Work Experience Report

NASA John C. Stennis Space Center (SSC)

by

Daniel Guo

(281) 919-9260

dg32366@tamu.edu

Department of Mechanical Engineering

Texas A&M Engineering

ENGR 385

Senior

First Co-op Work Term

Fall 2017

Approved by: Fernando Figueroa, Ph.D.

Fernando Figueroa

Lead Autonomous Systems and Operations

John C. Stennis Space Center, Mississippi

Date Submitted: December 8, 2017

Abstract

The NASA Platform for Autonomous Systems (NPAS) toolkit is currently being used at the NASA John C. Stennis Space Center (SSC) to develop the INSIGHT program, which will autonomously monitor and control the Nitrogen System of the High Pressure Gas Facility (HPGF) on site. The INSIGHT program is in need of generic timing capabilities in order to perform timing based actions such as pump usage timing and sequence step timing. The purpose of this project was to develop a timing module that could fulfill these requirements and be adaptable for expanded use in the future. The code was written in Gensym G2 software platform, the same as INSIGHT, and was written generically to ensure compatibility with any G2 program. Currently, the module has two timing capabilities, a stopwatch function and a countdown function. Although the module has gone through some functionality testing, actual integration of the module into NPAS and the INSIGHT program is contingent on the module passing later checks.

Table of Contents

List of Figures

Introduction

The High Pressure Gas Facility (HPGF) at the NASA John C. Stennis Space Center (SSC) provides high pressure air, helium, nitrogen, and hydrogen to facilities around the site. The success of all engine tests on site hinges directly on the uninterrupted delivery of these gases from the HPGF. Currently, many decisions and commands to the HPGF system are done manually, which can cause personnel to be forced to come in during off hours, weekends, and holidays. INSIGHT is a program under development by the Autonomous Systems Laboratory at Stennis Space Center that will be able to autonomously monitor and control the Nitrogen System at the HPGF, and utilizes the Nasa Platform for Autonomous Systems (NPAS) [1]. NPAS itself uses integrated system health management (ISHM)[2] concepts to operate. In order to more accurately determine system health, usage monitoring of pumps and valves is a necessity, as failures can cascade quickly; Wear or maintenance issues need to be addressed before they become an issue. In autonomous control, proper timing of the steps in a sequence is also crucial as there are events that must be finished before another action can begin. My project focused on the creation of a timing module to address these requirements.

