maestro>.
* Kanban link at <https://github.com/xOPERATIONS/maestro/projects/2>.
* STS-134 EVA YAML example is available at <https://gitlab.com/xOPERATIONS/sts-134>.
* Gitlab, <https://about.gitlab.com/>.
* Runkit, <https://github.com/runkitdev>.
* Node.js, https://nodejs.org/en/
* Express.js, <https://expressjs.com/>
* .NET Core, https://docs.microsoft.com/en-us/dotnet/core/

# Overall Description

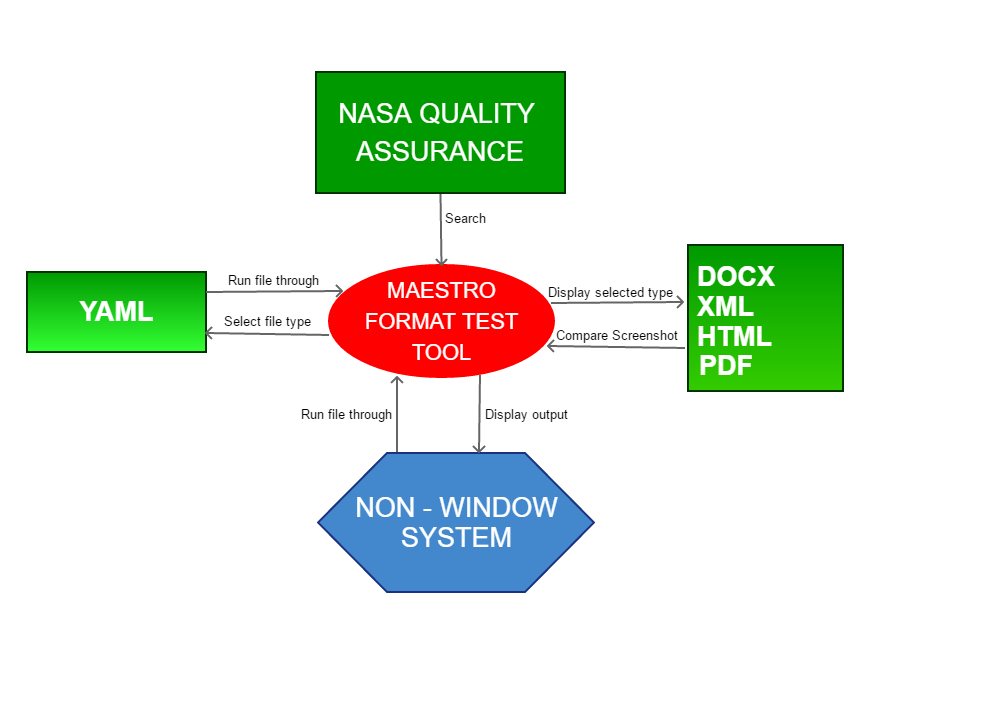
## Product Perspective

This product is a tool for the existing NASA Maestro software. This product adds a way to test and demonstrate any changes in formatting between expected Maestro outputs and actual outputs over time. Additionally, NASA is requesting test XML production by Maestro, move the repository and new software to Gitlab, add testing of Maestro through Runkit, and add the ability to track API changes through different versions.

## Product Features

This product will test Maestro’s ability to output the desired .docx format. The program will test whether the created file is a valid Microsoft Word document and shall be able to run on a non-windows system and without the use of Microsoft Word. Additionally, it will make the document into a PDF and then an image file (PNG or SVG) to allow pixel comparison with a previously generated expected outcome.

Additionally, XML creation will be tested in the Maestro software suite of outputs and the project will be ported to Gitlab. Furthermore, testing within Maestro will be automated using Runkit, and a system to track API changes will be added.



## Operating Environment

The front-end of the project is already built and consists of a console application written in JavaScript on Node.js. The output produced by the console app is displayed via HTTP in a browser using Angular. The HTTP server is written TypeScript using Angular. The word document testing and validating functionality shall be implemented in the form of RESTful[[1]](#footnote-1) web developed either in Node.js and Express.js or .Net Core according to customer preferences. LibreOffice will be used for the conversion of DOCX to PDF file format. Both will be contained in a Docker container running Linux as the operating system to ensure compatibility with different systems without changes in testing results.

## Design and Implementation Constraints

In this section, the constraints of this system are discussed below:

* The software needs to be system independent specifically designed to run without the use of Microsoft Windows or Microsoft Word. The goal is to be platform agnostic.
* One challenge created by the potential use of Linux is the requirement to support the Arial typeface which is not provided within GNU Linux. The use of created container images should alleviate this concern.
* The software testing suite (currently Mocha) needs to instantiate the Docker container providing the platform-agnostic document testing and formatting (explained below). Preliminary research indicates that this is possible with Node.js and commander.js but additional research is needed.

## User Documentation in each Sprint cycle

In each Sprint cycle, documentation required for adding into the NASA maestro software is discussed in this section.

