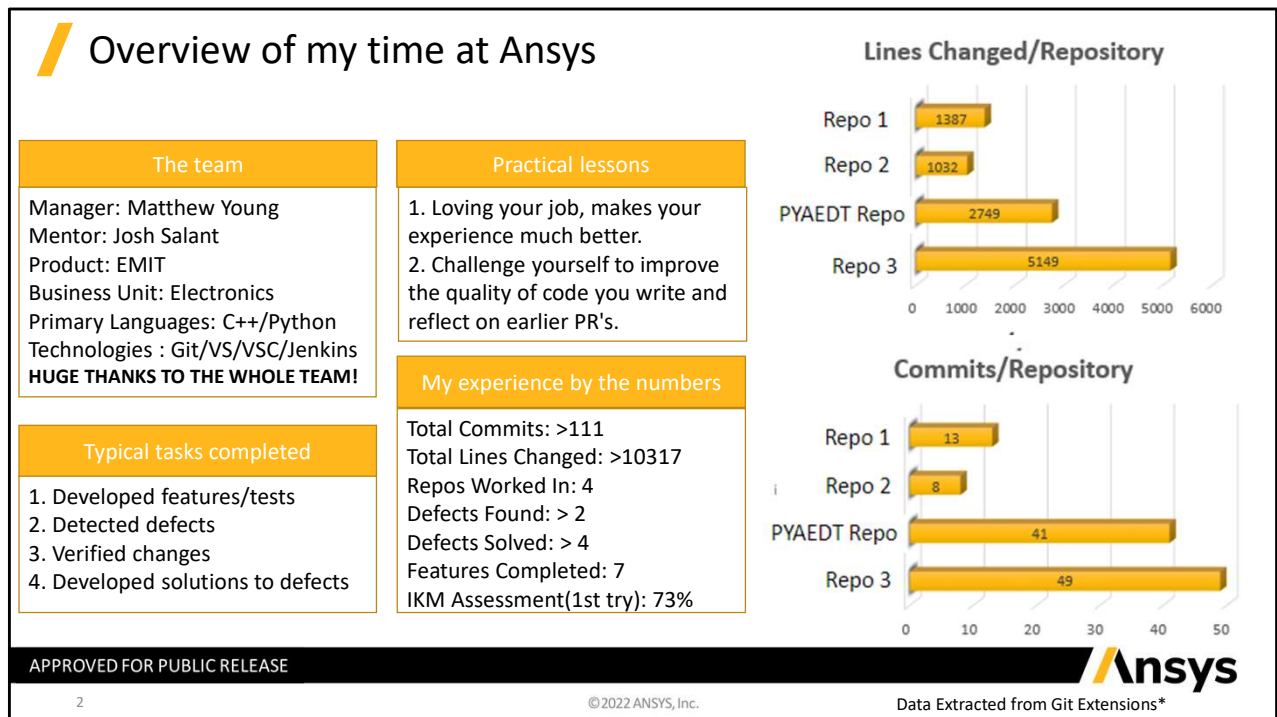


Internship Showcase Presentation Chandan Chandel

November 29, 2022

APPROVED FOR PUBLIC RELEASE

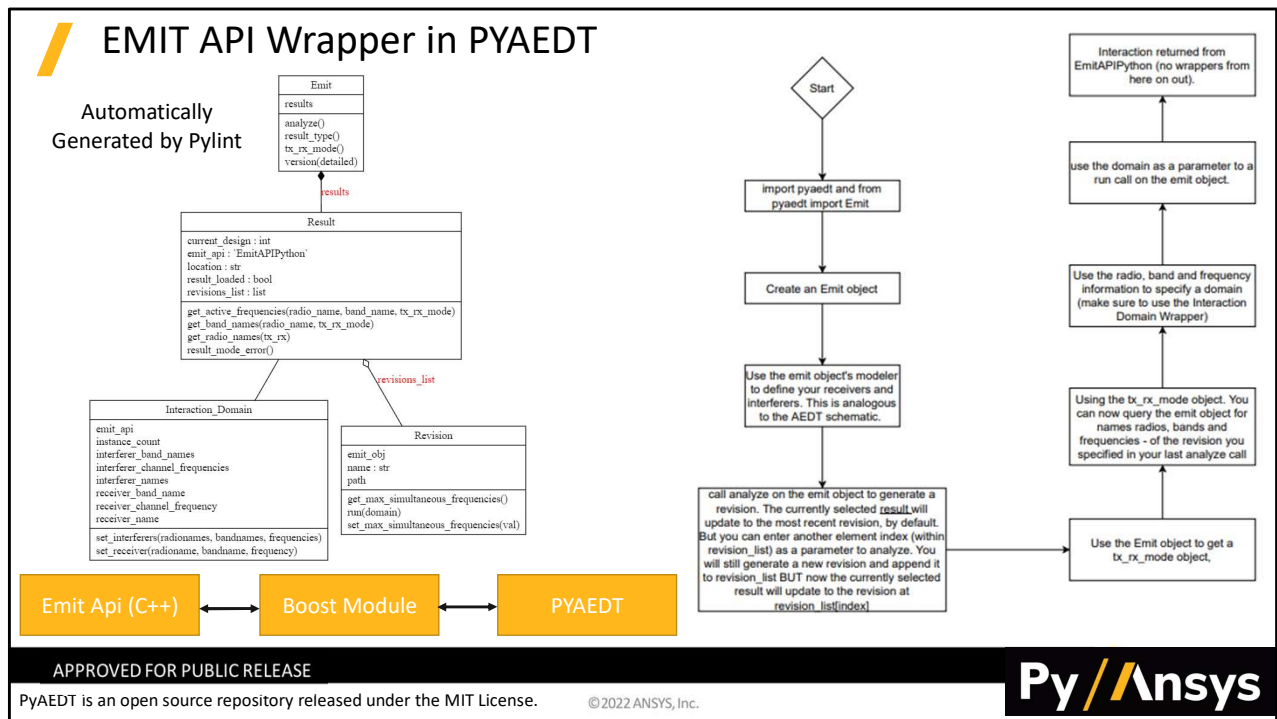




So let's start with a quick overview of my time at Ansys. I worked primarily in C++ and Python using technologies such as Git/ Visual Studio / Visual Studio Code & Jenkins while collaborating with every member of my team at some point. I'd like to give a special shout-out to Josh who helped me get setup and guided me closely through many of my earlier projects and Matthew who always had interesting insights and solutions while trusting me with some very cool projects. I have two practical lessons that helped me grow through my time at Ansys. The first would be to make sure you love your job – by doing so it'll make it easier for you to excel and learn faster. The second is to keep trying to improve the quality of code you write, try reflecting on old PR's to help you through this process.

I'm also going to try and provide a more quantitative overview of my time at Ansys. Before the end of my last month on my second rotation as an intern I worked on 7 features, and found at least 2 defects and solved at least 4 defects – this helped me improve my C++ skills and I scored a 73% on the IKM assessment. I also extracted some data from Git and I found that I had made at least 111 commits and changed 10317 lines while working in 4 repos. You can see most of my commits were in PYAEDT and in repo 3 but somehow PYAEDT has a discrepancy in the ratio of commits vs. The number of lines changed and that's a reflection of the nature of the work I

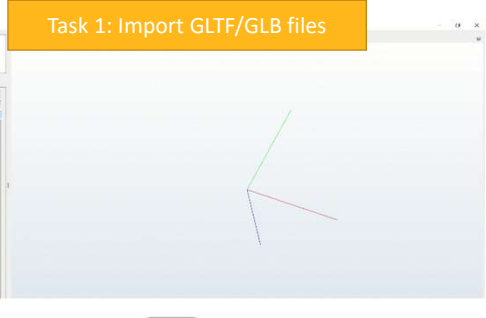
was doing which I'll discuss on the next slide.