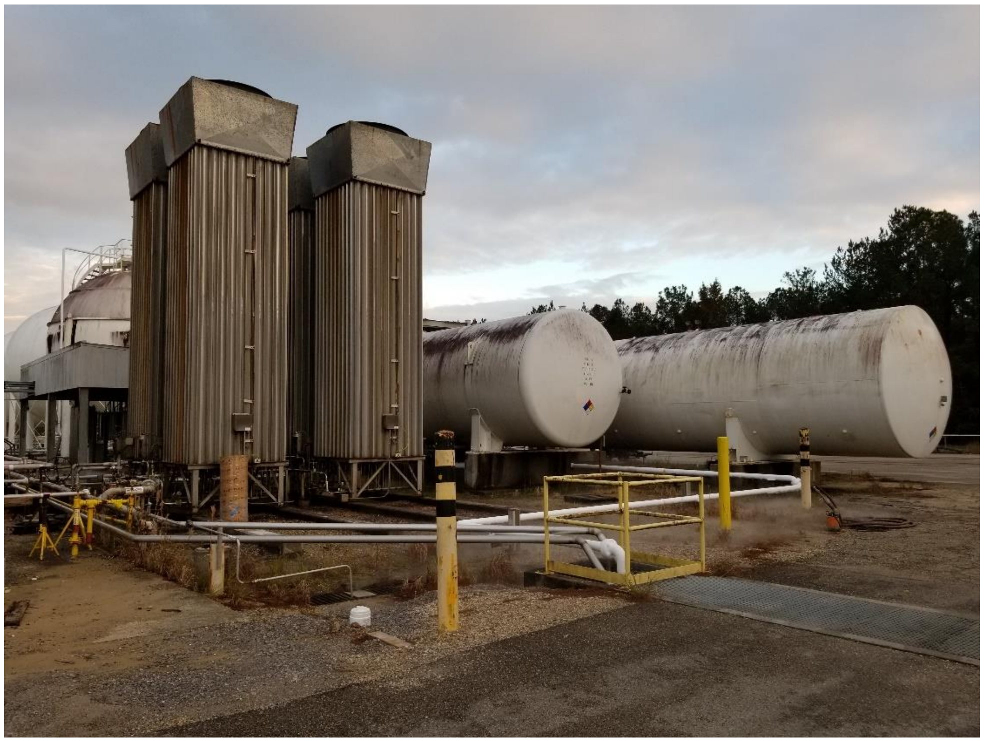


Figure 1: The High Pressure Gas Facility (HPGF) at Stennis

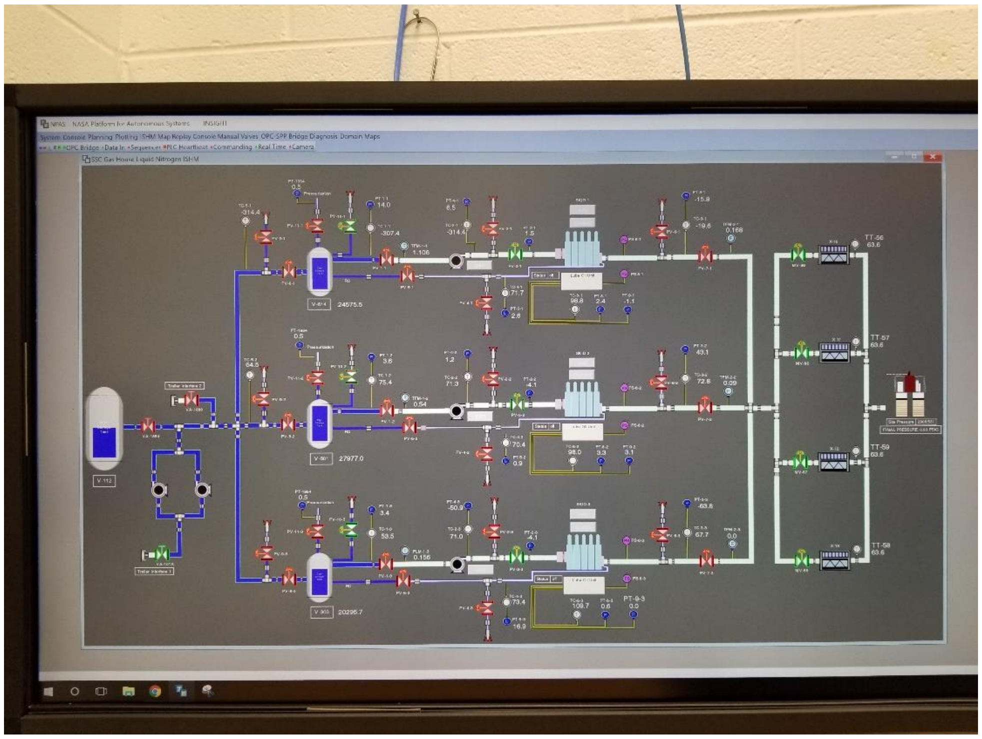


Figure 2: The INSIGHT program running in the HPGF

Overview of Jobs

The following tasks are an overview of my responsibilities:

- Create a timing module for the INSIGHT program
  - Desired functions of timing module were given at a conceptual level, actual implementation was open ended
  - Brainstorm potential ways for G2 to perform desired function, look in official documentation to see what G2 can actually do
  - Create code
  - Test code
  - Fix bugs and unintentional actions in code
- Document all necessary items relating to the timer module
  - Place comments in the code, particularly where the function of a chunk of code is not immediately obvious
  - Create a user guide that could be used by someone who is unfamiliar with coding in G2.
  - Identify limitations of the code and come up with possible future improvements

A Quick Intro to G2

Gensym G2 is a real-time expert system shell[3] with an object-oriented graphical development environment. It was developed in the 1980s by Gensym Corporation, a company based out of Burlington, Massachusetts. Intended for use in intelligent systems, it is currently used in environments such as factories, chemical plants, power systems, and satellites[4].

