Introducing **PyDUIT**

A new **Py**thon-native, **D**eclarative **U**ser **I**nterface **T**oolkit

By

Steven E. Summers

Begun: 9/05/20
Rev C3: 11/18/20

About Me (and this format)

I'm a software designer, retired after 30 years of:

- Helping to start a company and building it to 200+ people
- Writing a POS (Point of Sale) system using Turbo Pascal for DOS, for machines with 640K RAM & 20MB hard drives (mostly solo);
- Then implementing (mostly designing) a second POS system in Delphi Pascal for Windows (with a team of 20+ people).

I'm programming again, but now in Python, for "fun".

- I've had more interesting ideas since retiring than I ever had when I was focused on automating car washing, and Python is great for prototyping (except for making apps with GUIs. ☹)
- While I don't personally have enough time left in my life to chase even a fraction of these ideas, *I do have the financial resources to start a small company to do "implementing" parts for me.*
  - I still won't have time for *all* the ideas (especially since I keep having more!), but with help, more of them can be realized.
- This project is, I hope, the first of many, some of them far more "monetizable", but probably none more fun (for me at least).

A note about this format:

- I tend to write long paragraphs with too many subordinate clauses and not enough periods, describing things that can't be visualized with words unfamiliar to most people. It all makes sense to me (at the time I'm writing it, anyway!), but it doesn't convey the ideas well to others. ☺
- I have found that if I use PowerPoint instead of Word, I use fewer words, break ideas into smaller pieces, and include more mockups and diagrams. The result is much easier to understand.
- However, despite being created with PowerPoint, this format is *not* meant to be a “Presentation”.
  - Think of it more like a comic book version of the ideas, while my Word docs would be the "War and Peace" version (that nobody would read.)

I hope it works well for everyone. Feedback is welcome.

Introduction

- I started learning Python just over a year ago.
  - Almost immediately I needed a way to make apps with Graphical User Interfaces (GUI's)
    - Delphi is *great* for that. (and I took it for granted for 20 years!)
  - Python has a *lot* of alternatives for it (40+ last I counted).
    - I studied the top six candidates for my needs, and learned two (**WxPython** and **Kivy**) well enough to write apps with them.
  - While both have good qualities, neither is what I want.
- Well, ***what*** **do** ***I want in a GUI toolkit?*** A quick list:
  - Build apps for all common operating systems:
    - **Desktop** – Windows, Mac, Linux, even Raspberry Pi’s; **& Mobile** (multi-touch) – Win-10 in tablet mode, Android, iOS
      - Support multi-touch UI’s but still work well with a mouse;
    - Apps that **look “natural”**, even if not quite “native”
    - Have separate, appropriate component sets for each platform
      - But use the same tools, API, etc, regardless of which platform/library
      - Components in Python, not a C/C++ library, so they can be studied.
  - **Easy** to learn, **fun** to use.
    - Needs some kind of Tool to simplify defining the UI.
      - Eliminate need to memorize the component & property names
    - Delphi-like "rich" controls, usable without complex coding.
      - Accessible to beginners, still powerful enough for any use
    - Use for anything from games to mobile to business apps.
      - Component libraries available for every app type/need – e.g. writing games, databases, web service clients, graphing/plotting, media, etc.
  - Components & libraries easily **managed** & **modified**.
    - Tools to customize/save/organize/reuse and ***share***.
      - Ideally, a central place to share them, with *many* available.
  - Good **documentation**, tutorials, and user community
    - Well maintained with quick bug fixes & frequent releases.
- So basically, I want **"Visual Python for any OS"**

I do realize that's asking a lot, but it should be possible.

Introduction, Continued

- After a thorough search, I’ve found no existing Python GUI libraries that come close to matching my wish-list.
  - I suspect many Python users have a similar unfulfilled list.
    - Building GUI apps may be a declining fraction of Python use, but it’s still common, and the existing tools are hard to use.
    - Evidence: **PySimpleGUI**, the newest Python GUI in the top few, has been downloaded *over a million times* in <3 years.
- However, While learning Kivy,
  - I thought of some ideas for combining Delphi's strengths with Kivy's, that could ***create*** a tool that would match my wish-list, far better than anything else available for Python.
  - I *love* the idea of building a GUI toolkit. It's exactly the kind of thing I've enjoyed doing most during my career.
    - However, there are other things I enjoy doing too.
- So the big question is, *how much work would it take*?
  - And the next follow-up question is, would it provide enough value to be worth the effort it takes (whatever that is?)
    - Many of the 40+ GUI libraries listed in Python's Wiki are dead, having never reached a "critical mass" of users.
      - It wouldn't make sense to spend a year or more of developer time (let alone *my* time) even if we ended up with 10,000 users.
  - Which brings in the most important question:
    - Would these ideas really produce something that becomes the top (or one of the top few) Python GUI tools,
      - so the development cost per end-user stays reasonable (e.g. < $1)?
- To find out, I've done some analysis & design work.
  - This document will explain (without *too* much detail, I hope), what I think is involved, and the possible benefits.