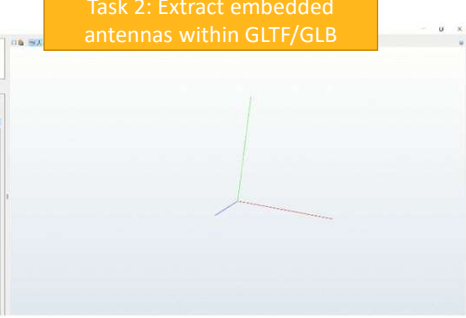
PYAEDT is an open source repo which is why I can mention its name. I was super fortunate to be trusted with the first big thrust into the EMIT team's PYAEDT contribution on the python side, I designed an API wrapper in python and worked closely with a teammate who worked on the C++ side and sought out input from other team members. I mentioned that there was a huge discrepancy in my ratio of commits to lines changed within this repo – this was partially because of the fluid nature of this work which required multiple rewrites and focused on generating a user friendly API while collaborating with and taking input from multiple team members. I also made some changes to the C++ side to enable successful transmission of some datatypes and was briefly exposed to the boost module through this project. The API wrapper and it's structure is still in flux to this day but I used a feature within Pylint to generate a UML diagram which sort of shows the structure of the wrapper to date.

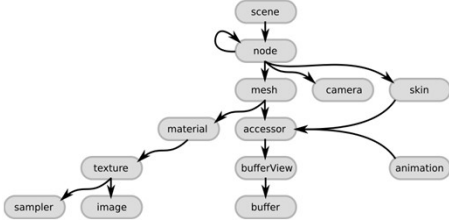
Antenna Extract from GLTF/GLB file

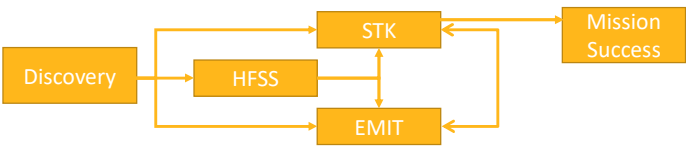
Task 1: Import GLTF/GLB files



Task 2: Extract embedded antennas within GLTF/GLB







APPROVED FOR PUBLIC RELEASE

Gltf structure: https://github.khronos.org/gltf-Tutorials/gltfTutorial/gltfTutorial_002_BasicGltfStructure.html

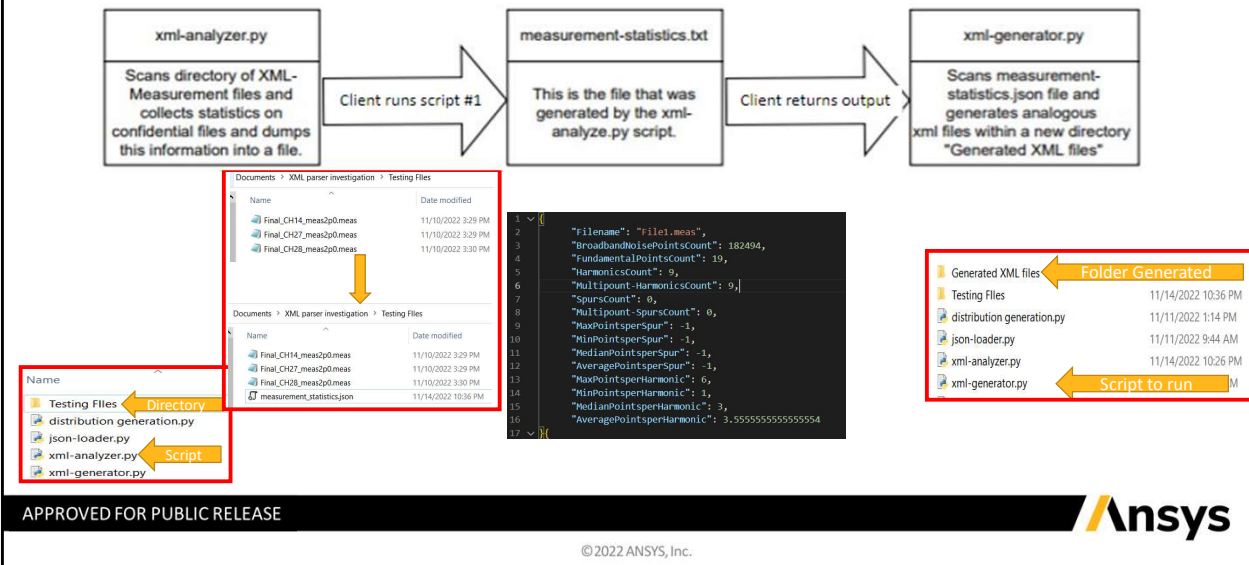
My first major project was split over two stories. For some background Ansys's CAD editor software (Discovery) has been integrating into a workflow with the Electronics Business Unit.

