Thoughts Workspace

Thursday 31 December 2020 00:58

Table of Contents

30/12/2020 (Back to Top)

- You got Qt Designer installed! *(Finally.* More and more it seems that Windows is needlessly awkward about everything.)
- https://www.learnpyqt.com/ - have a look through that after you have your C++ spitting out JSON.
- Will need to extract data from the ThinkGear Connector packets and make my own JSON string or object creation so I can timestamp readings.
- Java talks to the TGC socket server - more straight forward to use but still need to complete refactoring.
- C++ talks to the Bluetooth device directly - faster but will need to parse serial packets

31/12/2020 (Back to Top)

What you said you'd do today:	1. gawk through the ThinkGear (TG) Serial Stream SDK 2. use that info to write code to extract only the data points I'll actually use (for now) 3. commit 4. introduce JSON formatting, save readings to textfile (new file created for every session) 5. commit and push
What you actually did:	• Lost track of time and rabbit-holed. • Firebase and Appwrite both seem exclusively aimed toward mobile apps. Unfortunate because Firebase actually seems the simplest platform for my purposes. • Amazon DynamoDB has native support for JSON - *very* good to know. Seems preferable over Amazon S3 • Amazon Amplify gets mentioned a lot with Dart, seems a great option but did they say during that Re:Invent session that Flutter support is barely in preview? Must check again: ○ https://virtual.awsevents.com/media/1_5pym8q5r • Found out you have access to Travis CI via education.github.com • Connected GitHub For Jira to your Jira project • Needed to be done before Travis CI could be connected to Jira • Rabbit-holed through docs.microsoft.com at break-neck pace • Created "OCHRe-Projects" GitHub organisation and transferred repo to it • Both Jira and Travis CI seemed like they'd prefer that. Guess I'll find out if that was wise • Read this page though before you do damage (done, nothing to worry about)
Next Day?	1. Have Visual Studio documentation on-hand while you re-create your C++ project from scratch 1. This is the first thing you must do and only thing until it works like it used to again a. I hate project config files 2. gawk through the ThinkGear (TG) Serial Stream SDK 3. use that info to write code to extract only the data points I'll actually use (for now) 4. commit 5. introduce JSON formatting, save readings to textfile (new file created for every session) 6. commit and push

01/01/2020 (Back to Top)

What you said you'd do today:	1. Have Visual Studio documentation on-hand while you re-create your C++ project from scratch 1. This is the first thing you must do and only thing until it works like it used to again a. I hate project config files 2. gawk through the ThinkGear (TG) Serial Stream SDK 3. use that info to write code to extract only the data points I'll actually use (for now) 4. commit 5. introduce JSON formatting, save readings to textfile (new file created for every session) 6. commit and push
What you actually did:	• Sorted Visual Studio and got the libraries and stuff properly added for once • A whole lotta rabbit holing • Using

Next Day?	1. gawk through the ThinkGear (TG) Serial Stream SDK 2. use that info to write code to extract only the data points I'll actually use (for now) 3. commit 4. introduce JSON formatting, save readings to textfile (new file created for every session) 5. commit and push

04/01/2020 (Back to Top)

What you said you'd do today:	1. gawk through the ThinkGear (TG) Serial Stream SDK 2. use that info to write code to extract only the data points I'll actually use (for now) 3. commit 4. introduce JSON formatting, save readings to textfile (new file created for every session) 5. commit and push
What you actually did:	Honestly, I don't even know where the last four days went. The only day I didn't sit down at my laptop all day to work was the 6th January - and that was purely because I was too angry/frustrated at myself for not being able to show good work for all the hours I'm putting in to think straight.
Next Day?	1.

07/01/2020 (Back to Top)

What you said you'd do today:	1. gawk through the ThinkGear (TG) Serial Stream SDK 2. use that info to write code to extract only the data points I'll actually use (for now) 3. commit 4. introduce JSON formatting, save readings to textfile (new file created for every session) 5. commit and push
What you actually did:	• Rabbit-holed *a lot* on cppreference.com docs • C++20 made the chrono library a lot handier, may soon be time to change from VS2019's default (C++11) • It literally took you over six hours to put 20 lines into a C++ class header file once you got going. • You're still proud of yourself. It's the most tangible progress you've made in one sitting in weeks. •
Next Day?	1. Finish the object class for the TG Protocol data types 1. Finish the header file. 2. Then the .cpp file. 3. Keep following the TG Communications (TGC) Protocol documentation as you do. 2. Start planning the video 1. Script

Organisation - Keeping a Work Diary

Monday 30 November 2020 16:50

There may be a few different reasons for keeping a work diary:

- To track the design process, particularly the timing of key decisions, findings, and information (ie. treat it as a medical record for the project). This might be useful in response to a design failure later down the track, or to improve processes at an end of project reflection session.

- To provide evidence of work completed if a timesheet is challenged.

- To remember how and when you did things.

- As a place to keep notes, hints, tips, and other things worth remembering.

- Because you have to (eg. institutional requirement, statutory requirement, or registration requirement).

From <https://learnonline.gmit.ie/course/view.php?id=99>

- Journaling to include specific times and date of activities

- Burndown charts, Gantt charts - same thing really, same purpose

- Set up an Agile Board - Trello? MS Planner? Jira? GitHub Projects?
Need help taking everything out of my head and onto paper (not necessarily physical paper)

ENGINEERING JOURNAL TEMPLATE

Date

- every page MUST have a date
- should always be at the top corner.

Tasks

- list of tasks for the day
- this serves as a way for you to collect your thought and set goals for the day/meeting

Reflection

- should reflect what you have done for today
- may include more if you happen to achieve a few more goals than listed in the "Tasks" section
- should highlight interesting findings, especially those unexpected

Issues:

Hardware:

1. outstanding issues on hardware development
2. numbered entries.
3. say "none" if absolutely no issue.
4. content MUST be informative and contain substance. Information like "chassis does not work" does not provide any useful information. Instead, information content will be something like: "Chassis lacks bracing… too much friction on the axles so the chassis is sacking in the middle…"

Software:

- outstanding issues on software development
- numbered entries.
- again, content MUST be informative… do not say things like “navigation is not working” … that is not useful!

Solutions

Hardware:

- if this is a solution which something happened today, it can be recorded in the reflection.
- If this is to resolve an issue happened before today, you should refer it by the date followed by the number, e.g.
  e.g. issue 1 recorded on June 1st.
  solution: 6/1-1) solution is to …. etc.

Software:

- Same as the hardware section.

From <https://galwaymayoinstitute-my.sharepoint.com/personal/g00320978_gmit_ie/Documents/5thYear/ProjectEngineering/Engineering%20Journal%20Template.docx>

#1 - Week 10

Monday 30 November 2020 15:34

□ Complete Section 1 of the Flutter & Dart Udemy Course

Currently on Video 4

☑ Take notes in OneNote to prove to yourself that you did it

W13 - Journal #2, 21/12/2020

Thursday 17 December 2020 17:14

Tasks

- list of tasks for the day
- this serves as a way for you to collect your thought and set goals for the day/meeting

Reflection

- should reflect what you have done for today
- may include more if you happen to achieve a few more goals than listed in the "Tasks" section
- should highlight interesting findings, especially those unexpected

Issues:

Hardware:

1. outstanding issues on hardware development
2. numbered entries.
3. say "none" if absolutely no issue.
4. content MUST be informative and contain substance. Information like "chassis does not work" does not provide any useful information. Instead, information content will be something like: "Chassis lacks bracing… too much friction on the axles so the chassis is sacking in the middle…"

Software:

- outstanding issues on software development
- numbered entries.
- again, content MUST be informative… do not say things like "navigation is not working" … that is not useful!

Solutions

Hardware:

- if this is a solution which something happened today, it can be recorded in the reflection.
- If this is to resolve an issue happened before today, you should refer it by the date followed by the number, e.g.

    e.g. issue 1 recorded on June 1st.
    solution: 6/1-1) solution is to …. etc.

Software:

- Same as the hardware section.

From <https://galwaymayoinstitute-my.sharepoint.com/personal/g00320978_gmit_ie/Documents/5thYear/ProjectEngineering/Engineering%20Journal%20Template.docx>

Week 12 w/ Michelle, 2pm Tuesday

Tuesday 15 December 2020 19:20

Michelle was a massive help in helping to take stock of what I already have, wish I recorded it to listen back or at least was able to type, talk and listen simultaneously. The focus of the interaction was to see what I can do with what I already have.

- Was very impressed with the C++ code she saw. She liked the work with the chrono library. *(demo talking point? Go find that refill pad)*
- Also true for the Java code since I re-wrote and refactored instead of copy-pasted it.
- So I have two native code programs currently able to read data from the headset, the Java one can already write to a plaintext file.
    - Shouldn't take too much hassle to do the same with the C++ program
    - Could use something like Qt to have a Python UI interface with the C++
    - or have Python (or whatever lang really) read from the text file instead
- She pointed out that I'm caught between *needing* to make a project for college and *wanting* to develop a fully-realised service, considering the massive scope of what I *want* to eventually achieve. Not something I'd have recognised by myself.
    - You know what? She's 100% right. Definitely need to filter out the unnecessary ideas - can revisit them if there's time at the very end or, more sensibly, after college.
- Right, so the project core:
    - Windows desktop app
    - Interface with your existing code to read from Thinkgear Connector
    - Save to plaintext for now
    - Simple GUI to display graph(s) of data metrics *(principally Attention tracking, obvs)*
- What about Dart and Flutter?
    - For once in your life, you managed to explain something clearly and succinctly enough that, not only did you not lose them, they were on board with the idea.
    - Yeah sure, definitely talk about it, its current capabilities and relevant whys in the demo but worry about it later - after you have an MVP
    - Probably easiest to add any additional features during the Flutter stage of development - not to mention compatibility for other platforms.
    - Also, have a look on the Issues tab of Flutter's GitHub repo. There's bound to be a workaround for that Jetbrains Toolbox issue
- What about my plans involving AWS?
    - What about them? You were talking with Michelle for over an hour and ye never even got to discussing that.
    - Besides, you should probably have a working Flutter app *first* so you actually have something to throw up there and work with. It's not like you have a concrete plan for the cloud side of things yet anyway.
- Anyways, going back to the core:
    - From your existing code, you already have half/most of your proposal description sorted, just need to create and interface the GUI component any which way you can for now.
- Cool, is that everything?
    - No, you're forgetting some things. Hopefully you'll remember.
- So what now?
    a. Continue on tomorrow. Throw together a (for now) quick and dirty agile board on Jira
        i. You had no problem with Jira in Ericsson, try not to get overwhelmed like you did at the weekend
    b. Then get your existing code (both langs) up and running again and suss out which to work with
    c. GUI time
- You don't need to show all the pieces working together (for xmas demo), you just need to show them.

Ideas & Research

Saturday 14 November 2020 16:30

Core components:

1. Read brainwaves
2. Data
    1. Save/store data locally
    2. Process data
3. Attention detection workflows
    1. Possible operant conditioning actions
        1. Flash screen?
        2. Toast/push/desktop/browser notification?
        3. Blacklist/whitelist websites?
        4. Block/allow audio/music?
        5. Pause videos?
4. Front end (GUI)

Journaling - Weekly Reports

Tuesday 17 November 2020 13:09

There may be a few different reasons for keeping a work diary:

- To track the design process, particularly the timing of key decisions, findings, and information (ie. treat it as a medical record for the project). This might be useful in response to a design failure later down the track, or to improve processes at an end of project reflection session.

- To provide evidence of work completed if a timesheet is challenged.

- To remember how and when you did things.

- As a place to keep notes, hints, tips, and other things worth remembering.

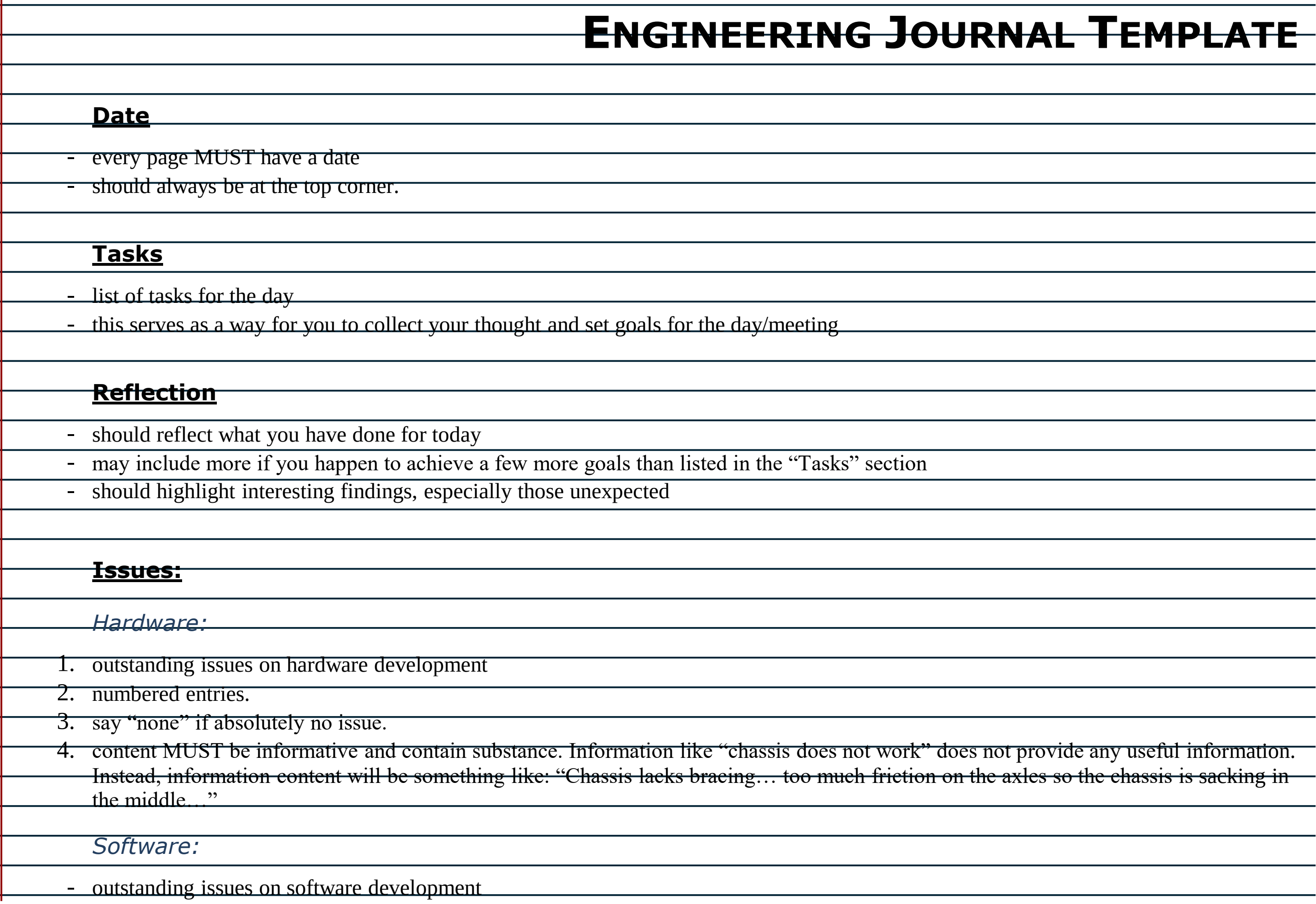
- Because you have to (eg. institutional requirement, statutory requirement, or registration requirement).