TL;DR version: *Lots of work, potentially **amazing** benefits!*

Introduction, Continued

- Unfortunately, ***FAR*** too much work to do by myself.
  - BUT: This is an *excellent* opportunity to build a team of young, energetic developers happy to make $20+/hour (+equity!) working part time from home on fun projects.
    - I already have several excellent candidates and don't expect difficulty finding more. (I'll just look for "smart" > "educated"!)
    - At those rates I can fund about 6000 hours a year until the company becomes self-sustaining.
    - I am *also* willing to put in some hours myself – say, quarter-to-half time, or more if I'm having fun rather than working. ☺
  - This project alone might not justify that much effort, but:
    - I have many other ideas (some *much* more monetizable) that won't ever get developed if I have to do them myself;
    - So building the team (& company) is not *just* for this project.
      - Even if this GUI toolkit isn't successful, the *next* project(s) might be.
- But why this project, and not a more profitable idea?

  (or, "How do I rationalize investing $100K+ and hundreds of my own hours to produce an open source software development tool to give away for free?")
  - Here's how I see it:
    - I could build a commercial product and try to find buyers willing to pay for an unknown product from an unknown company;
    - Or, I could build a free product that fills a *definite* need, and get upwards of 1M users (out of >12M Python programmers.)
  - I think this roughly $100K investment could produce:
    - A million+ potential future customers who use and like our product, and recognize our company's name (for 10¢ each!)
    - A toolkit that makes Python *really good* at producing GUI apps, that lots of people (especially ME!) enjoy working with.
    - Possibly the best thing I've ever invented/designed (so far)!
- Remember, 'ROI' doesn't have to be in $.
  - Having a million people like what we make? ***Priceless***!

The Basic Ideas

" Build apps for all common operating systems…"

- Kivy works on all of the major OS's by using SDL2: the "**S**imple **D**irectmedia **L**ayer". **We will use it too**.

  SDL2 is used for Valve's Steam game stuff, so it's well supported.

  - This provides everything needed to host an application –
    - Multiple windows (even on Android. Kivy only allows one);
    - GPU accelerated drawing (2D/3D, OpenGL & DirectX, etc.)
    - Access to input devices (Keyboard, Mouse, Touch, game, etc.)
    - Access to sound playback *and* recording devices (and cameras)
    - Images, Videos, Timers, Events, etc.
  - Only drawback: Controls won't be "Native".
    - Who cares, in 2020? Just needs to look nice, and not "alien".
    - It will be easy to create lots of different "styles" of parts.
- "Easy to learn, fun to use…"
  - Will include a design tool to eliminate name memorizing;
  - Use Delphi's "powerful, flexible components" approach, rather than Kivy's "minimalist" approach that requires coding to access even basic functionality (e.g. multi-select, drag & drop…)
    - Capabilities will include things like "theming", "data awareness", change tracking with undo/redo, rich text editing, and more.
- "Component libraries available for every app type…"
  - Improve on Delphi's "Component Library" with "Part Kits"
    - Tool will make them easy to manage, import, export, etc.
    - Also make them easy to *share* with an "app-store"-like service.
  - Goal: Make it easy so our *users* build all the Parts needed.

PLUS a *Unique Value Proposition* to build our user-base:

- Names, Hints, and Help topics for Parts *and* their Traits (settable attributes) and Events, can *all* be *"localized"*.

  e.g. "button.area.height" → "**botón.zona.altura**"

- Users will be able to do GUI design in their *own languages*.

Building It

Whether or not these features are enough to attract a million users can only be determined by building it.