Discovery supports gltf files and would like the ability to export those files to EMIT. It was suggested that I start by investigating how to read a gltf file using a parser and once I worked out the structure of the gltf data I was able to parse through it more effectively. The basic process was to start a depth first search through the nodes. I would reach into it's mesh to grab a whole bunch of data like transforms, color and the primitives that make up that node. I used this point data to generate triangles and assign a color to it. Luckily EMIT already has the infrastructures to take in triangles with colors and construct 3D objects from them, so I basically fed the data that was generated from the gltf file to that pre-existing engine.

A detail to note is that Discovery had added antenna "mount points" which can be exported to STK via the gltf format, which allows you to embed all kinds of information. EMIT doesn't use the CAD, but when importing the gltf it would be nice to automatically read in antenna locations that are associated with them . As a result the second major story associated with this project involved reaching into the

extensions of each node and checking for mount points and extracting that point data, running the transforms and then generating generic antennas at the specified locations and placing all of those nodes under a single group node so as to treat them as one object. This was an incredible learning project and I am so grateful to Matthew & Josh for encouraging me to take this project on at the start of my time here, and guiding me because it helped me so much and improved my skills and confidence.

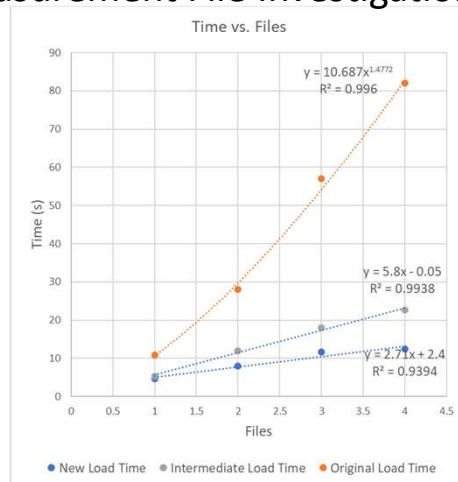
Measurement File Investigation (Part 1)



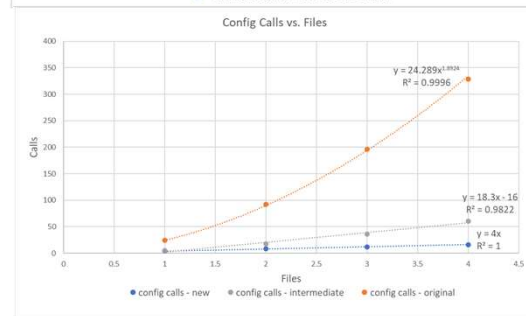
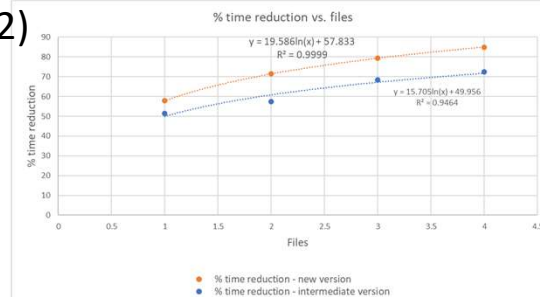
This next project has three parts some of which is still being worked on. And it all start with one of our clients. This client had measured a bunch of radios and generated xml measurement files from them and are now trying to use them with our EMIT product which theoretically they should be able to do, except that the processing is very slow.