From <https://learnonline.gmit.ie/course/view.php?id=99>

- Journaling to include specific times and date of activities

- Burndown charts, Gantt charts - same thing really, same purpose

- Set up an Agile Board - Trello? MS Planner? Jira? GitHub Projects?
  Need help taking everything out of my head and onto paper (not necessarily physical paper)



ENGINEERING JOURNAL TEMPLATE

Date

- every page MUST have a date
- should always be at the top corner.

Tasks

- list of tasks for the day
- this serves as a way for you to collect your thought and set goals for the day/meeting

Reflection

- should reflect what you have done for today
- may include more if you happen to achieve a few more goals than listed in the "Tasks" section
- should highlight interesting findings, especially those unexpected

Issues:

Hardware:

1. outstanding issues on hardware development
2. numbered entries.
3. say "none" if absolutely no issue.
4. content MUST be informative and contain substance. Information like "chassis does not work" does not provide any useful information. Instead, information content will be something like: "Chassis lacks bracing… too much friction on the axles so the chassis is sacking in the middle…"

Software:

- outstanding issues on software development

- numbered entries.
- again, content MUST be informative… do not say things like “navigation is not working” … that is not useful!

Solutions

Hardware:

- if this is a solution which something happened today, it can be recorded in the reflection.
- If this is to resolve an issue happened before today, you should refer it by the date followed by the number, e.g.

    e.g. issue 1 recorded on June 1st.

    solution: 6/1-1) solution is to …. etc.

Software:

- Same as the hardware section.

From <https://galwaymayoinstitute-my.sharepoint.com/personal/g00320978_gmit_ie/Documents/5thYear/ProjectEngineering/Engineering%20Journal%20Template.docx>

//TODO:

Friday 11 December 2020 23:55

- □ Get clear idea of what and how data will be used and presented (data analysis)
- ☑ View Coursera courses to see what is relevant
    - No courses specific to AWS or Dart/Flutter, only GCP
- □ Do AWS Qwiklabs
    - A few look very good
- □ Currently about to start the Flutter & Dart Udemy course.
- □ Build basic Flutter app and interface with MWM2
    - Is Flutter fully installed?
- □ AWS expires in Jan - aws-cs-educate-form@amazon.com
- □ Setup GitHub repo
- □ Start with Google Authenticator app as UI base

Show Me What You Got

Thursday 31 December 2020 15:14

1.

Week 8 w/ Paul Lennon, 1pm Tuesday

Tuesday 17 November 2020 13:21

What you did last week
What you're doing this week
Any trouble? (Blockers)

Weekly Report: Have GitHub and Gantt Charts at the ready

- Journaling to include specific times and date of activities

- Burndown charts, Gantt charts - same thing really, same purpose

- Set up an Agile Board - Trello? MS Planner? Taiga? Jira? GitHub Projects?
  Need help taking everything out of my head and onto paper (not necessarily physical paper)

- Both MS Project (RDP into college lab PCs) and Lucid Chart can do Gantt charts

Went on a rant about how important project management/organisation/journaling with consistency is, but didn't say anything actually fucking helpful in the way of doing such. I know he means well, but it's hard not to feel like a goldfish reprimanded for its inability to fight sharks.

Week 8 w/ Michelle Lynch, 2pm Tuesday

Tuesday 17 November 2020 14:11

Showed FYP videos from last year to get an idea of what is expected.

- She showed Renan's project video, he modded his washing machine
  Maybe contact Renan and ask about his data analyses

- Stephen McIntyre's project (recorded)
  Facial recognition and eye detection/tracking on a RaspPi w/ camera.
  Block diagram looks pretty good.
  Great detail given about eye detection technical info.

- Armen's project (recorded). Kinect game built using Unity 3D and Visual Studio. Tracks bodily movements, game controlled via gesture recognition.
  Fast forwarded because some of the video went into minute details about code

- Mac's project (recorded). Open source VR headset compatible w/ SteamVR.
  You don't remember what quaternion signals are.
  Had problems with HDCP, got a different screen to workaround.
  Michelle: great example of technical detail.

- Ruairi Doherty's project (recorded). Remote robot. One of the MIDAS winners (
  NodeJs, AWS IoT Core, MQTT, 2x ESP32, livestreamed camera.
  Flex sensors, MPU6050 and push buttons on glove controller.

- Ask next week about that ML project video Michelle said she has.

Week 8 w/ Brian O'Shea, 1pm Friday

Friday 20 November 2020 13:19

Cloud based

Data sent directly to cloud to be processed

To train people? Make app available to other users

Have trainer user to analyse data?

- Creating tasks/goals for users?

Need to set goals, features, use cases for planned finished product:

- Cloud-based (optional local operation + storage later?)
- Web app and desktop (Windows, Linux, then MacOS) primary platforms
- Mobile platform secondary (**BOTH** Android & iOS)

Mondays 11am

Tuesdays @ 4pm, Thursdays @ 11 - 11:10am

Cloud Computing 1am - 12pm Fridays

Week 8 w/ Niall O'Keefe, 2pm Friday

Friday 20 November 2020 14:13

Basically the whole time looking at Mohammed Otaki's edge detection (python, OpenCV) code for road markings.

- Canny edge detection using Gaussian blurring, then Hough transform (why?)

Week 9 w/ Brian O'Shea, 1pm Friday

Friday 27 November 2020 13:13

You've been forgetting to jump in on Cloud Computing and SWwTest lectures.

No one has or will say anything to you. You want to do this.

Week 10 w/ Paul Lennon, 1pm Tuesday

Tuesday 1 December 2020 13:51

Week 10 w/ Michelle, 2pm Tuesday

Tuesday 1 December 2020 13:51

Achieving Academic/Technical Content:

- Pick an area(s) to dig deep into. Understand the maths and concept behind it.
    - Example she gave: say your project relies heavily on a Haar cascade function as part of your core code.
    - Master and provide technical insight behind it.
- Creating unique software/code and/or code and/or hardware.
    - The more you've changed/adapted your designs, the better for demonstration
- Analysing and/or evaluating subject content, demonstrating critical thinking skills

Demonstration of Understanding

- Demonstrating technologies/skills outside of specific programme content/laboratories
- Demoing insight into the academic & technical concepts engaged in the field of the project
- Demoing deep learning about core project academic/technical area(s)
- Demoing understanding of any code re-used, and with proper acknowledgement of sources
- Handling questions at project demonstration(s)
- Producing architectural block diagrams/flowcharts/timing diagrams/circuit diagrams to explain & communicate project elements
    - Include your scribbles and notes in your report, demonstration and presentation

Making The Project Video

Kevin Stratvert (YouTube) has a video on how to use the Photos Windows app

Could also use Movie Maker 10

Screencast-o-matic

Week 11 RIP Lindsey.

Tuesday 15 December 2020 19:20

Week 12 w/ Michelle, 2pm Tuesday

Tuesday 15 December 2020 19:20

Michelle was a massive help in helping to take stock of what I already have, wish I recorded it to listen back or at least was able to type, talk and listen simultaneously. The focus of the interaction was to see what I can do with what I already have.

- Was very impressed with the C++ code she saw. She liked the work with the chrono library. *(demo talking point? Go find that refill pad)*
- Also true for the Java code since I re-wrote and refactored instead of copy-pasted it.
- So I have two native code programs currently able to read data from the headset, the Java one can already write to a plaintext file.
    - Shouldn't take too much hassle to do the same with the C++ program
    - Could use something like Qt to have a Python UI interface with the C++
    - or have Python (or whatever lang really) read from the text file instead
- She pointed out that I'm caught between *needing* to make a project for college and *wanting* to develop a fully-realised service, considering the massive scope of what I *want* to eventually achieve. Not something I'd have recognised by myself.
    - You know what? She's 100% right. Definitely need to filter out the unnecessary ideas - can revisit them if there's time at the very end or, more sensibly, after college.
- Right, so the project core:
    - Windows desktop app
    - Interface with your existing code to read from Thinkgear Connector
    - Save to plaintext for now
    - Simple GUI to display graph(s) of data metrics *(principally Attention tracking, obvs)*
- What about Dart and Flutter?
    - For once in your life, you managed to explain something clearly and succinctly enough that, not only did you not lose them, they were on board with the idea.
    - Yeah sure, definitely talk about it, its current capabilities and relevant whys in the demo but worry about it later - after you have an MVP
    - Probably easiest to add any additional features during the Flutter stage of development - not to mention compatibility for other platforms.
    - Also, have a look on the Issues tab of Flutter's GitHub repo. There's bound to be a workaround for that Jetbrains Toolbox issue
- What about my plans involving AWS?
    - What about them? You were talking with Michelle for over an hour and ye never even got to discussing that.
    - Besides, you should probably have a working Flutter app *first* so you actually have something to throw up there and work with. It's not like you have a concrete plan for the cloud side of things yet anyway.
- Anyways, going back to the core:
    - From your existing code, you already have half/most of your proposal description sorted, just need to create and interface the GUI component any which way you can for now.
- Cool, is that everything?
    - No, you're forgetting some things. Hopefully you'll remember.
- So what now?
    a. Continue on tomorrow. Throw together a (for now) quick and dirty agile board on Jira
        i. You had no problem with Jira in Ericsson, try not to get overwhelmed like you did at the weekend
    b. Then get your existing code (both langs) up and running again and suss out which to work with
    c. GUI time
- You don't need to show all the pieces working together (for xmas demo), you just need to show them.

Miscellaneous - Perhaps Useful Later

Wednesday 30 December 2020 22:13

"May be worth checking out when it's time for the poster or video. macOS only."

[Phone Mockup From 1 Million Angles - Rotato📱](https://www.rotato.xyz/)

https://www.rotato.xyz/

Generates instant 4K PNG, MOV, MP4 from MOV, PNG, PSD + 100 more. Stop hunting for iPhone, Android, Laptop PSDs, even if you don't have Photoshop...

"Same deal as above, but simple and free."

[Threed.io - Generate custom 3D Device Mockups.](https://threed.io/)

https://threed.io/

Generate custom 3D Device Mockups in your Browser.

Flutter/Dart Setup

Friday 20 November 2020 14:48

- Installed Flutter SDK v1.22.4 (Stable branch) via Git clone (easier to update and/or switch release channels)

Flutter - Reactive Programming - Streams - BLoC

Wednesday, November 18, 2020 6:31 PM

"Useful background knowledge"

Source: https://www.didierboelens.com/2018/08/reactive-programming-streams-bloc/

Introduction to the notions of Streams, Bloc and Reactive Programming. Theory and practical examples.

Difficulty: *Intermediate*

Introduction

It took me quite a while to find a way to introduce the notions of **Reactive Programming**, **BLoC** and **Streams**.

As this is something that can make a drastic change to the way to architecture an application, I wanted a practical example that shows that:

- it is very possible not to use them but could sometime be much harder to code and less performant,
- the benefits of using them, but also
- the impacts of using them (positive and/or negative).

The practical example I made is a pseudo application which, in short, allows a user to view a list of movies from an online catalog, filter them by genre and release dates, mark/unmark them as favourites. Of course, everything is interactive, user actions can happen in different pages or inside a same one and have impacts on the visual aspects, real time.

This is an animation that shows this application.

Streams Application

As you landed into this page to get information about **Reactive Programming**, **BLoC** and **Streams**, I will first start with an introduction to them. Thereafter, I will show you how to implement and use them, in practice.

A complement of this article which gives some pratical use cases can be found following this link.

What is a Stream?

Introduction

In order to easily visualize the notion of **Stream**, simply consider a *pipe* with 2 ends, only one allowing to insert something into it. When you insert something into the pipe, it flows inside the pipe and goes out by the other end.