Applications in G2 are called knowledge bases (KB), and can be running, paused, or reset. When a KB is paused, all transient data is saved, and resuming the KB will continue the application where it left off, and any changes made during the paused time will be instantly updated, allowing for very quick code changes and almost nonexistent compiling time. Inside of KBs are workspaces, which are blank areas that items can be created, organized, and interacted with [5].

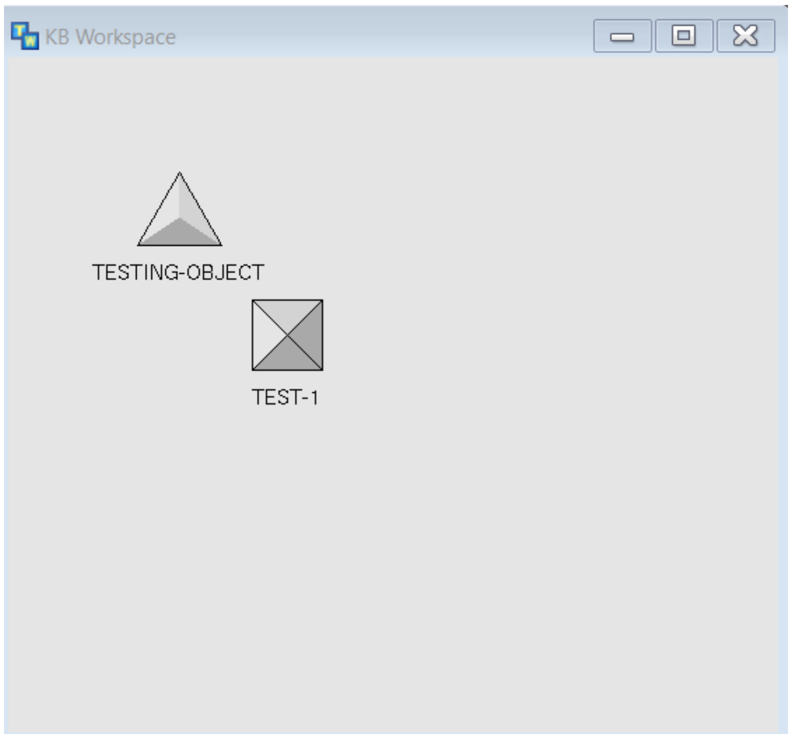
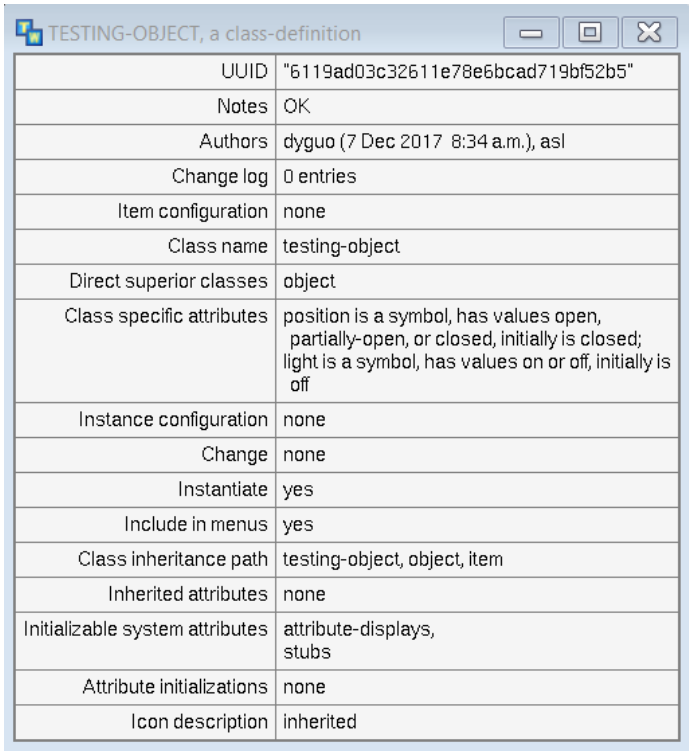


Figure 3: An example of a workspace in a KB

KBs can also be broken down in to smaller parts, called modules, with each containing a separate and distinct part of the KB. For example, a KB modeling how a car functions could have a module that has the functionality of an engine, or a module that has the functionality of an air conditioner. Individually, the modules are distinct, yet when combined, can form a larger application.