* Any improvement to the software shall be documented. The improvement shall facilitate the addition of XML and provide a tutorial software for merging within the existing codebase.
* Developers shall document any changes to the current version software.
* Developers shall create web API specifications adding to those existing APIs and software with links to further documentation.

# System Features

The system will have five features that will be integrated into the existing Maestro software to enable the comparison of changes in Maestro’s formatting output:

## Check DOCX Validity

CD1: The system shall use LibreOffice to ensure that the Microsoft word document is valid.

CD2: LibreOffice shall run in a Docker container launched by a Node.js script to handle the Maestro created document.

## Create Document Screenshot

CS1: The system shall convert the PDF document created by Maestro into a PNG picture file.

CS2: LibreOffice shall run in a Docker container launched by a Node.js script to convert the document to an image file.

## Compare Image Files

CF1: The system shall compare the created image file stemming from the original Maestro DOCX document to a user-supplied image that represents the user's expected output.

CF2: The system shall compare the original image file to the last previous image file created by the program. This previous image is saved locally. If there is not a previous image, then this step is skipped.

## Output Results

OR1: The system shall export the results in PDF or PNG file format based on the user’s selection.

OR2: The system shall estimate the percentage change between the three images.

OR3: The system shall export three HTML pages that highlight the changes between the three images.

## Save New Format

SF1: The system shall use local memory to save the results of the test for comparison in future tests.

SF2: The system shall save the most recent tests.

# External Interface Requirements

This section details the requirements for the added user interface within the current Maestro software. The user interface, hardware interface, and software interface are detailed:

## User Interfaces Overview

## The UI for the new program extends Maestro’s command-line interface.

## Use-Case Model survey

Twelve Use Cases make up the requirements for this system. Below is a summary.

|  |  |
| --- | --- |
| Use Case | Description |
| Select Document | The actor selects which document to run in Maestro. |
| Select Which Actors Application Steps to Display. | Since there may be several applications per Actors, The Actors select which application to display the document. |
| Select Docx Application Step to Display | The actor selects the DOCX application to display the document. |
| Select PDF Application Step to Display | The actor selects the pdf application to display the document. |
| Select XML Application Step to Display | The actor selects the XML application to display the document. |
| Step has Instructions | A step may contain instructions with Maestro Software. |
| Instruction exists for entire document used in application | The entire document may have instructions associated with the application. |
| Instructions exist for a set of application steps | The Instruction may exist for the set of applications within the document. |

### Use-Case Reports

#### Use Case Name: Select Document

**Basic course of events (Scenario):**

|  |  |
| --- | --- |
| **Actor** | **System** |
| The actor selects the YAML document. | The system selects the YAML and displays the Select document. |

#### Use Case Name: Select Which Actors Application Steps to Display.

**Basic course of events (Scenario):**

|  |  |
| --- | --- |
| **Actor** | **System** |
| The actor selects DOCX to display the YAML document | The system displays the YAML document as DOCX. |

#### Use Case Name: Select DOCX

**Basic course of events (Scenario):**

|  |  |
| --- | --- |
| **Actor** | **System** |
| The actor selects DOCX to display the YAML document | The system displays the YAML document as DOCX. |

#### Use Case Name: Select PDF

**Basic course of events (Scenario):**

|  |  |
| --- | --- |
| **Actor** | **System** |
| The actor selects PDF to display the YAML document | The system displays the YAML document as pdf |

#### Use Case Name: Select XML

**Basic course of events (Scenario):**

|  |  |
| --- | --- |
| **Actor** | **System** |
| The actor selects XML to display the YAML document | The system displays the YAML document as XML |

#### Use Case Name: Step has Instructions

**Basic course of events (Scenario):**

|  |  |
| --- | --- |
| **Actor** | **System** |
| The actor selects a step within the application. | The system displays the steps to view the documents. |

#### Use Case Name: Instructions exists for a set of application steps

**Basic course of events (Scenario):**

|  |  |
| --- | --- |
| **Actor** | **System** |
| The actor views a step of the application that has one or more documents. | The system displays the associated selected application. If there is more than one document for the step, the system allows the actor to see the document in all applications. |

#### Use Case Name: Instruction exists for entire document used in application

**Basic course of events (Scenario):**

|  |  |
| --- | --- |
| **Actor** | **System** |
| The actor views the document in the application. | The system displays steps on viewing the document within the application. |

## Hardware Interfaces

HI1: This system shall be deployed in Windows, Linux, or Mac OS.

HI2: The program shall run locally or through a hosted docker container.

HI3: The system shall store all data locally.

## Software Interfaces

SI1: The system shall use Electron to display the program's UI and Docker Containers to manage most backend development.

SI2: The system shall employ LibreOffice to facilitate the testing of DOCX files and conversion to PDF.

SI3: The system shall employ OpenCV to convert the PDF to an image file and verify the image file.

SI4: The system shall compare different versions of image files.

SI5: The system shall deploy JavaScript to handle the image comparison locally.

1. Representational State Transfer: a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations. Other kinds of Web services, such as SOAP Web services, expose their own arbitrary sets of operations. [↑](#footnote-ref-1)