



# A Programmer's Training for ESP UDPs, UDFs and CAPS' AIMs The Engineering Sketch Pad – Rev 1.28

Bob Haimes

[bob@geocentrictech.com](mailto:bob@geocentrictech.com) or [haimes@mit.edu](mailto:haimes@mit.edu)

and

John F. Dannenhoffer, III

[john@geocentrictech.com](mailto:john@geocentrictech.com)

Geocentric Technologies LLC

- You are a **C/C++** programmer
- You are here with a laptop containing:
  - **C & C++** (and optionally FORTRAN) compilers
  - ESP Rev 1.28 (or a recent 1.28 Beta)
  - OpenCASCADE 7.8.1 (from [HTTP://acdl.mit.edu/ESP](http://acdl.mit.edu/ESP) or a PreBuilt distribution)
  - A functioning Python at 3.12.10 or higher  
If from [HTTP://acdl.mit.edu/ESP](http://acdl.mit.edu/ESP) – Run the install script and make this the default Python in the shell/command-prompt used for ESP development
- ESP successfully built (and run) from source
- A desire to write an ESP UDF/UFD and/or an AIM (and hopefully in idea for one that you would like to work on!)

**If any item above is not true – you do not belong here!**



**Bob Haimes**

[bob@geocentrictech.com](mailto:bob@geocentrictech.com) or [haimes@mit.edu](mailto:haimes@mit.edu)

**John F. Dannenhoffer, III**

[john@geocentrictech.com](mailto:john@geocentrictech.com) or [jfdannen@syr.edu](mailto:jfdannen@syr.edu)

**Nitin Bhagat**

[nbhagat1@udayton.edu](mailto:nbhagat1@udayton.edu)

## Ideally a software system should:

- Work for user, not the user working for the system
- Be based on a mental model that is *easy* for the user to grasp
- Never lose anything they have done (backward compatibility)
- Solve the user's **real** problem, not their stated request
- Be responsive to the changing needs of the users
- Never surprise the user
- Not be reverse-engineered
- Be tested thoroughly

## This means that:

- The programmer *takes a hit* so that the user's life is easier

**We do not always succeed!**

## ESP is:

- a parametric geometry creation and manipulation system designed to **fully support** the analysis and design of aerospace vehicles (**aCAD+**)
- a stand-alone system for the development of geometric models
- an API can be embedded into other software systems to support their geometric and process needs (e.g., CAPS)
- extensible so that users can add their own geometric *features*

## ESP is not:

- a full-featured mechanical computer-aided design (**mCAD**) system
- a system to be used for creating “drawings”

## CAPS is:

- a software system that allows for building complex workflows
- a system that supports multidisciplinary and multi-fidelity analyses with direct access to geometry and its attribution
- a system that simplifies inter-analysis communication
- an API that provides access to sensitivities (supports gradient-based optimization)
- software designed for aircraft design settings
- extensible so that additional analyses can be supported

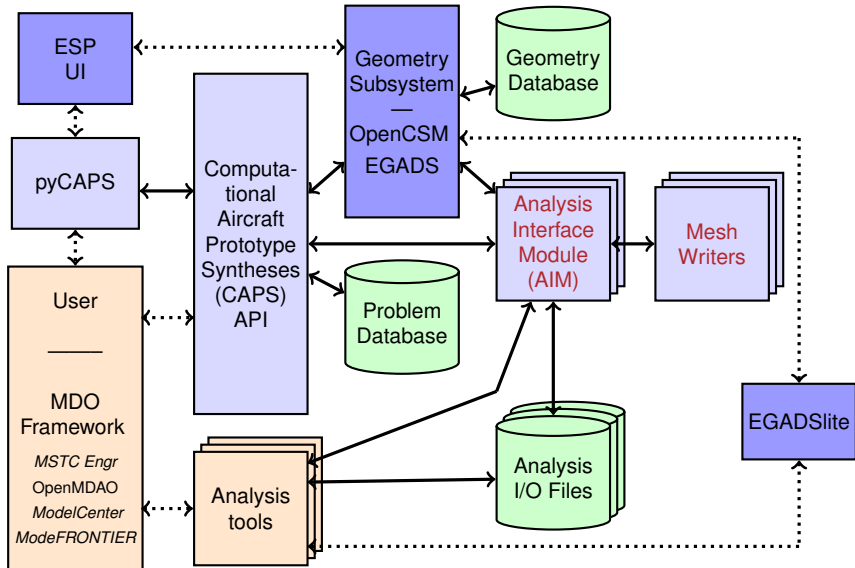
## CAPS is not:

- an MDO Framework (but can support them)

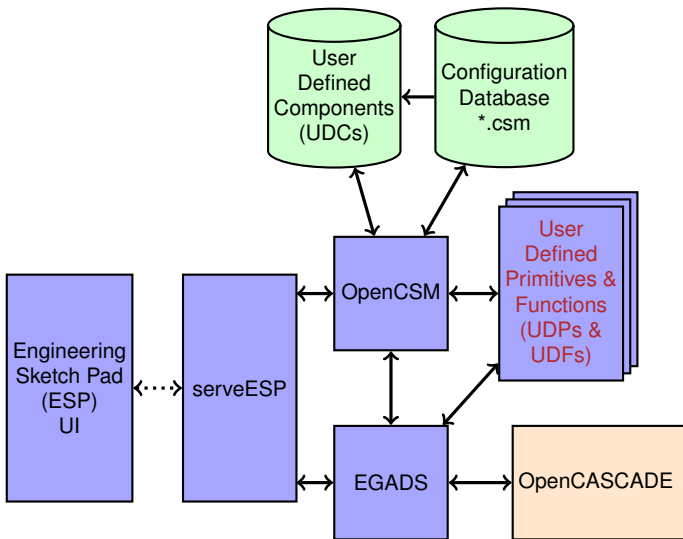
Programmers can add functionality to ESP in the following ways:

- User-Defined Primitives (UDPs)
  - Create and returns an EGADS Body (usually a *Solid*) to be placed on the CSM stack
- User-Defined Functions (UDFs)
  - Has access to other Bodies on the CSM stack
  - Usually creates a new EGADS Body through some form of manipulation and add back to the stack
- Analysis Interface Modules (AIMs)
  - Interface between the CAPS infrastructure and any Analysis & Meshing codes
  - Provides the ability to generate Analysis/Meshing inputs, perform execution and then access the results
  - AIM *Mesh Writers* are also handled in a similar manner

**All are dynamically loaded at run-time**







## Build from source

Use the same shell with the same environment as was used for building the rest of the system

## Build from within a PreBuilt distribution

Make sure you:

- Have a C/C++ compiler and *(N)Make* available in your shell/command-prompt
- Set the environment by running:
  - MAC/Linux: Double-click on the ESP 1.28 desktop icon  
Use the opened window for your development
  - Windows: Startup a command-prompt that supports Visual Studio  

```
> cd %ESP_ROOT%      (the location of EngSketchPad)
```

```
> ESPenv.bat
```

## Built like a shared library

- Windows – a dynamically loaded library (.DLL)
- LINUX – shared object (.so)
- MAC – a *bundle* – we use the extension .so

## Loaded at run-time

- Opened with `LoadLibrary / dlopen`
- Function pointers by name via `GetProcAddress / dlsym`
- Can have multiple *instances*, each may require its own *state data*
- Called by function pointer and *instance* via a dispatch table (i.e., name is only used to retrieve the pointer)
- Shared object/DLL not closed so that *valgrind* can report symbols

**Of prime importance** – this is the first training of its kind.  
We need feedback to improve future versions!

- Opportunity to provide anonymous (or named) immediate feedback for anything “not clear” (e.g. muddy)
- Ask questions about presentation material, previous exercises, point out errors, make suggestions, ...
- Questions will be answered at next session
- E-mail questions (post-training) to any member of the team

- 01 Plugin Overview (this session)
- 02 EGADS – Programmatic use of Geometry in the Plugins
- 03 OpenCSM UDP Basics
- 04 OpenCSM Full-featured UDP
- 05 OpenCSM Full-featured UDF
- 06 CAPS' AIM Overview
- 07 AIM Software Structure
- 08 AIM Utility Functions & Tessellations
- 09 AIM Analysis Input, Execution and Output
- 10 AIM Sensitivities
- 11 AIMS and Units
- 12 AIM Bounds & the AIM Discretization Structure
- 13 AIM Mesh Writing
  - AIM Documentation, Distribution & Testing Discussion