In G2, all things are either an item or a value. In a general sense, items are data structures, and values go inside of those data structures. Objects are a type of item. In the INSIGHT program, real life objects are represented as software objects, which must be defined by class definitions. Class definitions determine some behavior of the object as well as its attributes. Attributes define properties of the class. For example, a valve object could have an attribute named “position” that represents the valve’s position, with possible values “open”, “partially-open”, or “closed”.

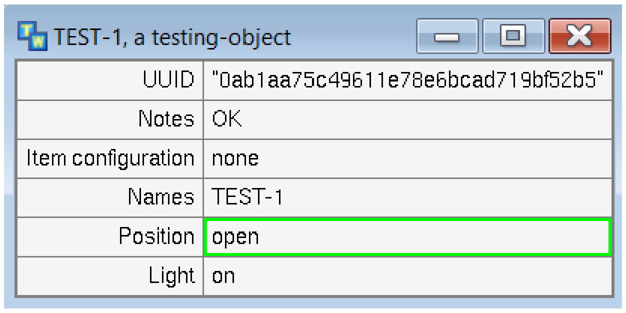
An item’s attributes can be viewed in its table, which is accessed by double clicking on the item. In addition to the attributes of an item, an item’s table also shows other properties of the item such as its name, any problems with it, and any users that have modified it. The tables for the items in the workspace from Figure 3 are shown on the next page.



TESTING-OBJECT, a class-definition

UUID	"6119ad03c32611e78e6bcad719bf52b5"
Notes	OK
Authors	dyguo (7 Dec 2017 8:34 a.m.), asl
Change log	0 entries
Item configuration	none
Class name	testing-object
Direct superior classes	object
Class specific attributes	position is a symbol, has values open, partially-open, or closed, initially is closed; light is a symbol, has values on or off, initially is off
Instance configuration	none
Change	none
Instantiate	yes
Include in menus	yes
Class inheritance path	testing-object, object, item
Inherited attributes	none
Initializable system attributes	attribute-displays, stubs
Attribute initializations	none
Icon description	inherited

Figure 4: An example of a class definition table (for a class called testing-object)



TEST-1, a testing-object

UUID	"0ab1aa75c49611e78e6bcad719bf52b5"
Notes	OK
Item configuration	none
Names	TEST-1
Position	open
Light	on

Figure 5: An example of an object table (for a testing-object named TEST-1)

A procedure is a sequence of operations that execute whenever the procedure is invoked. An object can also have methods, which are procedures that are specific to a particular object class. While most of G2 is graphical, procedures and methods are typed out just like in more common languages.

Developing the Timer Module

One part of the timing module that I needed to create was a state timer for usage timing. Factoring in the amount of time that pumps have been running or valves have been open is a significant benefit when it comes to assessing the health of a system and deciding what to do. Initially, I was to create a stopwatch, something that could count up, pause, and be reset. Later on, it became clear that what was really needed was a state timer, something that could track the attribute of another object (ex: a valve), and time how long that attribute has been a certain value (ex: "open" or "closed"). This would allow usage monitoring on pumps and valves inside the HPGF, improving the decision making capability of the INSIGHT system.

The stopwatch implementation that I ended up using was a method called "stopwatch-run" that ran continuously in the background once the stopwatch object was started, and regularly updated the attributes of the stopwatch based on a set update interval.