In Flutter,

- the pipe is called a **Stream**
- to control the *Stream*, we usually(*) use a **StreamController**
- to insert something into the *Stream*, the *StreamController* exposes the "*entrance*", called a *StreamSink*, accessible via the **sink** property
- the way out of the *Stream*, is exposed by the *StreamController* via the **stream** property

(*): I intentionally used the term "*usually*", since it is very possible not to use any *StreamController*. However, as you will read in this article, I will only make use of *StreamControllers*.

What can be conveyed by a Stream?

Everything and anything. From a value, an event, an object, a collection, a map, an error or even another Stream, any type of *data* may be conveyed by a Stream.

How do I know that something is conveyed by a Stream?

When you need to be notified that something is conveyed by a Stream, you simply need to **listen** to the *stream* property of the StreamController.

When you define a *listener*, you receive a **StreamSubscription object. This is via that** *StreamSubscription* object that you will be notified that something happens at the level of the *Stream*.

As soon as there is at least one **active** *listener*, the *Stream* starts generating **events** to notify the **active** *StreamSubscription object(s)* each time:

- some data goes out from the stream,
- when some error has been sent to the stream,
- when the stream is closed.

The *StreamSubscription* object also allows you to:

- stop listening,
- pause,
- resume.

Is a Stream only a simple pipe?

No, a *Stream* also allows to **process** the data that flows inside it before it goes out.

To control the processing of the data inside a *Stream*, we use a **StreamTransformer, which is nothing but**

- a function that “*captures*” the data that flows inside the *Stream*
- does something with the data
- the outcome of this transformation is also a *Stream*

You will directly understand from this statement that it is very possible to use several *StreamTransformers* in sequence.

A *StreamTransformer* may be used to do any type of processing, such as, e.g.:

- filtering: to filter the data based on any type of condition,
- regrouping: to regroup data,
- modification: to apply any type of modification to the data,
- inject data to other streams,
- buffering,
- processing: do any kind of action/operation based on the data,
- …

Types of Streams

There are 2 types of *Streams*.

Single-subscription Streams

This type of *Stream* only allows a **single** listener during the whole lifetime of that *Stream*.

It is not possible to listen twice on such *Stream*, even after the first subscription has been canceled.

Broadcast Streams

This second type of *Stream* allows **any number** of listeners.

It is possible to add a listener to a *Broadcast Stream* at any moment. The new listener will receive the events, as of the moment it starts listening to the *Stream*.

Basic Examples

Any type of data

This very first example shows a “*Single-subscription*” *Stream*, which simply prints the data which is input. As you may see the type of data does not matter.

```
import 'dart:async';

void main() {
//
// Initialize a "Single-Subscription" Stream controller
//
final StreamController ctrl = StreamController();

//
// Initialize a single listener which simply prints the data
// as soon as it receives it
//
final StreamSubscription subscription = ctrl.stream.listen((data) => print('$data'));

//
// We here add the data that will flow inside the stream
//
ctrl.sink.add('my name');
ctrl.sink.add(1234);
ctrl.sink.add({'a': 'element A', 'b': 'element B'});
ctrl.sink.add(123.45);

//
// We release the StreamController
//
ctrl.close();
}
```

view raw streams_1.dart hosted with ❤ by GitHub

StreamTransformer

This second example shows a “*Broadcast*” *Stream*, which conveys *integer* values and only prints the even numbers. To do so, we apply a *StreamTransformer* that filters (line #14) the values and only let the even numbers go through.

```
import 'dart:async';
```

```
void main() {
//
// Initialize a "Broadcast" Stream controller of integers
//
final StreamController<int> ctrl = StreamController<int>.broadcast();

//
// Initialize a single listener which filters out the odd numbers and
// only prints the even numbers
//
final StreamSubscription subscription = ctrl.stream
.where((value) => (value % 2 == 0))
.listen((value) => print('$value'));

//
// We here add the data that will flow inside the stream
//
for(int i=1; i<11; i++){
ctrl.sink.add(i);
}

//
// We release the StreamController
//
ctrl.close();
}
```

view raw streams_2.dart hosted with ❤ by GitHub

RxDart

Nowadays, the introduction to the *Streams* would no longer be complete if I would not mention the **RxDart Package.**

The **RxDart** package is an implementation for *Dart* of the ReactiveX API, which extends the original *Dart Streams* API to comply with the *ReactiveX* standards.

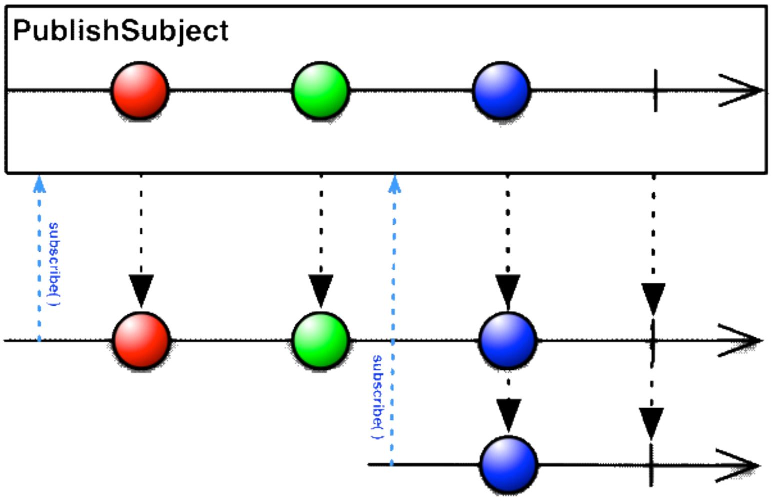
As it was not originally defined by Google, it uses a different vocabulary. The following table gives you the correlation between Dart and RxDart.

Dart	RxDart
Stream	Observable
StreamController	Subject

RxDart as I just said, **extends** the original *Dart Streams* API and offers 3 main variations of the *StreamController*:

PublishSubject

The **PublishSubject is a normal broadcast** *StreamController* with one exception: **stream** returns an **Observable rather than a Stream**.

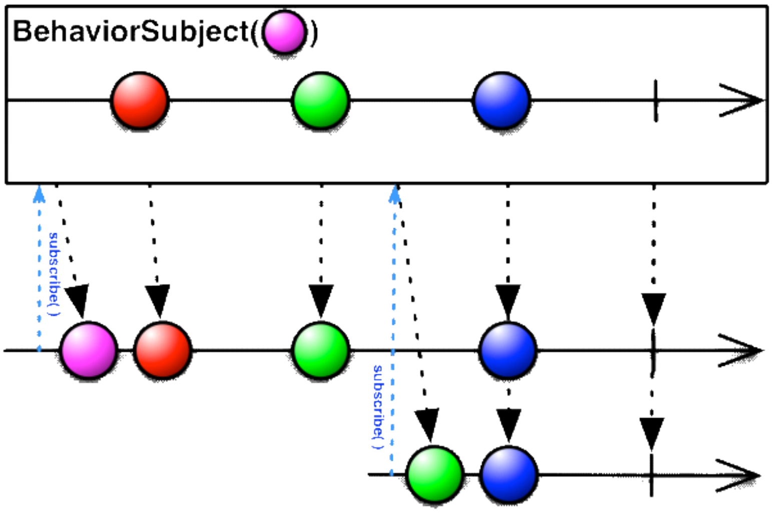


PublishSubject (c) ReactiveX.io

As you can see, *PublishSubject* sends to a *listener* only the *events* that are added to the *Stream* after the time of the *subscription*.

BehaviorSubject

The **BehaviorSubject is also a broadcast** *StreamController* which returns an **Observable** rather than a **Stream**.

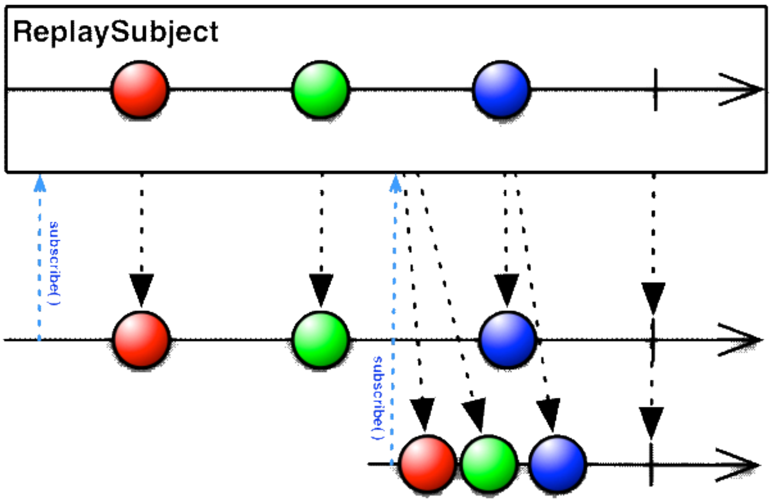


BehaviorSubject (c) ReactiveX.io

The main difference with a *PublishSubject* is that the *BehaviorSubject* also sends the very last *event* that was emitted to the *listener* that just subscribed.

ReplaySubject

The **ReplaySubject is also a broadcast** *StreamController* which returns an **Observable** rather than a **Stream**.



ReplaySubject (c) ReactiveX.io

A *ReplaySubject*, by default, sends *all the events* that were already emitted by the *Stream* to any new *listener* as the very first *events*.

Important note about the Resources

It is a very good practice to always release the resources which are no longer necessary.

This statement applies to:

- *StreamSubscription* - when you no longer need to listen to a *stream*, **cancel** the subscription;
- *StreamController* - when you no longer need a *StreamController*, **close** it;
- the same applies to *RxDart Subjects*, when you no longer need a *BehaviourSubject*, a *PublishSubject*…, **close** it.

How to build a Widget based on the data that goes out a Stream?

Flutter offers a very convenient StatefulWidget, called **StreamBuilder.**

A *StreamBuilder* listens to a *Stream* and, each time some data goes out that *Stream*, it automatically *rebuilds*, invoking its *builder* callback.

This is how to use the *StreamBuilder*:

```
StreamBuilder<T>(
    key: ...optional, the unique ID of this Widget...
    stream: ...the stream to listen to...
    initialData: ...any initial data, in case the stream would initially be empty...
    builder: (BuildContext context, AsyncSnapshot<T> snapshot){
        if (snapshot.hasData){
            return ...the Widget to be built based on snapshot.data
        }
        return ...the Widget to be built if no data is available
    },
)
```

The following example mimics the default "*counter*" application, but uses a *Stream* and no longer any *setState*.

```
import 'dart:async';
import 'package:flutter/material.dart';

class CounterPage extends StatefulWidget {
@override
_CounterPageState createState() => _CounterPageState();
}

class _CounterPageState extends State<CounterPage> {
int _counter = 0;
final StreamController<int> _streamController = StreamController<int>();

@override
void dispose(){
_streamController.close();
super.dispose();
}

@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(title: Text('Stream version of the Counter App')),
body: Center(
child: StreamBuilder<int>(
stream: _streamController.stream,
initialData: _counter,
builder: (BuildContext context, AsyncSnapshot<int> snapshot){
return Text('You hit me: ${snapshot.data} times');
}
),
),
floatingActionButton: FloatingActionButton(
child: const Icon(Icons.add),
onPressed: (){
_streamController.sink.add(++_counter);
},
),
);
}
```

```
    }
```

view raw streams_3.dart hosted with ❤ by GitHub

Explanation and comments:

- Lines #24-30: we are listening to a the *stream*, and each time a new value goes out this stream, we update the *Text* with that value;
- Line #35: when we hit the *FloatingActionButton*, we increment the counter and send it to the *Stream*, via the *sink*; the fact of injecting a value in the *stream* causes the *StreamBuilder* that listens to it to rebuild and "*refresh*" the counter;
- We no longer need the notion of *State*, everything is taken on board via the *Stream*;
- This is a big improvement since the fact of calling the *setState()* method, forces the **whole** Widget (and any sub widgets) to rebuild. Here, **ONLY** the *StreamBuilder* is rebuilt (and of course its children widgets);
- The only reason why we are still using a *StatefulWidget* for the page, is simply because we need to release the *StreamController*, via the *dispose* method, line #15;

What is Reactive Programming?

Reactive programming is programming with asynchronous data streams.

In other words, everything from an event (e.g. tap), changes on a variable, messages, … to build requests, everything that may change or happen will be conveyed, triggered by a data stream.

In clear, all this means that, with reactive programming, the application:

- becomes **asynchronous**,
- is architectured around the notion of **Streams** and **listeners**,
- when something happens somewhere (an event, a change of a variable …) a notification is sent to a *Stream*,
- if "*somebody*" listens to that *Stream*, it will be notified and will take appropriate action(s), **whatever its location in the application**.

There is no longer any tight coupling between components.

In short, when a Widget sends something to a *Stream*, that Widget **does no longer need to know**:

- what is going to happen next,
- who might use this information (no one, one or several Widgets…)
- where this information might be used (nowhere, same screen, another one, several ones…),
- when this information might be used (almost directly, after several seconds, never…).

…that Widget does only care about its own business, that's all !!

At first glance, reading this, this may seem to lead to a "*no-control*" of the application but, as we will see, this is the contrary. It gives you:

- the opportunity to build parts of the application only responsible for specific activities,
- to easily mock some components' behavior to allow more complete tests coverage,
- to easily reuse components (somewhere else in the application or in another application),
- to redesign the application and be able to move components from one place to another without too much refactoring,
- …

We will see the advantages shortly… but before I need to introduce the last topic: the **BLoC Pattern**.