- That will require the Design and Implementation of at least the following sub-projects:
  - Core functionality:
    - Our interface to the SDL2 platform, for window management, drawing, font rendering, input handling, and everything else;
    - Our Declarative User Interface language ("DUI") processor, and the infrastructure for creating Parts as declared;
    - Our overall framework that connects GUIs to their Apps, and organizes Parts into Kits, so GUI creation can be automated.
  - UI "Parts": (a.k.a. Components, Widgets, Controls...)
    - "Complete" set of parts, at least on par with Delphi 1.0 (1995)
    - All parts support themes; Controls (e.g. buttons, menus) utilize Actions; Fields (e.g. StrField, NumField) are "data aware", etc.
  - The "DUITool":
    - Editor for DUI language – provides lists of Parts, Traits, etc…
    - Manager for Parts Kits – organize, import, export, document…
    - Part Editor – define traits, set defaults, add/replace drawing command lists, edit hints and help topics, etc.
    - Other utilities – *lots* of ideas for making this very powerful.
  - "Partsmart": (until a better name comes along?)
    - We will need a web site and/or web service (with client built-into the DUITool) for downloading & uploading custom part kits
      - Like an app store – with ratings, reviews, and good search tools
  - Documentation and Testing:
    - For this to be successful, it has to be reliable and safe to tweak.
      - It will *definitely* need to incorporate automated testing.
    - It will also need good documentation, so our part-sharing users can correctly build reliable parts.

The rest of this document describes these in more detail.