The biggest part of my job was to create desired functionality for the module, as well as fix any bugs or issues that came up during development. For example, one of the issues that I came across was when the stopwatch-run method was called twice for a single object, two identical methods would run. This caused extra processing power to be used, and slowed down the entire program. The way this was fixed was by creating a new attribute for the stopwatch class called timer-existing, which the stopwatch-run method checked for every time it began. If timer-existing was true, that instance of the method would stop running. If timer-existing was false, then that meant that no other instance of the method was running. The method would then change timer-existing to true and continue running. This way, only one instance of the stopwatch-run method could run at any time, even if multiple methods were started.

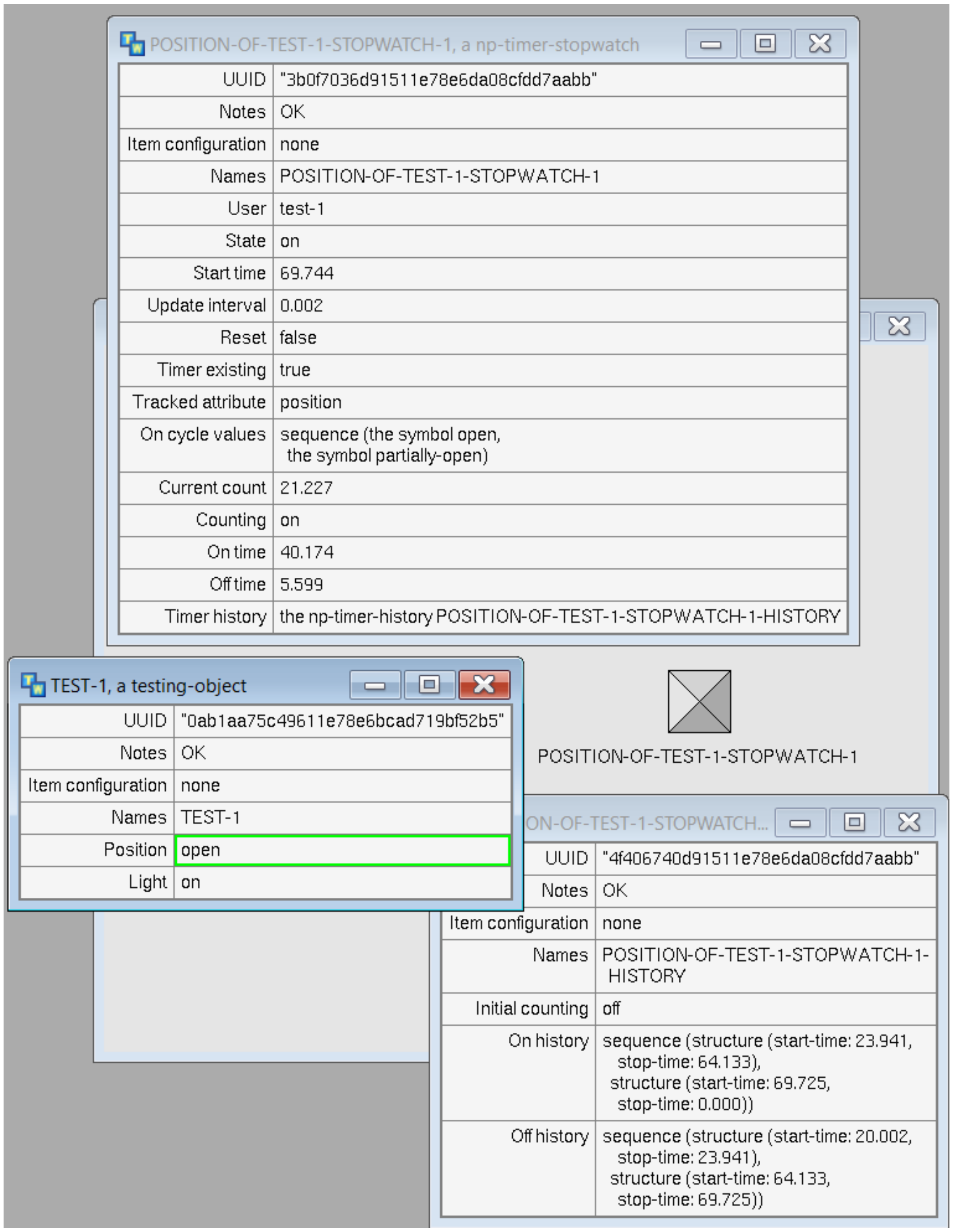
Functionality of the Timer Module - Stopwatch