The BLoC Pattern

The **BLoC Pattern** has been designed by *Paolo Soares* and *Cong Hui*, from Google and first presented during the *DartConf 2018* (January 23-24, 2018). See the video on YouTube.

BLoC stands for **B**usiness **Lo**gic **C**omponent.

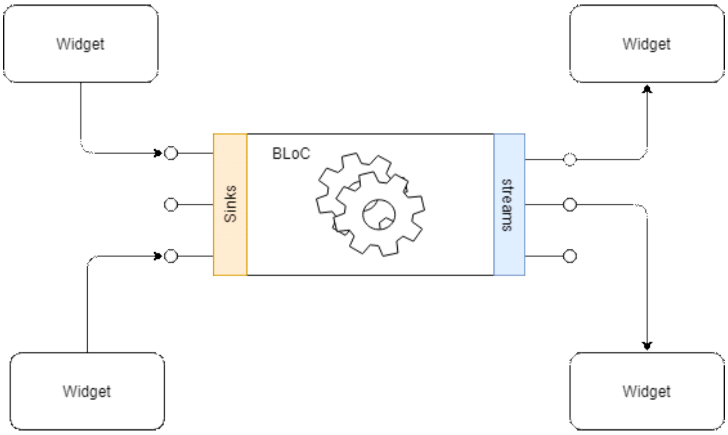
In short, the **Business Logic** needs to:

- be moved to one or several *BLoC* s,
- be removed as much as possible from the Presentation Layer. In other words, UI components should only worry about the UI things and not about the business,
- rely on *exclusive* use of *Streams* for both input (*Sink*) and output (*stream*),
- remain *platform independent*,
- remain *environment independent*.

In fact, the *BLoC* pattern was initially conceived to allow to reuse the very same code independently of the platform: web application, mobile application, back-end.

What does it actually mean?

The *BLoC* pattern makes use of the notions that we just discussed above: *Streams*.



- Widgets send *events* to the BLoC via *Sinks*,
- Widgets are notified by the BLoC via *streams*,
- the business logic which is implemented by the BLoC is none of their concern.

From this statement, we can directly see a **huge** benefit.

Thanks to the decoupling of the Business Logic from the UI:

- we may change the Business Logic at any time, with a minimal impact on the application,
- we may change the UI without any impact on the Business Logic,
- it is now much easier to test the Business Logic.

How to apply this BLoC Pattern to the Counter Application sample?

Applying the BLoC pattern to this Counter Application might seem to be an overkill but let me first show you…

```
void main() => runApp(new MyApp());

class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return new MaterialApp(
title: 'Streams Demo',
theme: new ThemeData(
primarySwatch: Colors.blue,
),
home: BlocProvider<IncrementBloc>(
bloc: IncrementBloc(),
child: CounterPage(),
),
);
}
}

class CounterPage extends StatelessWidget {
@override
Widget build(BuildContext context) {
final IncrementBloc bloc = BlocProvider.of<IncrementBloc>(context);

return Scaffold(
appBar: AppBar(title: Text('Stream version of the Counter App')),
body: Center(
child: StreamBuilder<int>(
stream: bloc.outCounter,
initialData: 0,
builder: (BuildContext context, AsyncSnapshot<int> snapshot){
return Text('You hit me: ${snapshot.data} times');
}
),
),
floatingActionButton: FloatingActionButton(
child: const Icon(Icons.add),
onPressed: (){
bloc.incrementCounter.add(null);
},
),
);
}
}

class IncrementBloc implements BlocBase {
int _counter;

//
// Stream to handle the counter
//
StreamController<int> _counterController = StreamController<int>();
StreamSink<int> get _inAdd => _counterController.sink;
Stream<int> get outCounter => _counterController.stream;

//
// Stream to handle the action on the counter
//
StreamController _actionController = StreamController();
StreamSink get incrementCounter => _actionController.sink;

//
// Constructor
//
IncrementBloc(){
_counter = 0;
_actionController.stream
.listen(_handleLogic);
}

void dispose(){
_actionController.close();
_counterController.close();
}

void _handleLogic(data){
_counter = _counter + 1;
_inAdd.add(_counter);
}
}
```

view raw streams_4.dart hosted with ❤ by GitHub

I already hear you saying "*wow… why all this? Is this all necessary?*".

First, the separation of responsibilities

If you check the *CounterPage* (lines 21 - 45), there is *absolutely* not any business logic inside it.

This page is now only responsible for:

- displaying the counter, which is now only refreshed when necessary (even without the page having to know it)
- providing a button which when hit, requests an action to be performed on the counter

Also, the **whole** the Business Logic is centralized in one single class "*IncrementBloc*".

If now, you need to change the business logic, you simply need to update the method *_handleLogic* (lines 77-80). Maybe the new business logic will request to do very complex things… The CounterPage will never know about it and this is very good!

Second, testability

It is now much easier to test the business logic.

No need anymore to test the business logic via the user interface. Only the *IncrementBloc* class needs to be tested.

Third, freedom to organize the layout

Thanks to the use of *Streams*, you can now organize the layout independently of the business logic.

Any action could be launched from any place in the application: simply call to the *.incrementCounter* sink.

You may display the counter anywhere, in any page, simply listen to the *.outCounter* stream.

Fourth, reduction of the number of “build"s

The fact of not using *setState()* but *StreamBuilder*, drastically reduces the amount of “*build*“s to only the ones which are required.

From a performance perspective, this is a huge improvement.

There is only 1 constraint… accessibility of the BLoC

In order to have all this working, the BLoC needs to be accessible.

There exists several ways of making it accessible:

- via a *global* Singleton
  This way is very possible but not really recommended. Also as there is no class *destructor* in Dart, you will never be able to release the resources properly.
- as a *local* instance
  You may instantiate a local instance of a BLoC. Under some circumstances, this solution perfectly fits some needs. In this case, you should always consider initializing inside a *StatefulWidget* so that you can take profit of the *dispose()* method to free it up.
- provided by an *ancestor*
  The most common way of making it accessible is via an **ancestor** Widget, implemented as a *StatefulWidget*.
  The following code shows a sample of a *generic* **BlocProvider**.

```
// Generic Interface for all BLoCs
abstract class BlocBase {
void dispose();
}

// Generic BLoC provider
class BlocProvider<T extends BlocBase> extends StatefulWidget {
BlocProvider({
Key key,
@required this.child,
@required this.bloc,
}): super(key: key);

final T bloc;
final Widget child;

@override
_BlocProviderState<T> createState() => _BlocProviderState<T>();

static T of<T extends BlocBase>(BuildContext context){
final type = _typeOf<BlocProvider<T>>();
BlocProvider<T> provider = context.ancestorWidgetOfExactType(type);
return provider.bloc;
}

static Type _typeOf<T>() => T;
}

class _BlocProviderState<T> extends State<BlocProvider<BlocBase>>{
@override
void dispose(){
widget.bloc.dispose();
super.dispose();
}

@override
Widget build(BuildContext context){
return widget.child;
}
}
```

view raw streams_5.dart hosted with ❤ by GitHub

Some explanations on this generic BlocProvider

First, how to use it as a *provider*?

If you have a look at the sample code “*streams_4.dart*", you will see the following lines of codes (lines #12-15)

```
   home: BlocProvider<IncrementBloc>(
          bloc: IncrementBloc(),
          child: CounterPage(),
        ),
```

With these lines, we simply instantiate a new *BlocProvider* which will handle a *IncrementBloc*, and will render the *CounterPage* as a child.

From that moment on, any widget **part of the sub-tree, starting at BlocProvider** will be able to get access to the *IncrementBloc*, via the following line:

```
IncrementBloc bloc = BlocProvider.of<IncrementBloc>(context);
```

Could we have multiple BLoCs?

Of course and this is highly advisable. Recommendations are:

- (if there is any business logic) one BLoC on top of each page,
- why not an ApplicationBloc to handle the Application State?
- each “*complex enough component*” has a corresponding BLoC.

The following sample code shows an *ApplicationBloc* on top of the whole application, then the *IncrementBloc* on top of the *CounterPage*.

The sample also shows how to retrieve both blocs.

```
void main() => runApp(
BlocProvider<ApplicationBloc>(
bloc: ApplicationBloc(),
child: MyApp(),
)
);

class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context){
return MaterialApp(
title: 'Streams Demo',
home: BlocProvider<IncrementBloc>(
bloc: IncrementBloc(),
child: CounterPage(),
),
);
}
}

class CounterPage extends StatelessWidget {
@override
Widget build(BuildContext context){
final IncrementBloc counterBloc = BlocProvider.of<IncrementBloc>(context);
final ApplicationBloc appBloc = BlocProvider.of<ApplicationBloc>(context);

...
}
}
```

view raw streams_6.dart hosted with ❤ by GitHub

Why not using an InheritedWidget?

In most articles related to BLoC, you will see the implementation of the *Provider* as an *InheritedWidget*.

Of course, nothing prevents this type of implementation. However,

- an *InheritedWidget* does not provide any *dispose* method and, remember, it is a very good practice to always release the resources when they are no longer needed;
- of course, nothing would prevent you from wrapping the *InheritedWidget* inside another *StatefulWidget*, but then, what is the added value of using an *InheritedWidget*?
- finally, the use of an *InheritedWidget* often leads to *side effects* if not controlled (see *Reminder on InheritedWidget*, below).

These 3 bullets explain the choice I made to implement the *generic* **BlocProvider** as a *StatefulWidget*, so that I am able to release the resources, when this widget is disposed.

Flutter is not able to instantiate a generic type

As unfortunately Flutter is not able to instantiate a generic type, we have to pass the instance of the BLoC to the *BlocProvider*. To enforce the implementation of the *dispose()* method in each BLoC, all BLoCs have to implement the *BlocBase* interface.

Reminder on InheritedWidget

When we are using an *InheritedWidget* and are invoking the *context.inheritFromWidgetOfExactType(…)* method to get the nearest widget of the given type, this method invocation **automatically** registers this "*context*" (= *BuildContext*) to the list of ones which **will be rebuilt** each time a change applies to the *InheritedWidget* subclass or to **one of its ancestors**.

Please note that, in order to be fully correct, the problem related to the *InheritedWidget* as I have just explained only occurs when we combine the InheritedWidget with a StatefulWidget. When you simply use an *InheritedWidget* without a *State*, the issue does not happen. But… I will come back on this remark in a next article.

The type of Widget (Stateful or Stateless) linked to the *BuildContext*, does not matter.

Personal note on BLoC

Third rule related to *BLoC* says: "*rely on exclusive use of Streams for both input (Sink) and output (stream)*".

My personal experience relativises this statement a little bit… Let me explain.

At first, the BLoC pattern was conceived to share the very same code across platforms (AngularDart, …) and, in this perspecti ve, that statement makes full sense.

However, if you only intend to develop a Flutter application, this is, based on my humble experience, a little bit overkill.

If we stick to the statement, no *getter* or *setter* are possible, only *sinks* and *streams*. The drawback is "all this is **asynchronous**".

Let's take 2 samples to illustrate the drawbacks:

- you need to retrieve some data from the BLoC in order to use this data as input of a page that should display these parameters right away (for example, think of a *parameters* page), if we had to rely on *Streams*, this makes the *build* of the page asynchronous (which is complex). Sample code to make it work via *Streams* could be like the following one… Ugly isn't it.

```
class FiltersPage extends StatefulWidget {
@override
FiltersPageState createState() => FiltersPageState();
}

class FiltersPageState extends State<FiltersPage> {
MovieCatalogBloc _movieBloc;
double _minReleaseDate;
double _maxReleaseDate;
MovieGenre _movieGenre;
bool _isInit = false;

@override
void didChangeDependencies() {
super.didChangeDependencies();

// As the context of not yet available at initState() level,
// if not yet initialized, we get the list of the
// filter parameters
if (_isInit == false){
_movieBloc = BlocProvider.of<MovieCatalogBloc>(context);
_getFilterParameters();
}
}

@override
Widget build(BuildContext context) {
return _isInit == false
? Container()
: Scaffold(
...
);
}

///
/// Very tricky.
///
/// As we want to be 100% BLoC compliant, we need to retrieve
/// everything from the BLoCs, using Streams...
///
/// This is ugly but to be considered as a study case.
///
void _getFilterParameters() {
StreamSubscription subscriptionFilters;

subscriptionFilters = _movieBloc.outFilters.listen((MovieFilters filters) {
_minReleaseDate = filters.minReleaseDate.toDouble();
_maxReleaseDate = filters.maxReleaseDate.toDouble();

// Simply to make sure the subscriptions are released
subscriptionFilters.cancel();

// Now that we have all parameters, we may build the actual page
if (mounted){
setState((){
_isInit = true;
});
}
});
});
}
}
```

view raw streams_7.dart hosted with ❤ by GitHub

- at the BLoC level, you also need to convert a “fake” injection of some data to trigger the provision of the data you expect to receive via a stream. Sample code to make this work could be:

```
class ApplicationBloc implements BlocBase {
///
/// Synchronous Stream to handle the provision of the movie genres
///
StreamController<List<MovieGenre>> _syncController = StreamController<List<MovieGenre>>.broadcast();
Stream<List<MovieGenre>> get outMovieGenres => _syncController.stream;

///
/// Stream to handle a fake command to trigger the provision of the list of MovieGenres via a Stream
///
StreamController<List<MovieGenre>> _cmdController = StreamController<List<MovieGenre>>.broadcast();
StreamSink get getMovieGenres => _cmdController.sink;

ApplicationBloc() {
//
// If we receive any data via this sink, we simply provide the list of MovieGenre to the output stream
//
_cmdController.stream.listen((_){
_syncController.sink.add(UnmodifiableListView<MovieGenre>(_genresList.genres));
});
}

void dispose(){
_syncController.close();
_cmdController.close();
}

MovieGenresList _genresList;
}

// Example of external call
```

```
BlocProvider.of<ApplicationBloc>(context).getMovieGenres.add(null);
```

view raw streams_8.dart hosted with ❤ by GitHub

I don't know your opinion, but personally, **if I don't have any constraints related to code porting/sharing**, I find this too heavy and I would rather use regular **getters/setters** when needed and use the **Streams/Sinks** to keep the separation of responsibilities and broadcast the information where needed, which is **awesome**.