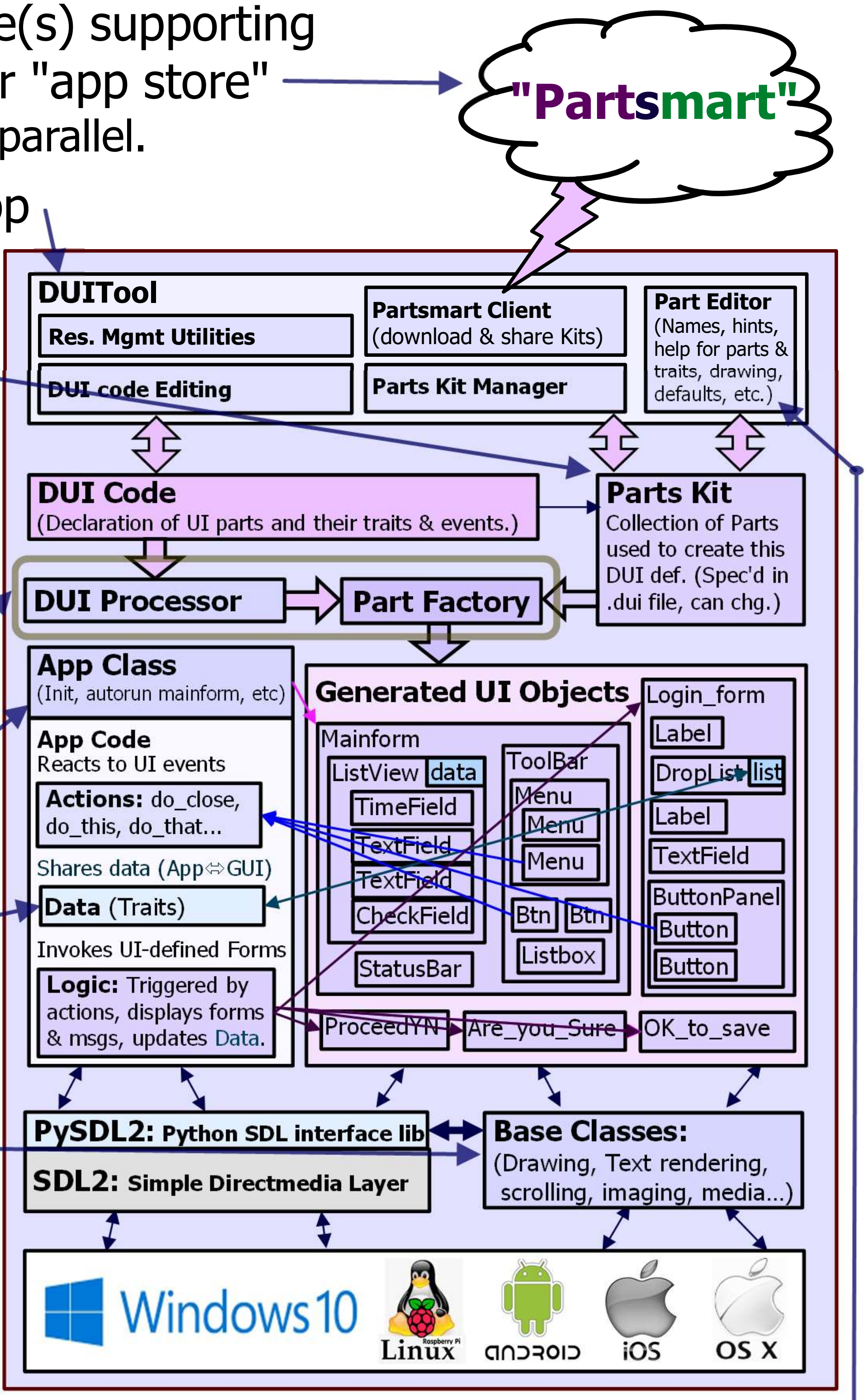
PyDUIT System Architecture

The four subprojects (in reverse order):

- The web service(s) supporting "Partsmart", our "app store"
  - Can be built in parallel.
- The DUITool app
  - Must build last (*Using* PyDUIT.)
- The UI Parts
  - Needs Core modules to be built first
- Core Modules
  - DUI language processor (builds UI)
  - App Class (Handles startup process, etc.)
  - Traitlet library (Needs extensions to meet our needs)
  - Base Classes, (For building Part classes)

+Documentation

- Will include Help topics for our parts, which we will edit with the DUITool's Part Editor.



"Partsmart"

DUITool

Res. Mgmt Utilities

DUI code Editing

Partsmart Client (download & share Kits)

Parts Kit Manager

Part Editor (Names, hints, help for parts & traits, drawing, defaults, etc.)

DUI Code (Declaration of UI parts and their traits & events.)

Parts Kit Collection of Parts used to create this DUI def. (Spec'd in .dui file, can chg.)

DUI Processor

Part Factory

App Class (Init, autorun mainform, etc)

App Code Reacts to UI events

Actions: do_close, do_this, do_that...

Shares data (App⇔GUI)

Data (Traits)

Invokes UI-defined Forms

Logic: Triggered by actions, displays forms & msgs, updates Data.

Generated UI Objects

Mainform

ListView data

TimeField

TextField

TextField

CheckField

StatusBar

ToolBar

Menu

Menu

Menu

Btn

Btn

Listbox

Login_form

Label

DropList list

Label

TextField

ButtonPanel

Button

Button

ProceedYN

Are_you_Sure

OK_to_save

PySDL2: Python SDL interface lib

SDL2: Simple Directmedia Layer

Base Classes: (Drawing, Text rendering, scrolling, imaging, media...)

Windows 10

Linux

Raspberry Pi

android

iOS

OS X

Core Modules

(Starting from the bottom again…)

- **Base Class examples:** (Foundation classes for Parts)
  - Area: Drawing surface, w/ Position&Size, & drawing cmds;
  - Font: Converts text strings to texture bitmaps for drawing;
  - Image: Converts various pic formats to texture bitmaps;
  - Traited: Supports Traits, generates Events (base for Parts);
  - Placing, Scrolling, Textual, Editing… (and probably more.)