From a user perspective, the stopwatch timer observes the values of an attribute of an object, and tracks how long that attribute has had a set of desired values. In order to use the stopwatch, an object must be identified that has an attribute that is to be tracked. The desired values of the attribute must also be identified. The object, tracked attribute, and desired values must then be inputted into a method and the method must be started. In the code below, the object named “test-1” has an attribute named “position”, with the desired value “open” and “partially-open”.

```
start create-stopwatch (test-1, the symbol position, sequence(the symbol open, the symbol partially-open))
```

Figure 6: Sample code that creates and starts the stopwatch

Once the method is started, a stopwatch object will be created. As seen on the next page in Figure 5, the stopwatch has an attribute called “state”. This effectively acts as an on/off switch and must be changed to on before any counting will happen. The stopwatch will now track the amount of time that the position attribute has the values “open” or “partially open”. It will treat the time that the attribute is a desired value as an on-cycle, and the time that the attribute is not a desired value as an off-cycle (This terminology is due to the intended use of the stopwatch for usage timing). Current-count represents the amount of time that the attribute has been in an on-cycle if it is currently in an on-cycle (current-count is 0 during an off-cycle). Counting shows whether or not the stopwatch is in an on or off cycle. Since the value of the attribute “position” of the object “TEST-1” is open (one of the desired values), the stopwatch is currently in an on-cycle. The on-time represents the last complete cycle that was on (this does not include the current cycle). The off-time represents the last complete off-cycle (in this case, it would be the previous cycle since it is currently an on-cycle).



POSITION-OF-TEST-1-STOPWATCH-1, a np-timer-stopwatch

UUID	"3b0f7036d91511e78e6da08cfdd7aabb"
Notes	OK
Item configuration	none
Names	POSITION-OF-TEST-1-STOPWATCH-1
User	test-1
State	on
Start time	69.744
Update interval	0.002
Reset	false
Timer existing	true
Tracked attribute	position
On cycle values	sequence (the symbol open, the symbol partially-open)
Current count	21.227
Counting	on
On time	40.174
Off time	5.599
Timer history	the np-timer-history POSITION-OF-TEST-1-STOPWATCH-1-HISTORY

TEST-1, a testing-object

UUID	"0ab1aa75c49611e78e6bcad719bf52b5"
Notes	OK
Item configuration	none
Names	TEST-1
Position	open
Light	on

ON-OF-TEST-1-STOPWATCH...