It is now time to see all this in practice…

As mentioned in the beginning of this article, I built a pseudo-application to show how to use all these notions. The full source code can be found on Github.

Please be indulgent since this code is far from being perfect and could be much better and/or better architectured but the only objective is simply to show you how all this works.

As the source code is documented a lot, I will only explain the main principles.

Source of the movie catalog

I am using the free TMDB API to fetch the list of all movies, as well as the posters, rating and descriptions.

In order to be able to run this sample application, you will need to register and obtain the API key (**which is totally free**), then put your API key in the file "*/api/tmdb_api.dart*", line #15.

Architecture of the application

This application uses:

- 3 main BLoCs:
  - *ApplicationBloc* (on top of everything), responsible for delivering the list of all movie genres;
  - *FavoriteBloc* (just underneath), responsible for handling the notion of "*Favorites*";
  - *MovieCatalogBloc* (on top of the 2 main pages), responsible for delivering the list of movies, based on filters;
- 6 pages:
  - *HomePage*: landing page that allows the navigation to the 3 sub-pages;
  - *ListPage*: page that lists the movies as a GridView, allows filtering, favorites selection, access to the Favorites and display of the Movie details in a sub-sequent page;
  - *ListOnePage*: similar to *ListPage* but the list of movies is displayed as a horizontal list and the details, underneath;
  - *FavoritesPage*: page that lists the favorites and allows the deselection of any favorites;
  - *Filters*: *EndDrawer* that allows the definition of filters: genres and min/max release dates. This page is called from *ListPage* or *ListOnePage*;
  - *Details*: page only invoked by *ListPage* to show the details of a movie but also to allow the selection/unselection of the movie as a favorite;
- 1 sub BLoC:
  - *FavoriteMovieBloc*, linked to a *MovieCardWidget* or *MovieDetailsWidget* to handle the selection/unselection of a movie as a favorite
- 5 main Widgets:
  - *FavoriteButton*: widget responsible for displaying the number of favorites, real-time, and redirecting to the *FavoritesPage* when pressed;
  - *FavoriteWidget*: widget responsible for displaying the details of one favorite movie and allow its unselection;
  - *FiltersSummary*: widget responsible for displaying the filters currently defined;
  - *MovieCardWidget*: widget responsible for displaying one single movie as a card, with the movie poster, rating and name, as well as a icon to indicate the selection of that particular movie as a favorite;
  - *MovieDetailsWidget*: widget responsible for displaying the details related to a particular movie and to allow its selection/unselection as a favorite.

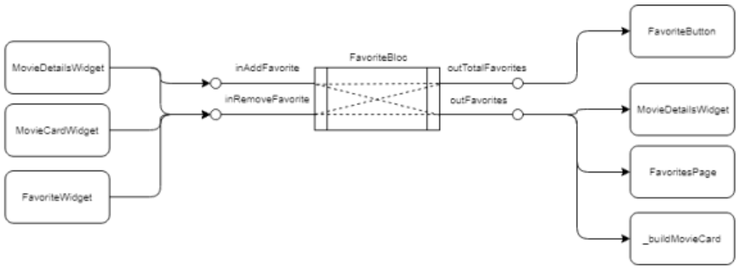
Orchestration of the different BLoCs / Streams

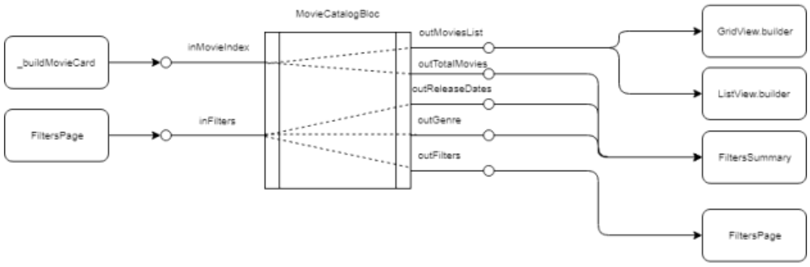
The following diagram shows how the main 3 BLoCs are used:

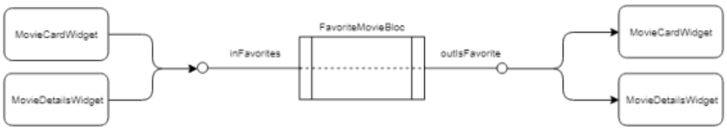
- on the left side of a BLoC, which components invoke the *Sink*
- on the right side, which components listen to the *stream*

As an example, when the *MovieDetailsWidget* invokes the *inAddFavorite Sink*, 2 *streams* are triggered:

- *outTotalFavorites* stream forces a rebuild of *FavoriteButton*, and
- *outFavorites* stream
  - forces a rebuild of *MovieDetailsWidget* (the "favorite" icon)
  - forces a rebuild of*_buildMovieCard* (the "favorite" icon)
  - is used to build each *MovieDetailsWidget*







Observations

Most of the Widgets and Pages are *StatelessWidgets*, which means that:

- the *setState()* which forces to rebuild is almost never used. Exceptions are:
    - in the *ListOnePage* when a user clicks a MovieCard, to refresh the MovieDetailsWidget. This could also have been driven by a *stream*…
    - in the *FiltersPage* to allow the user to change the filters before accepting them, via Sink.
- the application does not use any *InheritedWidget*
- the application is almost 100% *BLoCs/Streams* driven which means that most of the Widgets are independent of each other and of their location in the application
  A practical example is the *FavoriteButton* which displays the number of selected favorites in a badge. The application counts 3 instances of this *FavoriteButton*, each of them, displayed in 3 different pages.

Display of the list of movies (explanation of the trick to display an infinite list)

To display the list of movies that meet the filters criteria, we use either a GridView.builder (*ListPage*) or a ListView.builder (*ListOnePage*) as an infinite scroll list.

Movies are fetched using the *TMDB API* in pages of 20 movies at a time.

As a reminder, both *GridView.builder* and *ListView.builder* take as input an *itemCount* which, if provided, tells the number of items to be displayed. The *itemBuilder* is called with *index* varying from 0 up to *itemCount - 1*.

As you will see in the code, I arbitrary provide the *GridView.builder* with an additional *30 more*. The rationale is that in this example, we are manipulating a presumed infinite number of items (*which is not totally true but who cares for this example*). This will force the *GridView.builder* to request "*up to 30 additional*" items to be displayed.

Also, both *GridView.builder* and *ListView.builder* only call the *itemBuilder* when they consider that a certain item (*index*) has to be rendered in the viewport.

This *MovieCatalogBloc.outMoviesList* returns a *List<MovieCard>*, which is iterated in order to build each Movie Card. The very first time, this *List<MovieCard>* is empty but thanks to the *itemCount: …+30*, we fool the system which will ask to render 30 non-existing items via the *_buildMovieCard(…)*.

As you will see in the code, this routine makes a weird call to the Sink:

```
// Notify the MovieCatalogBloc that we are rendering the MovieCard[index]
    movieBloc.inMovieIndex.add(index);
```

This call tells the *MovieCatalogBloc* that we want to render the MovieCard[index].

Then the *_buildMovieCard(…)* goes on validating that data related to the MovieCard[index] exists. If yes, the latter is rendered, otherwise a CircularProgressIndicator is displayed.

The call to *MovieCatalogBloc.inMovieIndex.add(index)* is listened by a *StreamSubscription* that translates the *index* to a certain *pageIndex* number (one page counts up to 20 movies). If the corresponding page has not yet been fetched from the *TMDB API*, a call to the API is made. Once the page has been fetched, the new list of all fetched movies is sent to the *_moviesController*. As that *stream* (= *movieBloc.outMoviesList*) is listened by the *GridView.builder*, the latter requests to *rebuild* the corresponding MovieCard. As we now have the data, we may render it.

Credits and additional links

Images that describe the PublishSubject, BehaviorSubject and ReplaySubject were published by ReactiveX.

Some other interesting articles worth reading:

- Fundamentals of Dart Streams [Thomas Burkhart]
- rx_command package [Thomas Burkhart]
- Build reactive mobile apps in Flutter - companion article [Filip Hracek]
- Flutter with Streams and RxDart [Brian Egan]

Conclusion

Very long article but there is still much more to say about it as it is more than obvious to me that this is the way to go forward with the development of a Flutter application. It provides so much flexibility.

Stay tuned for new articles, soon. Happy coding.

This article is also available on Medium - Flutter Community

This article has also been translated in Chinese on https://www.jianshu.com/p/e7e1bced6890.

PWA vs Hybrid App vs Native: Choose the Right Mobile App

Friday, November 20, 2020 6:22 PM

Source: https://www.agileinfoways.com/blog/quick-tips-for-choosing-the-right-mobile-app-pwa-vs-hybrid-app-vs-native-app/

The number of mobile apps is increasing in the Google play store with over 2.6 million. Apple app store remains the second-largest store with 1.8 million apps available. Additionally, there are 4.1 billion internet users that use the internet on earth. We relate our daily life to all these statistics. And recently, there has been a fuss about PWA Vs Hybrid Vs Native going around.

Don't wait long, jump from building a website to the realm of building a mobile application. You can make basic level choices that will change everything. Build the right app for your mobile users, review, and compare possible solutions with choosing one of the below platforms.

- Progressive Web app (PWA)
- Hybrid App
- Native App

In this article, we'll show advanced improvement to utilize Hybrid applications, Progressive Web Application, and later Native Script. It's a cross-stage instrument that is progressively helpful than other platforms to build a useful app.

Here, we try to compare all platforms and find the best-suited platform for the next mobile app. Note that while building more than one app, create a component collection and share it with your team to make a component like Lego which is available free.

A mobile app development company creates applications in all three sections. It shows an excellent way to deal with pursuing is a standout amongst the most widely recognized inquiries that we run over. Everyone of three accompanies their arrangement and points of interest that hindrances the business bet after relying upon their end objectives.

You'll find several ways to create a mobile presence and let your user's access content on mobile. The responsive design will allow users to collect via the website. While the native mobile apps can download from an app store and add it to the home screen. Here, you require a portable application but don't know yet where to start from?

Numerous variables influence your versatile technique, for example, your group's advancement aptitudes, required gadget usefulness, the significance of security, disconnected ability, interoperability, and so forth., that must be considered. Screens are little, applications are enormous, and life as we probably are aware it is on its head once more. In a world that is progressively social and open, versatile applications assume an indispensable job and have changed the emphasis from what's on the Web, to the applications on our cell phone. Versatile applications are never again an alternative; they're a goal. At last, it's not only an issue of what your app will do, yet how you'll get it there.

Mistake to avoid while developing a mobile app

Let's review the best solution among these applications…

Progressive Web App

Progressive Web App isn't a real app but an extension to a website. Just like any website, PWA can be installed on a server and distributed using URLs. Local gadget highlights upheld by present-day internet browsers like camera, sound account, video catch is the limit for such applications. Administration Workers permit to reserve the site on the gadget and give a symbol to the bookmark made on the device. PWA lacks support for a few devices-based features which are native or hybrid providers.

Check: ultimate guide to Progressive Web App

Progressive Web Development Company could work as a blend of a versatile application and a site page. It pulls the best components of both and makes a fresh new experience for the user. It makes utilization of a portion of the advanced web abilities to convey an application like work to its clients. PWA solves the issue of websites not adapting to mobile applications and apps taking too long to load.

Don't miss: Things to know while building PWA

Based on smashing ideas estimate PWA statistics, there is:

- 20% increase in mobile sales and revenue in the market
- 8% increase in recovered shopping carts
- 50% increase in customer engagement
- 33% increase in maintenance and development

Let us go through some benefits of PWA:

- Fast Installation
- Platform and Device skepticism
- Enhanced with Security
- Better Performance
- No issue with updating

Tools generally used while developing PWA

The quickest method to make a PWA is to utilize PWABuilder and rapidly assemble an administration specialist for disconnected usefulness, which works by pulling and serving the " offline.html" from your web server at whatever point clients lose web network. You can likewise present your PWA to the application store for Android and iOS gadgets. Know the crazy trends in the PWA industry.

On the off chance that you as of now have involvement with substance the board frameworks yet are ignorant regarding creating web applications, at that point, you can get the essentials down through Google Developers, a library of assets that can enable you to figure out how to code. At last, Knockout and Webpack are such tools used for Progressive Web Application, and GitHub is a network-driven site that keeps up stores of undertakings. It covers a full scope of programming-related subjects, including JavaScript and PWA, benefit laborers. You can discover the PWA.rocks and Webpack vaults inside the stage.

How one can make money from free apps?

Hybrid App

Hybrid applications are fast as compared to other app platforms. Due to its 90% of sharing code, it is cheaper to develop, and a 10% advantage is taken by Hybrid to go with some functionalities. You can use standard web technologies such as Angular, JavaScript, HTML, and CSS. It saves time on development if your team isn't as familiar with Java, Objective-C, or Swift. Hybrid apps are native apps only because they can download from the platform's app store like a native app. It can get access to all the native platform features. It can have performance close to a native app.