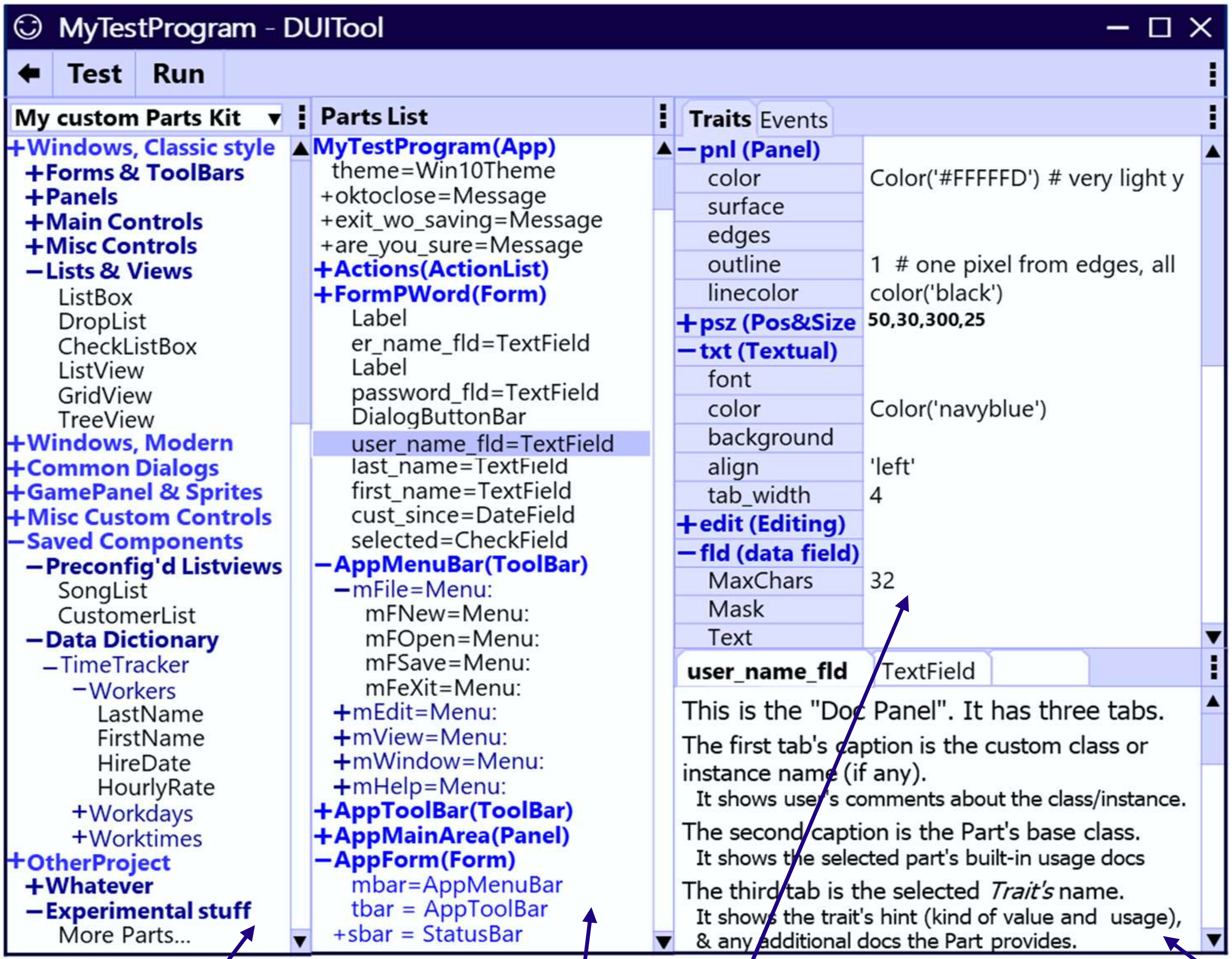
    These work via PySDL2 & family to provide functionality for parts.
- **Traitlets**: (github.com/ipython/traitlets)
  - As designed, this system will require "active properties"
    - Setting button.color='red' must make button redraw itself red.
  - IPython's Traitlets library seems to be a good fit
    - but requires some extending to provide everything we'll need.
- **App Class:** interface between UI & app functionality
  - Automates initialization & startup – easy for programmer.
  - Essentially, this defines our framework
    - Initializes Factory from Parts Kit, processes DUI code to build UI objects, then displays the main form, awaiting UI events.
- **DUI Language Processor:**
  - The heart of the PyDUIT idea is a "DSL" (domain specific language, like Kivy's KV) that concisely declares the GUI.
    - This includes all Forms, Popups, Messages, image and media files, text strings, and anything else ever seen by end-users.
    - Creates a clean "separation of responsibilities" between GUI specification and App functionality.
    - Greatly simplifies creation of our DUITool, for automating the creation, editing and especially *reuse* of the DUI language code.
- **Lots of little things:**
  - Like all projects, there will be lots of "plumbing & wiring", but we'll figure that out as we go.