The first bit of work that was done was to try and understand how big these files were and what kinds of information was in them, unfortunately there were restrictions on our access to those files. So my mentor Josh suggested that I write a script that the client could run which would scan a directory full of these files and spit out statistics on them which could then be given to us and then I could use a second script to generate analogous xml data, without any of the proprietary info in it so we can have a clearer picture on the data they are trying to upload. So that was part 1.

Measurement File Investigation (Part 2)



Hot Path: is the most expensive call path based on either sample count (Sampling Profiling) or execution times (Instrumentation profiling). Hot Path shows the branch of your application's call tree with the highest inclusive samples or inclusive time.



APPROVED FOR PUBLIC RELEASE

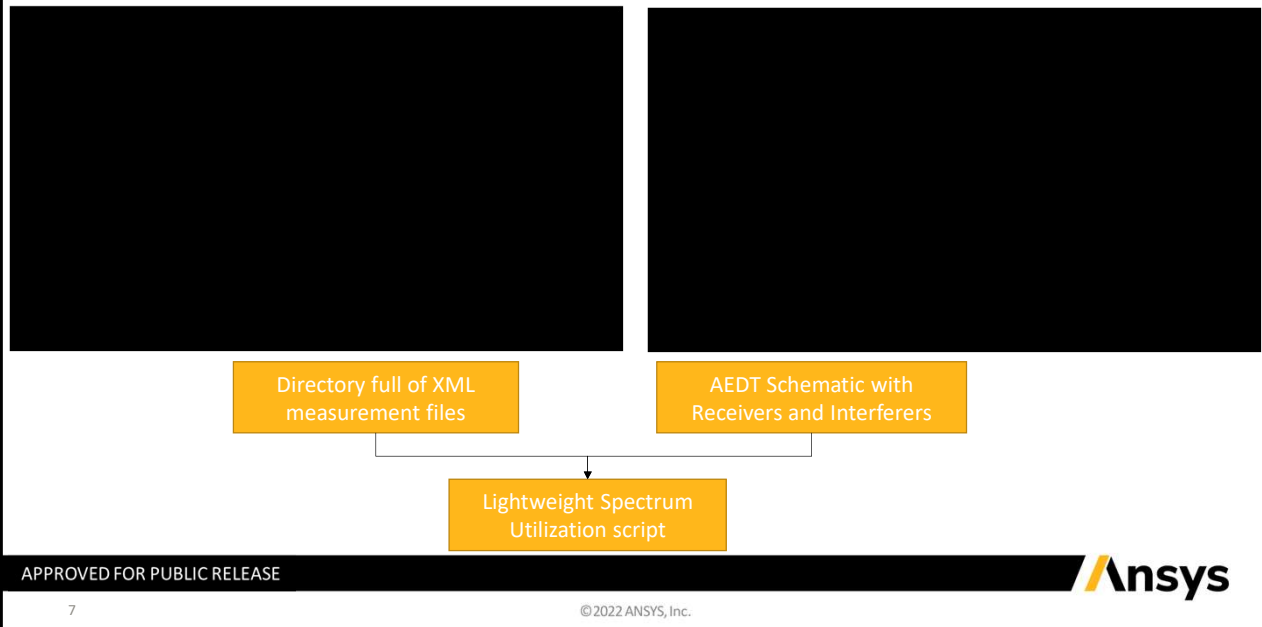
© 2022 ANSYS, Inc.