It ensures the pace of web development along with the customized user experience that comes through native mobile app development. And once created, you can publish your hybrid mobile app in the Apple, Google, and Windows App Stores. Hybrid Mobile Applications, necessarily, fall amidst the local and web application range. They show the client encounter properties of both Native and Web, giving an assortment of excellent benefits like:

- Reduce development cost across multiple platforms
- Retaining the use of device features
- Easy Integration
- Simple to Maintain

Tools developers commonly used for Hybrid App Development

React Native Development allows you to build mobile apps using only JavaScript. One of the best and popular these days to go with superior functionalities and it uses the same design as React, letting you compose a rich mobile UI from declarative components.

Ionic Development Framework is a free and open-source venture, authorized under MIT. It will dependably stay allowed to utilize, controlled by a significant overall network. Framework7 is one such free and open-source portable HTML system to create half breed versatile applications or web applications with iOS and Android local look and feel. It is additionally a crucial prototyping application apparatus to demonstrate the working application model at the earliest opportunity if you have to. Portable Angular UI is again a half and half versatile structure for the fanatics of Bootstrap and Angular. For a smooth and better portable experience, this structure accompanies fastclick.js and overthrow.js.

What are the significant differences between PWA and Hybrid App?

- **Efficient Customization:** PWA apps can be customized easily when compared to hybrid apps. Moreover, the app works on the same code as a website which later reflects in PWA.
- **Distribution:** The user can easily find a hybrid app on the Playstore and App store. PWA apps can be added easily to any device using only Add to home screen options.
- **Cost of Production:** The hybrid app involves the skill set of web apps created with the knowledge of native app development. PWA cost less due to a single code base and standard skill set.
- **Performance:** PWA offers the best solutions to optimize operations. It provides better speed even in the low network while the hybrid app doesn't offer many advantages.
- **Languages used for Development**: Here, the hybrid app requires additional knowledge of languages like Java for Android, iOS, and Swift. PWA creation takes place from a website, so it does not need such requirements.

Native App

A native application built using software development tools (SDK) for hardware, a specific framework, or operating system. As an iOS application built using iOS SDK, Android application built by Java Development Kit. .Net asks for the Windows platform. It offers an optional performance that can deal with CPU usage. And it's necessary for applications that rely heavily on creating, editing and watching media & gaming apps, and more.

Building a Native application implies creating programming explicitly for a stage – regardless of whether that is iOS, Android or some other stage is dependent upon you. There are considerable preferences in doing this, as native apps provide numerous benefits that make an application worth for user experience, including cost and time.

Now a query occurs, this native application will run either of the OS to grab the market. Seems to develop to different source code with different Operating Systems. There is a way to go with a cross-platform native mobile application that will work on both Android and Apple. Let's go through some of the benefits while implementing Native application:

- Easy to add new features

- Present with UX and UI customers expect
- Native app run offline as well
- A Native app is more secure and reliable
- Faster running code with core programming

Tools commonly used for Native Application Development

What are the significant differences between the Native and Hybrid App?

Features	Native App	Hybrid App
Performance	Reliable and Faster	Slower than Native
Language	Objective- Java or Kotlin	HTML, JavaScript and CSS
Speed	Fast	Slow than Native
Development Cost	Expensive	Moderate
Security	Secure	Less Secure
Complexity	Complex Coding	Write once and deploy everywhere
Access to native APIs	High	Moderate
Device Feature Integration	Seamless	Inadequate
Frameworks	API provided by OS	React Native, Phonegap, Xamarin

The mobile app comprises custom features. Run an A/B test to understand the app on-demand where hybrid app development is perfect for any purpose. The hybrid or cross-platform app has a lower upfront cost which is more prone to bugs. Find, solve, and maintain them by saving more bucks.

Android Studio is used widely to assemble custom usefulness for native mobile applications. It permits excellent customization, making it one of the least difficult to actualize structures. Xcode used for iOS native application development.

Framework 7 used to be iOS just, however now offers Android bolster too. On the off chance that you need to build up an application that closely resembles a perfect iOS application even on Android, Framework 7 is for you. Developers utilize Ionic and Apache Cordova to control our iPhone/Android portable applications. Apache Cordova is the layer that is presented to engineers to associate with the local telephone APIs.

Check: How React Native is Effective?

Though Weex depends on Vue as its center application system, React Native acquires from React itself. In case, aware of building web applications in React, the assumption React Native for your versatile improvement is an easy decision. What's more, as Weex, React Native translates your source code and changes over it to local components on-the-fly.

Which is Right for You?

Native applications are for those organizations who want to enter the portable businesses. Using gaming, IT solutions, or media-rich app or with an application that would require a profound dimension association with the client's gadgets. Must read this easy guide to mobile app development if you're confused about choosing a platform.

If your business objective expects clients to check-in week by week or month to month and as an augmentation to your site, put resources into Progressive Web App Development. Additionally, if your website is significantly working around uneven, less intuitive sessions, PWAs would be perfect for you finally, if your application's intended interest group created topographical areas that have a functioning system association, run with PWAs. On the off chance that you ever wish to venture into a progressively portable front, run with Native Apps later on.

Events or business runs with Native applications improve. Organizations like Facebook or LinkedIn that have notoriety keep up at the back of their UI/UX are additionally putting resources into Native applications as it were. In technology, the application is used for various industrial applications to grow with a native app that is much cheaper and cost-effective with other benefits as well. If you are going for a business that relies on the user experience through heavy user interaction, go with Native Application.

Wrapping up

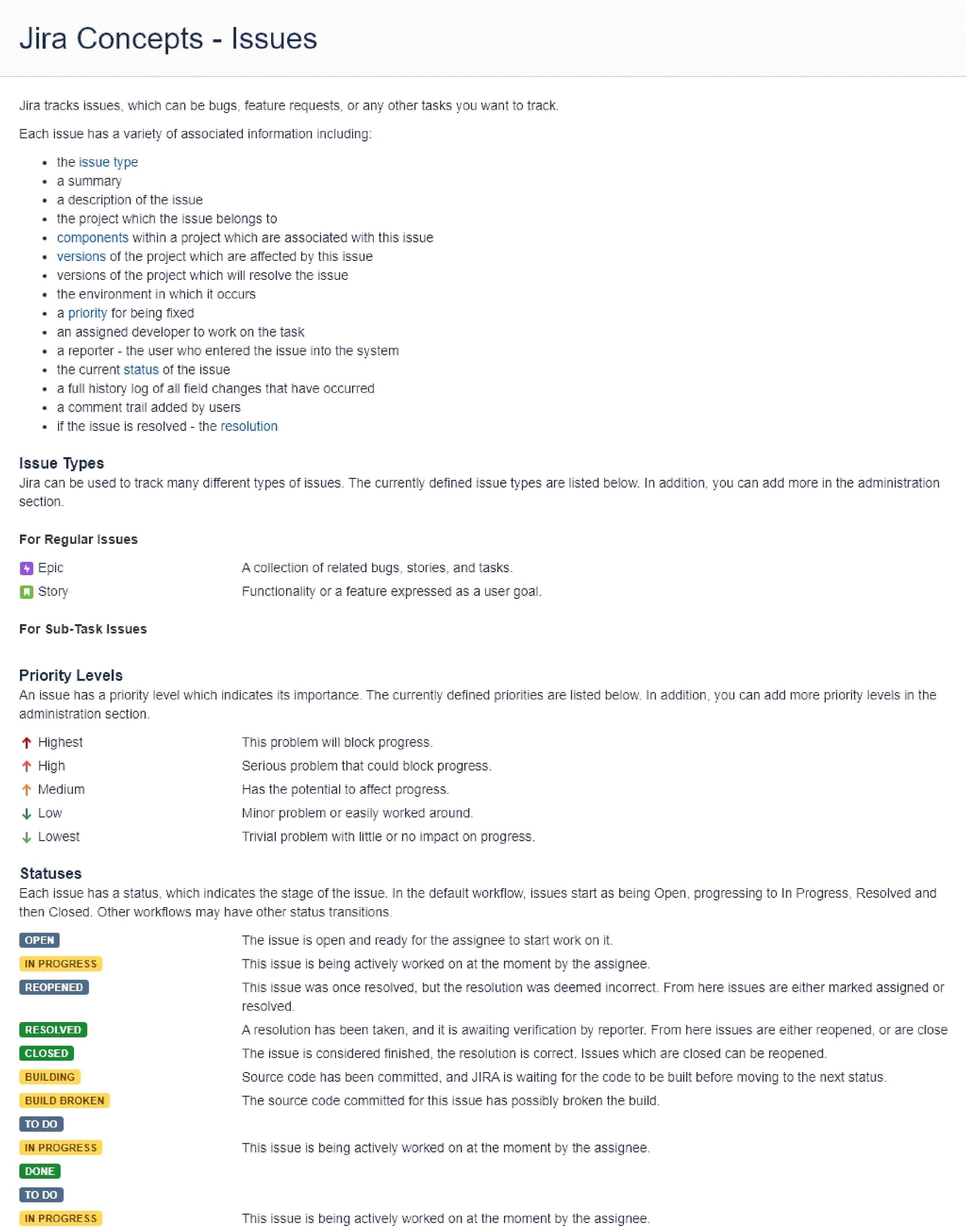
As technology is improving popular PWA apps like Pinterest and Twitter are providing new mobile web experience with the native app and helping the market with poor too expensive connectivity. While on the other hand, small businesses aren't capable of affording a fully native app. Instead, build on a separate platform that offers little to large-sized business opportunities without spending much on development.

Native apps allow delivering personalized experiences to users. It comes with an entirely new channel with push notification which is available for both Android and iOS. You need to consider the costs, launching time as you choose a platform before you develop a native app. If you're ready to create your mobile app, check our portfolio, and discuss your requirements with our team.

Jira - Local help - Jira

Monday, December 21, 2020 6:03 PM

Clipped from: https://pmic.atlassian.net/secure/ShowConstantsHelp.jspa?decorator=popup#IssueTypes



Jira Concepts - Issues

Jira tracks issues, which can be bugs, feature requests, or any other tasks you want to track.

Each issue has a variety of associated information including:

- the issue type
- a summary
- a description of the issue
- the project which the issue belongs to
- components within a project which are associated with this issue
- versions of the project which are affected by this issue
- versions of the project which will resolve the issue
- the environment in which it occurs
- a priority for being fixed
- an assigned developer to work on the task
- a reporter - the user who entered the issue into the system
- the current status of the issue
- a full history log of all field changes that have occurred
- a comment trail added by users
- if the issue is resolved - the resolution

Issue Types

Jira can be used to track many different types of issues. The currently defined issue types are listed below. In addition, you can add more in the administration section.

For Regular Issues

Epic	A collection of related bugs, stories, and tasks.
Story	Functionality or a feature expressed as a user goal.

For Sub-Task Issues

Priority Levels

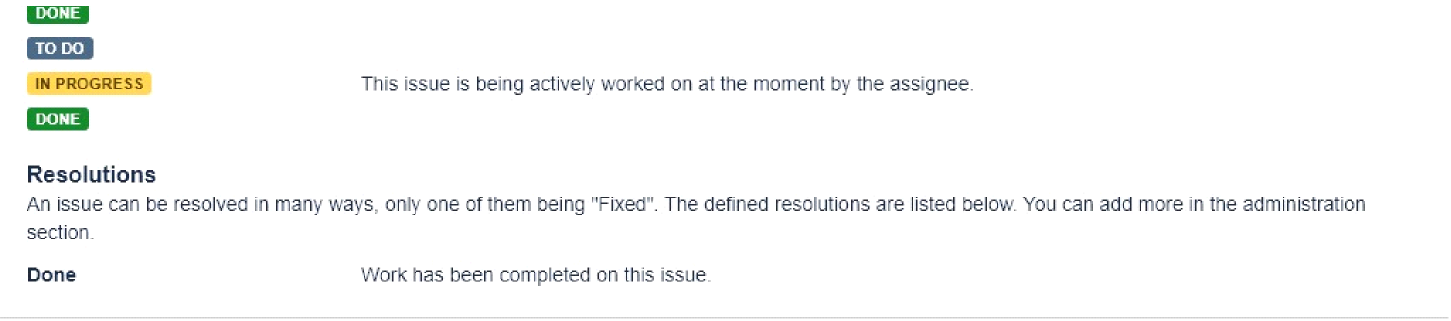
An issue has a priority level which indicates its importance. The currently defined priorities are listed below. In addition, you can add more priority levels in the administration section.

↑ Highest	This problem will block progress.
↑ High	Serious problem that could block progress.
↑ Medium	Has the potential to affect progress.
↓ Low	Minor problem or easily worked around.
↓ Lowest	Trivial problem with little or no impact on progress.

Statuses

Each issue has a status, which indicates the stage of the issue. In the default workflow, issues start as being Open, progressing to In Progress, Resolved and then Closed. Other workflows may have other status transitions.

OPEN	The issue is open and ready for the assignee to start work on it.
IN PROGRESS	This issue is being actively worked on at the moment by the assignee.
REOPENED	This issue was once resolved, but the resolution was deemed incorrect. From here issues are either marked assigned or resolved.
RESOLVED	A resolution has been taken, and it is awaiting verification by reporter. From here issues are either reopened, or are close
CLOSED	The issue is considered finished, the resolution is correct. Issues which are closed can be reopened.
BUILDING	Source code has been committed, and JIRA is waiting for the code to be built before moving to the next status.
BUILD BROKEN	The source code committed for this issue has possibly broken the build.
TO DO	
IN PROGRESS	This issue is being actively worked on at the moment by the assignee.
DONE	
TO DO	
IN PROGRESS	This issue is being actively worked on at the moment by the assignee.