UI Parts (and Part *Kits*)

- A GUI toolkit is only as useful as its "widget library" (and its applicability to your particular project areas.)
  - One goal of this project is to make creating and sharing them so easy there will be *thousands* of UI Parts available.
    - Unfortunately, ***we*** will have to build all the parts we *start* with!
- We will start with **Windows** and **Android** platforms
  - These will provide experience with both mouse and touch (and they're the two platforms that I use personally right now.)
- Initial **Goals**:
  - For Windows, as good as Delphi 1.0's VCL (back in 1995!)
    - All the basics for desktop business apps like Toolbars, Menus, StatusBar, various Panels, Labels, Images, Buttons, Progress bars, Fields, Viewers, Editors, ListBoxes, CommonDialogs, etc;
  - For Android, equivalent to KivyMD's capabilities, at least.
    - It has 30-40 "Material-Design" widgets for phones/tablets.
  - Non-visible parts too- Themes, Actions, DBConnections…
    - Support for Themes from the very first release, so we don't end up with a dozen different (and incompatible) ways to do that;
  - Also parts for games, media players, other "fun" stuff
    - Encourage use teaching Python – let them teach PyDUIT too!
- Plan is to create ***multiple*** **"Part Kits"**, for different:
  - Platforms: Windows, Android, Linux, Ras-Pi, Mac, iOS. Others?
  - Styles: Windows Classic vs "Modern UI", dark/light themes…
  - Uses: e.g. 2D games, DB controls, graphing, multimedia…
  - Places: Language-translated names/hints/help, RTL vs LTR text… Hopefully our users will help add all this stuff for their own needs, then share their work with everyone else through our Parts store.
- We can start with just a few of these:
  - Form, Message, ToolBar, Panel, TextBox, StrField, Timer… i.e., the minimum necessary to test the Core modules.
    - The rest can be implemented gradually as needed until release.

The DUITool

Once those two projects are far enough along to be usable, we'll build something like this:



```
+Windows, Classic style
  +Forms & ToolBars
  +Panels
  +Main Controls
  +Misc Controls
  −Lists & Views
     ListBox
     DropList
     CheckListBox
     ListView
     GridView
     TreeView
+Windows, Modern
+Common Dialogs
+GamePanel & Sprites
+Misc Custom Controls
−Saved Components
  −Preconfig'd Listviews
     SongList
     CustomerList
  −Data Dictionary
    −TimeTracker
      −Workers
         LastName
         FirstName
         HireDate
         HourlyRate
      +Workdays
      +Worktimes
+OtherProject
  +Whatever
  −Experimental stuff
     More Parts...
```

```
MyTestProgram(App)
  theme=Win10Theme
 +oktoclose=Message
 +exit_wo_saving=Message
 +are_you_sure=Message
+Actions(ActionList)
+FormPWord(Form)
    Label
    er_name_fld=TextField
    Label
    password_fld=TextField
    DialogButtonBar
    user_name_fld=TextField
    last_name=TextField
    first_name=TextField
    cust_since=DateField
    selected=CheckField
−AppMenuBar(ToolBar)
  −mFile=Menu:
     mFNew=Menu:
     mFOpen=Menu:
     mFSave=Menu:
     mFeXit=Menu:
  +mEdit=Menu:
  +mView=Menu:
  +mWindow=Menu:
  +mHelp=Menu:
+AppToolBar(ToolBar)
+AppMainArea(Panel)
−AppForm(Form)
    mbar=AppMenuBar
    tbar = AppToolBar
  +sbar = StatusBar
```

−pnl (Panel)	
color	Color('#FFFFFD') # very light y
surface	
edges	
outline	1 # one pixel from edges, all
linecolor	color('black')
+psz (Pos&Size	50,30,300,25
−txt (Textual)	
font	
color	Color('navyblue')
background	
align	'left'
tab_width	4
+edit (Editing)	
−fld (data field)	
MaxChars	32
Mask	
Text	

user_name_fld | TextField

This is the "Doc Panel". It has three tabs.

The first tab's caption is the custom class or instance name (if any).
It shows user's comments about the class/instance.

The second caption is the Part's base class.
It shows the selected part's built-in usage docs

The third tab is the selected *Trait's* name.
It shows the trait's hint (kind of value and usage), & any additional docs the Part provides.

The left panel is the current "Parts Kit".

You can have many of these, but only one in-use at a time.

Panel can be pinned as shown, or opened only when needed to fit on small screens.

The center panel is the current app's "Parts list".

This shows all the parts being used by the app, and their relationships.

Each entry is a Part, and may have Trait and Event settings that apply to it, shown in the next panel.

This panel is the Trait & Event Settings editor currently showing the traits of the selected part (user_name_fld).

Note: This can also show a single trait or event at a time, for a selected *list* of Parts.

The Info Panel: Shows Hints and Help for selected Parts, Traits & Events.