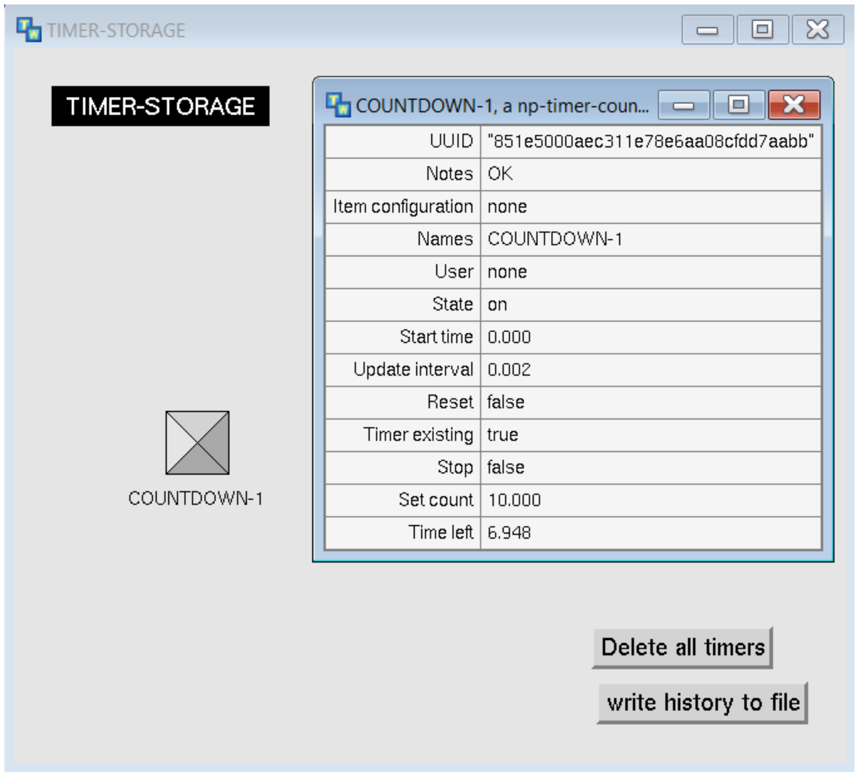
UUID	"4f406740d91511e78e6da08cfdd7aabb"
Notes	OK
Item configuration	none
Names	POSITION-OF-TEST-1-STOPWATCH-1-HISTORY
Initial counting	off
On history	sequence (structure (start-time: 23.941, stop-time: 64.133), structure (start-time: 69.725, stop-time: 0.000))
Off history	sequence (structure (start-time: 20.002, stop-time: 23.941), structure (start-time: 64.133, stop-time: 69.725))

Figure 7: An example of a running stopwatch

There is also a history storage option for the stopwatch. As seen in the bottom right of Figure 7, there is an object named “POSITION-OF-TEST-1-STOPWATCH-1-HISTORY” which itself is an attribute of the stopwatch. This object stores the times when the stopwatch transitioned between different cycles. On-history stores the times that the stopwatch was in an on-cycle, and off-history stores the times that the stopwatch was in an off-cycle. If a cycle is currently running, then its stop-time will be 0.000, as seen in the second cycle in on-history. In addition, the history times can also be written to an excel file if desired.

Functionality of the Timer Module – Countdown

In addition to the stopwatch functionality, a countdown functionality was also implemented. For sake of length, it will not be discussed in as much detail as the stopwatch timer was. Unlike the stopwatch timer, the countdown timer works just as the name suggests. It takes a set amount of time as an input, waits for that set amount of time, then performs an action. In the countdown's current form, the action performed is simply a notification to the user, but can be replaced with any chunk of code in the future.



UUID	"851e5000aec311e78e6aa08cfdd7aabb"
Notes	OK
Item configuration	none
Names	COUNTDOWN-1
User	none
State	on
Start time	0.000
Update interval	0.002
Reset	false
Timer existing	true
Stop	false
Set count	10.000
Time left	6.948

Figure 8: An example of a countdown

The countdown timer also has pause, reset, stop, and override countdown features. The set-count attribute determines how long the countdown will wait. When the "state" attribute is off, the countdown will pause, resuming once "state" is on again. When the "reset" attribute is on, the timer will restart the countdown. When the “stop” attribute is on, the countdown stops, and does not execute the action. If a new instance of a countdown-run method is started with the same countdown object, the new countdown time will override the existing one and the countdown will reset.

Documenting the Module

Since my time working on this timing module was limited, someone else would have to be able to use my work after I left. This meant that documentation was extremely important. One of the items of documentation I made was a few activity diagrams of the main timer running methods. The activity diagrams were just a matter of tracing how the methods modified the timer attributes and displaying them visually.