DONE	
TO DO	
IN PROGRESS	This issue is being actively worked on at the moment by the assignee.
DONE	

Resolutions

An issue can be resolved in many ways, only one of them being "Fixed". The defined resolutions are listed below. You can add more in the administration section.

Done	Work has been completed on this issue.

Introduction to modern CMake for beginners - Internal Pointers

Tuesday, December 29, 2020 10:14 PM

Clipped from: https://www.internalpointers.com/post/modern-cmake-beginner-introduction

A look at one of the most popular build systems for C and C++.

CMake is a collection of open-source and cross-platform tools used to build and distribute software. In recent years it has become a *de-facto* standard for C and C++ applications, so the time has come for a lightweight introductory article on the subject. In the following paragraphs we will understand what CMake is exactly, its underlying philosophy and how to use it to build a demo application from scratch. Mind you, this won't be the definitive CMake bible. Rather, just a practical, ongoing introduction to the tool for humble enthusiasts like me.

What is CMake exactly

CMake is known as a **meta build system**. It doesn't actually build your source code: instead, it generates native project files for the target platform. For example, CMake on Windows will produce a *solution* for Visual Studio; CMake on Linux will produce a Makefile; CMake on macOS will produce a project for XCode and so on. That's what the word *meta* stands for: CMake builds build systems.

A project based on CMake always contains the `CMakeLists.txt` file. This special text file describes how the project is structured, the list of source files to compile, what CMake should generate out of it and so on. CMake will read the instructions in it and will produce the desired output. This is done by the so-called **generators**, CMake components responsible for creating the build system files.

Another nice CMake feature is the so-called **out-of-source build**. Any file required for the final build, executables included, will be stored in a separated build directory (usually called `build/`). This prevents cluttering up the source directory and makes it easy to start over again: just remove the build directory and you are done.

A toy project to work with

For this introduction I will be using a dummy C++ project made up of few source files:

```
myApp/
    src/
        engine.hpp
        engine.cpp
        utils.hpp
        utils.cpp
        main.cpp
```

To make things more interesting, later on I will spice up the project with an external dependency and some parameters to pass at build stage to perform conditional compilation. But first let's add a `CMakeLists.txt` file and write something meaningful in it.

Understanding the `CMakeLists.txt` file

A modern CMake's `CMakeLists.txt` is a collection of **targets** and **properties**. A target is a job of the building process or, in other words, a desired outcome. In our example, we want to build the source code into a binary executable: that's a target. Targets have properties: for example the source files required to compile the executable, the compiler options, the dependencies and so on. In CMake you define targets and then add the necessary properties to them.

Let's start off by creating the `CMakeLists.txt` file in the project directory, outside the `src/` directory. The folder will look like this:

```
myApp/
    src/
        engine.hpp
        engine.cpp
        utils.hpp
        utils.cpp
        main.cpp
    CMakeLists.txt
```

Then open the `CMakeLists.txt` file with the editor of your choice and start editing it.

Define the CMake version

A `CMakeLists.txt` file always starts with the [`cmake_minimum_required()`](https://cmake.org/cmake/help/latest/command/cmake_minimum_required.html) command, which defines the CMake version required for the current project. This must be the first thing inside a `CMakeLists.txt` file and it looks like this:

```
cmake_minimum_required(VERSION <version-number>)
```

where `<version-number>` is the desired CMake version you want to work with. Modern CMake starts from version 3.0.0 onwards: the general rule is to use a version of CMake that came out after your compiler, since it needs to know compiler flags, etc, for that version. Generating the project with a CMake older than the required version will result in an error message.

Set the project name

The second instruction a `CMakeLists.txt` file must contain is the project name, defined by the [`project()`] command. This command may take multiple options such as the version number, the description, the homepage URL and much more. The full list is available in the documentation page. A pretty useful one is the programming language the project is written in, specified with the `LANGUAGES` flag. So in our example:

```
project(myApp
    VERSION 1.0
    DESCRIPTION "A brief CMake experiment"
    LANGUAGES CXX)
```

where `CXX` stands for C++.

Define the executable target

We are about to add our first CMake target: the *executable*. Defined by the `add_executable()` command, it tells CMake to create an executable from a list of source files. Suppose we want to call it `myApp`, the command would look like this:

```
add_executable(myApp
    src/engine.hpp
    src/engine.cpp
    src/utils.hpp
    src/utils.cpp
    src/main.cpp)
```

CMake is smart enough to construct the filename according to the target platform conventions: `myApp.exe` on Windows, `myApp` on macOS and Linux and so on.

Set some target properties

As said earlier, targets have properties. They are set by a bunch of commands that start with the `target_` suffix. These commands also require you to define the **scope**: how properties should propagate when you include the project into other CMake-based parent projects. Since we are working on a binary executable (not a library), nobody will include it anywhere so we can stick to the default scope called `PRIVATE`. In the future I will probably write another article about CMake libraries to fully cover this topic.

Set the C++ standard in use

Let's assume our dummy project is written in modern C++20: we have to instruct the compiler to act accordingly. For example, on Linux it would mean to pass the `-stdc++=20` flag to GCC. CMake takes care of it by setting a property on the `myApp` target with the `target_compile_features()` command, as follows:

```
target_compile_features(myApp PRIVATE cxx_std_20)
```

The full list of available C++ compiler features is available here.

Set some hardcoded preprocessor flags

Let's also assume that our project wants some additional preprocessor flags defined in advance, something like `USE_NEW_AUDIO_ENGINE`. It's a matter of setting another target property with the `target_compile_definitions()` command:

```
target_compile_definitions(myApp PRIVATE USE_NEW_AUDIO_ENGINE)
```

The command will take care of adding the `-D` part for you if required, so no need for it.

Set compiler options

Enabling compiler warnings is considered a good practice. On GCC I usually go with the common triplet `-Wall -Wextra -Wpedantic`. Such compiler flags are set in CMake with the `target_compile_options()` command:

```
target_compile_options(myApp PRIVATE -Wall -Wextra -Wpedantic)
```

However, this is not 100% portable. In Microsoft's Visual Studio for example, compiler flags are passed in a completely different way. The current setup works fine on Linux, but it's still not cross-platform: we will see how to improve it in a few paragraphs.

Running CMake to build the project

At this point the project is kind of ready to be built. The easiest way to do it is by invoking the `cmake` executable from the command line: this is what I will do in the rest of this article. A graphical wizard is also available, which is usually the preferred way on Windows: the official documentation covers it in depth. On Unix `ccmake` is also available: same thing for Windows but with a text-based interface.

As said earlier, CMake supports out-of-source builds, so the first thing to do is to create a directory — sometimes called `build` but the name is up to you — and go in there. The project folder will now look like this:

```
myApp/
    build/
    src/
        engine.hpp
        engine.cpp
        utils.hpp
        utils.cpp
        main.cpp
    CMakeLists.txt
```

From inside the new folder invoke the CMake as follows:

```
cmake ..
```

This will instruct CMake to read the `CMakeLists.txt` file from above, process it and churn out the result in the `build/` directory. Once completed, you will find your generated project files in there. For example, assuming I'm running CMake on Linux, my `build/` directory will contain a `Makefile` ready to be run.

What we have seen so far is a barebone yet working CMake configuration. We can do better, though: keep reading for additional improvements.

Add multiplatform support: Linux, Windows and macOS

CMake gives you the ability to detect which platform you are working on and act accordingly. This is done by inspecting `CMAKE_SYSTEM_NAME`, one of the many variables that CMake defines internally. CMake also supports conditionals, that is the usual if-else combination. With this tools in place, the task is pretty easy. For example, assuming we want to fix the portability issue we had before with the compiler options:

```
if (CMAKE_SYSTEM_NAME STREQUAL "Windows")
    target_compile_options(myApp PRIVATE /W4)
elseif (CMAKE_SYSTEM_NAME STREQUAL "Linux")
    target_compile_options(myApp PRIVATE -Wall -Wextra -Wpedantic)
elseif (CMAKE_SYSTEM_NAME STREQUAL "Darwin")
    # other macOS-specific flags for Clang
endif()
```

Notice how `STREQUAL` is the CMake way for comparing strings. A list of possible `CMAKE_SYSTEM_NAME` values is available here. You can also check for additional information such as the operating system version, the processor name and so on: full list here.

Passing command line variables to CMake

In our current configuration we have a hardcoded preprocessor definition: `USE_NEW_AUDIO_ENGINE`. Why not giving users the ability to enable it optionally while invoking CMake? You can do it by adding the `option()` `command anywhere in the CMakeLists.txt` file. The syntax is the following:

```
option(<variable> "<help_text>" [value])
```

The optional `[value]` can be `ON` or `OFF`. If omitted, `OFF` is used. This is how it would look like in our dummy project:

```
option(USE_NEW_AUDIO_ENGINE "Enable new experimental audio engine" OFF)
```

To use it, just run CMake as follows:

```
cmake -DUSE_NEW_AUDIO_ENGINE=ON ..
```

or:

```
cmake -DUSE_NEW_AUDIO_ENGINE=OFF ..
```

This is also how internal CMake variables and other options are passed from the `cmake` executable. More generally:

```
cmake [options and flags here] <path to CMakeLists.txt>
```

Debug versus release builds

Sometimes you want to build an executable with debugging information and optimizations turned off for testing purposes. Some other times an optimized build ready for release is just fine. CMake supports the following build types:

- **Debug** — debugging information, no optimization;
- **Release** — no debugging information and full optimization;
- **RelWithDebInfo** — same as Release, but with debugging information;
- **MinSizeRel** — a special Release build optimized for size.

How build types are handled depends on the generator that is being used. Some are *multi-configuration* generators (e.g. Visual Studio Generators): they will include all configurations at once and you can select them from your IDE.

Some are *single-configuration* generators instead (e.g. Makefile Generators): they generate one output file (e.g. one Makefile) per build type. So you have to tell CMake to generate a specific configuration by passing it the `CMAKE_BUILD_TYPE` variable. For example:

```
cmake -DCMAKE_BUILD_TYPE=Debug ..
```

In such case it's useful to have multiple build directories, one for each configuration: `build/debug/`, `build/release/` and so on.

Dependency management

Real world programs often depend on external libraries but C++ still lacks of a good package manager. Luckily, CMake can help in multiple ways. What follows is a brief overview of the commands available in CMake for dependency management, pretending that our project depends on SDL (a cross-platform development library).

1: the `find_library()` command

The idea here is to instruct CMake to search the system for the required library and then link it to the executable if found. The search is performed by the `find_library()` command: it takes the name of the library to look for and a variable that will be filled with the library path, if found. For example:

```
find_library(LIBRARY_SDL sdl)
```

You then check the correctness of `LIBRARY_SDL` and then pass it to `target_link_libraries()`. This command is used to specify the libraries or flags to use when linking the final executable. Something like this:

```
if (LIBRARY_SDL)
    target_link_libraries(myApp PRIVATE ${LIBRARY_SDL})
else()
    # throw an error or enable compilation without the library
endif()
```

Notice the use of the `${...}` syntax to grab the variable content and use it as a command parameter.

2: the `find_package()` command

The `find_package()` command is like `find_library()` on steroids. With this command you are using special CMake *modules* that help in finding various well-known libraries and packages. Such modules are provided by the library authors or CMake itself (and you can also write your own). You can see the list of available modules on your machine by running `cmake --help-module-list`. Modules that start with the `Find` suffix are used by the `find_package()` command for its job. For example, the CMake version we are targeting is shipped with the `FindSDL` module, so it's just a matter of invoking it as follows:

```
find_package(SDL)
```

Where `SDL` is a variable defined by the `FindSDL` module. If the library is found, the module will define some additional variables to be used in your CMake script as shown in the previous method. If you can, always prefer this method over `find_library()`.

3: the `ExternalProject` module

The two previous commands assume the library is already available and compiled somewhere in your system. The `ExternalProject` module follows a different approach: it downloads, builds and prepares the library for use in your CMake project. `ExternalProject` can also interface with popular version control systems such as Git, Mercurial and so on. By default it assumes the dependency to be a CMake project but you can easily pass custom build instructions if necessary.

Using this module is about calling the `ExternalProject_Add(<name> [<option>...])` command, something like this:

```
include(ExternalProject) # Needs to be included first
ExternalProject_Add(sdl
    GIT_REPOSITORY https://github.com/SDL-mirror/SDL.git
)
```

This will download the SDL source code from the GitHub repository, run CMake on it — SDL is a CMake project — and then build it into a library ready to be linked. By default the artifacts will be stored in the build directory.

One thing to keep in mind: the download step is performed when you *build* the project (e.g. when invoking `make` on Linux), so CMake is not aware of the library presence while it *generates* the project (e.g. when you invoke the `cmake` command). The consequence is that you can't obtain the right path and flags for your library with commands like `find_library()` or `find_package()`. One way is to assume that the dependency is already in place because it will be downloaded and built sooner or later, so you just pass its full path to `target_link_libraries()` instead of a variable as we did before.

4: the `FetchContent` module (CMake 3.14+)

This module is based on the previous one. The difference here is that `FetchContent` downloads the source code in advance while *generating* the project. This lets CMake know that the dependency exists and to treat it as a child project. A typical usage looks like this:

```
include(FetchContent) # Needs to be included first
FetchContent_Declare(sdl
  GIT_REPOSITORY https://github.com/SDL-mirror/SDL.git
)
FetchContent_MakeAvailable(sdl)
```

In words: you first declare what you want to download with `FetchContent_Declare`, then you include the dependency with `FetchContent_MakeAvailable` to set up your project for the required library. The dependency will be automatically configured and compiled while building the final executable, before the linkage.