Also is a notepad for user-entered comments on the Parts & settings.

- This will enable true "Point and click" GUI implementation.
  - No names to memorize, no imports to specify, help always "right there".
  - We might even provide a Python code editor for event handlers, if useful.

More DUITool Functionality

- **String List** Management:
  - Can replace string literals with entries in a string list.
    - Example: `caption="Password"` ➔ `caption=sl[57]`
    - Process is invisible to user. (Always shows the strings in editor.)
  - Allows multiple StringLists, & provides an editor for them:
    - Clone "English.sl" to "Spanish.sl", replace each English string with Spanish equivalent, & select at runtime. Presto! Localized!
- **Resource** Management:
  - Similar idea. Images, Icons, Fonts, & any other files, stored in one compressed file, loaded on startup. Accessed like `image=re['filename']` or `icon=re['iconname']`.
    - Editor shows filename (dictionary key). Clicking opens the resource editor, which can show & modify the images, icons, etc.
- **Part Kits** Management:
  - **Open** existing Part Kit or **create** new, empty one;
  - **Import** (all or parts of) an existing Kit into the current Kit
  - **Add** customized "Parts List" parts into current kit for reuse
  - **Move**, **remove**, **rename**, **recategorize** & **reorder** Kit's parts (including **hiding** without removing from the kit.)
    - Category/subcategory outline is user-defined and easily modified
  - **Edit & Translate** Part metadata defined in Part Kits, e.g.
    - Part, Trait & Event Names, Hints and Help topics
  - **Restyle** components by editing their Canvas Commands and visual Trait defaults (all defined in the DUI code.)
  - **Export** (all or a subset of) the current Kit into a new Kit
    - Option to **Create an *AppKit* –** a Kit w/ same name as the app, but *just* the parts used by the app's DUI code (& no help text.)
      - Useful to minimize distribution size and startup time.
- Maybe a **debugger**?

Note: Not all of this functionality will be in the first release!

The "Partsmart"

- **My bet is this:** If we make it easy to create Part Kits, then our *users* will fill in the gaps we leave by:
  - Making Kits for OSX and iOS and various Linux flavors (& maybe improving our Windows and Android Part Kits)
  - Translating them to lots of languages;
  - And maybe creating more complex parts like a rich text editor, HTML viewer (and editor?), database controls, etc.
  - I also hope we'll have some talented users who can make Parts that look *really* nice.
    - For some of us, "tinkering" with these will be fun, and the process will be easy, so maybe lots of people will try it?
  - This could be a big advantage over other Python GUIs.
    - Most make apps that look like they were written for Windows*XP*. (The others, like they weren't written for Windows at all.)
- To make the idea work, we need a really easy way for users to **find** & **download** these **Custom Part Kits**.
  - We'll address that with the "Partsmart" (Parts Market) – an **"app-store"-like service,** with descriptions, images, ratings and reviews, download (and payment) stats, etc.
    - Ideally, as a web service with the client built-into the DUITool.
    - That way, Kits can be installed immediately & uploaded directly.
  - We'll also encourage development by making it practical to pay for Parts Kits (and eBooks, tutorials and other stuff);
    - As well as report bugs, get help, discuss usage, etc.
- This might be a way to **monetize the project**
  - We can take a small cut from payments (like app stores).
    - It will only work if there are *lots* of Parts and lots of users. **I plan to "prime the pump"** by sponsoring a Part Kit contest. I could easily justify spending $5-10K on prizes if we end up with hundreds of new Parts and Part Kits available right away, to attract lots of new users!
  - We can also sell our own advanced part kits and eBooks.

Feature Summary

A Brief list of **PyDUIT's most important features**:

- Create apps for (and with) all common operating systems Windows/Mac/Linux (& Ras-Pi), Android & iOS (phones & tablets)
  - Build Desktop (mouse) *and* Mobile (multi-touch) apps
- Provide UI Part Kits "tuned" for each target (OS, app type)
- Facilitate creation of new and customized Parts and Kits,
  - and enable easy translation of all names, hints & help topics (including for traits & events!) to users' native languages
- Have an "app store" service for sharing/downloading parts
  - With hundreds, maybe thousands of parts & kits available, with ratings, reviews, bug reporting, discussions, updates, etc.
- Minimize learning curve – eliminate need to memorize names and provide instant help (in user's own language)
- Be usable by GUI designers and translators (i.e., non-programmers), without access to (risk to) the source code
- Enable apps to have ***any number of different UI's***
- Be a great tool for teaching Python to beginners:
  - Create nice looking, usable GUIs without complex OOP coding
  - Have Parts for writing games, media players, other fun stuff
- Highly productive tool for building real-world applications:
  - DUITool to facilitate reuse – menu structures, action lists, data fields, grid layouts, often reused forms (e.g. login dialog), etc.
  - Include parts for business apps – e.g. DB & web service clients
  - Clean separation of GUI spec from app logic for easier changes

No other Python GUI library can do these things – at least, not more than a few of them, and not very well*.