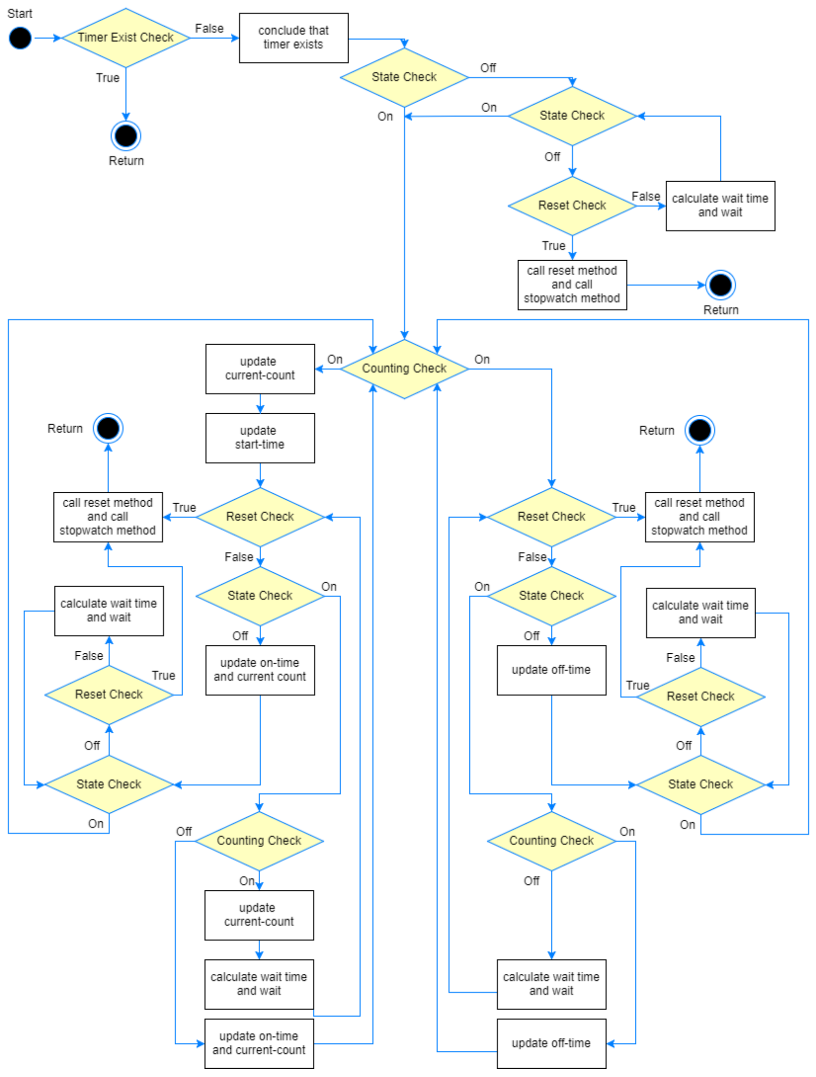


Figure 9: The activity diagram for the stopwatch-run method

Conclusion

Development of this application required a significant amount of brainstorming and critical thinking. The state timer can now be potentially used for usage timing and the countdown can be used for sequence step timing, meeting the initial requirements. Documentation for the code is also complete, with comments in the code, a user guide, and a list of future improvements. Once this code passes further tests and improvements, it can be integrated into the INSIGHT program, where it will provide more information in decision making and aid in autonomous control of the HPGF.

References

[1] Technology Development & Transfer - Stennis Space Center. (2017). NASA Platform for Autonomous Systems (NPAS) - Technology Development & Technology Transfer. [online] Available at: https://sites.google.com/a/nasa.gov/tdt/technology-development/NASA_Platform_for_Autonomous_Systems [Accessed 7 Dec. 2017].

[2] John Valasek, "Integrated Systems Health Management for Intelligent Systems", Advances in Intelligent and Autonomous Aerospace Systems, Progress in Astronautics and Aeronautics, pp. 173-200.

[3] (2007) Expert System Shells, Environments and Languages. In: An Introduction to Knowledge Engineering. Springer, London

[4] Businesswire.com. (September 13, 2006). General Atomics Selects Gensym G2 Rule Engine for Power Systems Management; G2-based Solution to Help Ensure Availability and Readiness of Mission-Critical Facilities. [online] Available at: http://www.businesswire.com/news/home/20060913005997/en/General-Atomics-Selects-Gensym-G2-Rule-Engine [Accessed 7 Dec. 2017]

[5] G2 Reference Manual. (2017). Ver. 8.4 Rev. 2. Burlington, MA: Gensym Corporation, pp.4-11.