ANSYS

Part 2 was focused on Performance Profiling to find the sections of the program which were slowing the upload process down. Using tools like the Hot Path I was able to identify a couple of methods that were a source of most of this bloat. My first instinct was to try and slim those function down, but Matthew pointed out that these were relatively simple functions and that a better approach would be to check and see how often it is being called, and from there try to gauge whether or not it's being called too often. So I used static variables to count the number of time the affected method was called and found that there were 24 times more calls to the function than there were applicable places in the uploaded document where it could be used. Once I implemented some limits to the number of calls being made to the update there were huge improvements that can be seen in the bottom right graph which show a transformation of the number calls used to initiate an update of the measurement node from an $O n^2$ function to a linear $O n$ function, the problem was that each measurement node that was added would trigger a refresh of its siblings. But we weren't done yet the next round of performance profiling showed that we were still spending a huge amount of time constructing Trace2D objects and it was suggested that I should store these Trace2D objects instead of rebuilding them so many times. And that gives us the final product with the Time vs. Files graph shown on the left, and the % time reduction vs. Files uploaded graph in the top right. So for

example if you were to upload 4 files you would have originally spent 82 seconds vs 12 seconds and so you should save about 84% of the original time.

/ Measurement File Investigation (Part 3)



And finally the last branch of this measurement file investigation arose when my mentor Josh suggested that I could expand a lightweight script that we already have which allows you to quickly check all radios in the AEDT schematic and generate a profile of which frequencies each radio band occupies. Josh pointed that a directory full of these measurement files has most of the data we would be getting from the AEDT schematic and once that data is extracted and transformed it can be fed to this light weight script. You can see in the two gifs above that we are basically able to use an entirely new source of input to our script.

Spectrum Utilization Defect

Correct band stop value in this case is 0.10000125

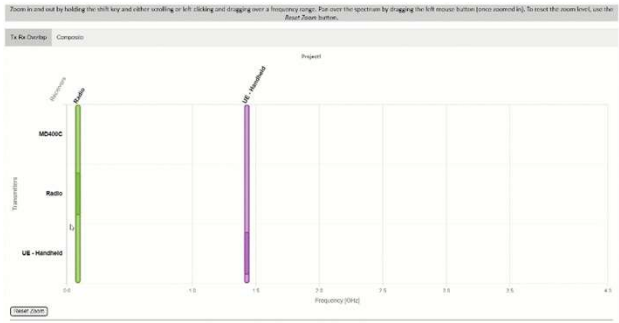
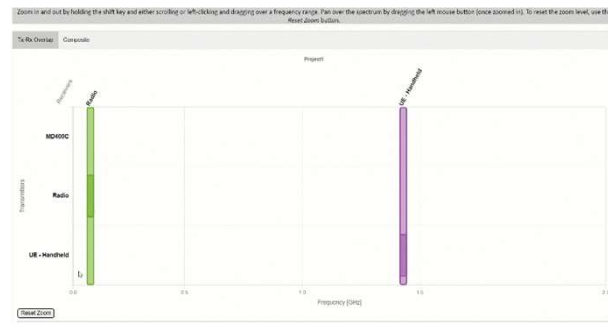
Defect shows value of 0.116225

Original defect behavior with incorrect values for very small bands

Python File grabs AEDT schematic data and performs pre-processing

JavaScript/HTML/CSS is generated

All bands are always displayed as at least 1% of visible x-axis and maintain correct values upon closer zoom



APPROVED FOR PUBLIC RELEASE

ANSYS

© 2022 ANSYS, Inc.

This last project that I will discuss was also on the same lightweight script that I mentioned in the last slide. It is a relatively minor project that I took on but I had fun and delivered a solution that I was proud of because of the care I took for the user experience. So the original notes for this defect basically said that when higher frequency radios are added to the chart the smaller radio bands have incorrect values when you really zoom into it. So I started investigating what was going on here and basically realized this done on purpose by the original developer to ensure that really thin bands were still visible and the more the chart is getting stretched out the more incorrect the thin bands will need to be so that they are visible. I came up with a solution to this by ensuring that the minimum size a band can take up on the chart is 1% of the x-axis but on each zoom the value is updated constantly and once the true size of the band reaches 1% of the chart size then a seamless transition happens to the true value as shown in this gif.



APPROVED FOR PUBLIC RELEASE