- I strongly suspect if we build this, it will be *very* popular. Hopefully enough to become the "tool of choice" for building GUI apps with Python, *especially* for non-native English speakers.

The question remains: **How much work will it take?**

How much work???

- I still can't answer this with any accuracy without ***much*** more detailed design work, but…

  It *feels* like something that could be built by a couple good programmers in a year or so, maybe even less, since:
  - Python will make some of this a lot easier than Delphi would;
  - We have *tons* of example code to study – Kivy's especially.
  - Breaking it down, it looks like:
    - The **Core** modules will need lots of little pieces:
      - Drawing functionality to work with SDL2, & TrueTypeFont class;
      - Event generating handlers for keyboard, mouse, surface, etc;
      - Timer, event-loop, clipboard, initialization, and… lots more;
      - DUI Language Processor, which Python might make reasonably easy.
    - **UI Parts** – probably 100-150 between Windows & Android?
      - Most components are fairly simple (although not all!), but we will need a *lot* of them. I'm hoping starting with building a few base classes that we can "mix" to build the others will make the process reasonably fast.
    - **DUITool** – Need to build with PyDUIT, so can't do in parallel.
      - Plan is to use this app as a detailed example of a real PyDUIT app, and make its code be available to tweak and learn from.
      - I think I could have the basic functionality (Part Kit menu, DUI code editing, app module interaction) working smoothly in a month or so.
    - **Partsmart** (app store) – Can be built completely independently
      - I'm hoping we can implement this as a web service, with the app part built-into the DUITool. But it could start as a simple web site if easier.
    - **Documentation** – Mostly API, beginner tutorials, etc;
      - It is crucial that this library have good documentation, especially for our Parts and Traits, where we will be setting a standard for everyone.
      - Advanced tutorials, detailed documentation as (monetizable) e-books?
    - **Testing** – especially integrated test code
      - If I'm right that this could become "the new standard", we need to code for maintainability, not prototyping speed, so this will add time.
- Adding it up, it still "seems" like 2000-4000 hrs or so…
  - Plus another 1000-2000+ hours for docs, QA, overhead, etc.
  - And whatever it takes to finish prototyping and designing it all.

(Preliminary) Conclusions

- As you can tell, I am excited about doing this project.
  - I think it has the potential to be *very* successful, and if so, will be a great stepping stone to even better things.
- It's the "even better things" I'm ***really*** excited about.
  - As I said in the introduction: Since retiring, I've come up with *lots* of potentially profitable ideas.
    - But few will ever be implemented if I have to do it all myself.
    - I need a team of smart, talented developers who can learn to work well with me, and together, without lots of bureaucracy.
    - I can then provide designs and guidance, and have those ideas be understood (and improved on, by triggering their ideas).
    - And then, get them implemented by a whole team, far more quickly than I possibly could have done them working alone.
- I think PyDUIT is a great "first project" for building this team, because financial success *doesn't* matter.
  - Worst case: Nobody cares, but we learn a lot, & have fun.
  - *Best* case: What we build turns out *really great*,
    - gets downloaded at least a million times in the first 3 months,
    - becomes the "tool of choice" for teaching Python & UI design,
    - gets nominated to replace tkinter in Python's standard library,😉
    - and we end up selling 150,000 copies of our $20 e-books.

    (Clearly, this best case is unlikely, but so is that worst case!)
  - Whatever happens, we have a low pressure, "room for experiments & mistakes" environment to build our team.
    - We'll have time to find the right tools and develop systems for tracking requirements & issues, work time, code revisions, etc, without "get it done before we run out of money" driving us.
- Longer term, I have a long list of projects to work on.
  - If this experiment works well, in a year or so we'll be done with this project, and deciding what's next.
  - Did I mention that I'm really excited? **Here goes!**