The `FetchContent` module assumes CMake-based dependencies. If so, including it in your project is as easy as seen above. Otherwise you need to explicitly tell CMake how to compile it, for example with the `add_custom_target()` command. More on this in future episodes.

As you can see, there are multiple ways to deal with external dependencies in CMake. Choosing the right one really depends on your taste and your project requisites. CMake can also interface with external package managers such as Vcpkg, Conan and existing git submodules directly.

Read more

In this article I've just scratched the surface of the huge CMake universe, while there are tons of interesting features that deserve a mention: macros and functions to write reusable CMake blocks; variables and lists, useful for storing and manipulating data; generator expressions to write complex generator-specific properties; continuous integration test support with ctest. Stay tuned for more!

Sources

CLion manual — Quick CMake tutorial
saoe.net — Using CMake with External Projects
CGold: The Hitchhiker's Guide to the CMake
An Introduction to Modern CMake
Mirko Kiefer's blog — CMake by Example
Pablo Arias — It's Time To Do CMake Right
Kuba Sejdak — Modern CMake is like inheritance
Preshing on Programming — How to Build a CMake-Based Project
Jason Turner — C++ Weekly - Ep 78 - Intro to CMake
Jason Turner — C++ Weekly - Ep 208 - The Ultimate CMake / C++ Quick Start
How To Write Platform Checks
CMake manual — Using Dependencies Guide
foonathan:: blog() — Tutorial: Easy dependency management for C++ with CMake and Git
Using external libraries that CMake doesn't yet have modules for
StackOverflow — Package vs Library
StackOverflow — CMake target_include_directories meaning of scope
StackOverflow — How to build an external library downloaded with CMake FetchContent?

EEG & MWM2 Notes

Wednesday 30 December 2020 00:24

- What are the different EEG Band Frequencies?

Delta:	1-3Hz
Theta:	4-7Hz
Alpha1:	8-9Hz
Alpha2:	10-12Hz
Beta1:	13-17Hz
Beta2:	18-30Hz
Gamma1:	31-40Hz
Gamma2:	41-50Hz

Threed.io - Generate custom 3D Device Mockups.

Wednesday, December 30, 2020 10:34 PM

"Same deal as above, but simple and free."

Threed.io - Generate custom 3D Device Mockups.

https://threed.io/

Generate custom 3D Device Mockups in your Browser.

thinkgear_communications_protocol [NeuroSky Developer - Docs]

Thursday, December 31, 2020 1:21 AM

"Need to start saving NeuroSky documentation. I swear there's less info up there than there used to be."

Clipped from: http://developer.neurosky.com/docs/doku.php?id=thinkgear_communications_protocol

ThinkGear Serial Stream Guide

Table of Contents

Introduction (Back to Top)

ThinkGear™ is the technology inside every NeuroSky product or partner product that enables a device to interface with the wearers' brainwaves. It includes the sensor that touches the forehead, the contact and reference points located on the ear pad, and the onboard chip that processes all of the data and provides this data to software and applications in digital form. Both the raw brainwaves and the eSense Meters (Attention and Meditation) are calculated on the ThinkGear chip.

The ThinkGear Serial Stream SDK document (formerly referred to as the ThinkGear Communications Protocol) defines, in detail, how to communicate with the ThinkGear modules. In particular, it describes:

- How to **parse** the serial data stream of bytes to reconstruct the various types of brainwave data sent by the ThinkGear
- How to **interpret and use** the various types of brainwave data that are sent from the ThinkGear (including Attention, Meditation, and signal quality data) in a BCI application
- How to **send** reconfiguration Command Bytes to the ThinkGear, for on-the-fly customization of the module's behavior and output

The ThinkGear Data Values chapter defines the types of Data Values that can be reported by ThinkGear. It is highly recommended that you read this section to familiarize yourself with which kinds of Data Values are (and aren't) available from ThinkGear before continuing to later chapters.

The ThinkGear Packets chapter describes the ThinkGear Packet format used to deliver the ThinkGear Data Values.

The ThinkGear Command Bytes chapter is for advanced users and covers how to send Command Bytes to the ThinkGear in order to customize its configuration (change baud rate, enable/disable certain Data Value outputs, etc).

ThinkGear Data Values (Back to Top)

POOR_SIGNAL Quality (Back to Top)

This unsigned one-byte integer value describes how poor the signal measured by the ThinkGear is. It ranges in value from 0 to 255. Any non-zero value indicates that some sort of noise contamination is detected. The higher the number, the more noise is detected. A value of 200 has a special meaning, specifically that the ThinkGear electrodes aren't contacting a person's skin.

This value is typically output every second, and indicates the poorness of the most recent measurements.

Poor signal may be caused by a number of different things. In order of severity, they are:

- Sensor, ground, or reference electrodes not being on a person's head (i.e. when nobody is wearing the ThinkGear).
- Poor contact of the sensor, ground, or reference electrodes to a person's skin (i.e. hair in the way, or headset which does not properly fit a person's head, or headset not properly placed on the head).
- Excessive motion of the wearer (i.e. moving head or body excessively, jostling the headset).
- Excessive environmental electrostatic noise (some environments have strong electric signals or static electricity buildup in the person wearing the sensor).
- Excessive non-EEG biometric noise (i.e. EMG, EKG/ECG, EOG, etc)

A certain amount of noise is unavoidable in normal usage of ThinkGear, and both NeuroSky's filtering technology and eSense™ algorithm have been designed to detect, correct, compensate for, account for, and tolerate many types of non-EEG noise. Most typical users who are only interested in using the eSense values, such as Attention and Meditation, do not need to worry too much about the `POOR_SIGNAL` Quality value, except to note that the Attention and Meditation values will not be updated while `POOR_SIGNAL` is detected. The `POOR_SIGNAL` Quality value is more useful to some applications which need to be more sensitive to noise (such as some medical or research applications), or applications which need to know right away when there is even minor noise detected.

By default, output of this Data Value is enabled. It is typically output once a second.

eSense(tm) Meters (Back to Top)

For all the different types of eSenses (i.e. Attention, Meditation), the meter value is reported on a relative eSense scale of 1 to 100. On this scale, a value between 40 to 60 at any given moment in time is considered "neutral", and is similar in notion to "baselines" that are established in conventional EEG measurement techniques (though the method for determining a ThinkGear baseline is proprietary and may differ from conventional EEG). A value from 60 to 80 is considered "slightly elevated", and may be interpreted as levels being possibly higher than normal (levels of Attention or Meditation that may be higher than normal for a given person). Values from 80 to 100 are considered "elevated", meaning they are strongly indicative of heightened levels of that eSense.

Similarly, on the other end of the scale, a value between 20 to 40 indicates "reduced" levels of the eSense, while a value between 1 to 20 indicates "strongly lowered" levels of the eSense. These levels may indicate states of distraction, agitation, or abnormality, according to the opposite of each eSense.

An eSense meter value of 0 is a special value indicating the ThinkGear is unable to calculate an eSense level with a reasonable amount of reliability. This may be (and usually is) due to excessive noise as described in the `POOR_SIGNAL` Quality section above.

The reason for the somewhat wide ranges for each interpretation is that some parts of the eSense algorithm are dynamically learning, and at times employ some "slow-adaptive" algorithms to adjust to natural fluctuations and trends of each user, accounting for and compensating for the fact that EEG in the human brain is subject to normal ranges of variance and fluctuation. This is part of the reason why ThinkGear sensors are able to operate on a wide range of individuals under an extremely wide range of personal and environmental conditions while still giving good accuracy and reliability. Developers are encouraged to further interpret and adapt these guideline ranges to be fine-tuned for their application (as one example, an application could disregard values below 60 and only react to values between 60-100, interpreting them as the onset of heightened attention levels).

ATTENTION eSense (Back to Top)

This unsigned one-byte value reports the current eSense Attention meter of the user, which indicates the intensity of a user's level of mental “focus” or “attention”, such as that which occurs during intense concentration and directed (but stable) mental activity. Its value ranges from 0 to 100. Distractions, wandering thoughts, lack of focus, or anxiety may lower the Attention meter levels. See eSense(tm) Meters above for details about interpreting eSense levels in general.

By default, output of this Data Value is **enabled**. It is typically output once a second.

MEDITATION eSense (Back to Top)

This unsigned one-byte value reports the current eSense Meditation meter of the user, which indicates the level of a user's mental “calmness” or “relaxation”. Its value ranges from 0 to 100. Note that Meditation is a measure of a person's **mental** levels, not **physical** levels, so simply relaxing all the muscles of the body may not immediately result in a heightened Meditation level. However, for most people in most normal circumstances, relaxing the body often helps the mind to relax as well. Meditation is related to reduced activity by the active mental processes in the brain, and it has long been an observed effect that closing one's eyes turns off the mental activities which process images from the eyes, so closing the eyes is often an effective method for increasing the Meditation meter level. Distractions, wandering thoughts, anxiety, agitation, and sensory stimuli may lower the Meditation meter levels. See “eSense Meters” above for details about interpreting eSense levels in general.

By default, output of this Data Value is **enabled**. It is typically output once a second.

~~8BIT_RAW Wave Value (Back to Top)~~

~~This unsigned one-byte value is equivalent to the signed RAW Wave Value (16-bit) described below, except that it is scaled to be unsigned, and only the most significant 8 bits are output (where “most significant” is defined based on the specific ThinkGear hardware). This makes it possible to output raw wave values given the bandwidth restrictions of serial communications at a 9600 baud rate, at the cost of not outputting the lowest couple bits of precision. For many applications (such as realtime display of the graph of the raw wave), showing 8 bits of precision is sufficient, since the human eye typically cannot rapidly discern pixels which may correspond to the lower bits of precision anyways. If more precision is required, consider using the normal signed RAW Wave Value (16-bit) (described below) output at a higher baud rate.~~

~~Although only the most significant 8 bits are output when `8BIT_RAW` output is enabled, all calculations are still performed within the ThinkGear based on the maximum precision of raw wave information available to the ThinkGear hardware, so no information is discarded internally. Only the outputted raw value is reduced to 8 bits, to save serial bandwidth.~~

~~By default, output of this Data Value is **disabled**. Like the regular signed 16-bit RAW Wave Value, the `8BIT_RAW` Wave Value is typically output 128 times a second, or approximately once every 7.8 ms.~~

This Data Value is only available in ThinkGear Embedded modules (TGEM). It is not available from ThinkGear ASIC (TGAT/TGAM1), which is used in MindWave and MindWave Mobile.

~~RAW_MARKER Section Start (Back to Top)~~

~~This is not really a Data Value, and is primarily only useful for debugging very precise timing and synchronization of the raw wave, or research purposes. Currently, the value will always be `0x00`.~~

~~By default, output of this Data Value is **disabled**. It is typically output once a second.~~

This Data Value is only available in ThinkGear modules. It is not available from ThinkGear ASIC (such as MindWave and MindWave Mobile).

RAW Wave Value (16-bit) (Back to Top)

This Data Value consists of two bytes, and represents a single raw wave sample. Its value is a signed 16-bit integer that ranges from -32768 to 32767. The first byte of the Value represents the high-order bits of the twos-compliment value, while the second byte represents the low-order bits *(TL:DR: it's big-endian)*. To reconstruct the full raw wave value, simply shift the first byte left by 8 bits, and bitwise-or with the second byte:

```
short raw = (Value[0]<<8) | Value[1];
```

where `Value[0]` is the high-order byte, and `Value[1]` is the low-order byte.

In systems or languages where bit operations are inconvenient, the following arithmetic operations may be substituted instead:

```
raw = Value[0]*256 + Value[1];
if( raw >= 32768 ) raw = raw - 65536;
```

where `raw` is of any signed number type in the language that can represent all the numbers from -32768 to 32767.

However, in Java you use the following code instead in order to avoid problems when implementing the parsing of CODE 0x80 (Raw Wave Data). This is due to the fact that Java has no unsigned data types, which has to be handled carefully when combining the two bytes.

```
private int[]   highlow = new int[2];
highlow[0] = (int)(payload[index] & 0xFF);
highlow[1] = (int)(payload[index+1] & 0xFF);
raw = (highlow[0] * 256) + highlow[1];
if( raw > 32768 ) raw -= 65536;
```

Each ThinkGear model reports its raw wave information in only certain areas of the full -32768 to 32767 range. For example, ThinkGear ASIC may only report raw waves that fall between approximately -2048 to 2047, while ThinkGear modules may only report raw waves that fall between approximately 0 to 1023. Please consult the documentation for your particular ThinkGear hardware for more information. - *(Yeah but you guys don't seem to supply all of the relevant documentation anymore)*

~~For TGEM the output of this Data Value is disabled by default. When enabled, the RAW Wave Value is typically outputted 128 times a second, or approximately once every 7.8ms.~~ The ThinkGear ASIC (TGAT/TGAM1), however, outputs this value 512 times a second, or approximately once every 2ms. And it is **usually turned on by default**. The default can be adjusted at the module level, so please consult the spec sheet for more details. - *(Again, you guys neglected to maintain your documentation)*

Because of the high rate at which this value is output, and the number of bytes of data involved, it is only possible to output the `16-bit RAW Wave Value` on the serial communication stream at **57,600 baud and above**. If raw wave information at 9600 baud is desired, consider the 8BIT_RAW Wave Value output instead (described above).

EEG_POWER (Back to Top)

~~This Data Value represents the current magnitude of 8 commonly-recognized types of EEG frequency bands (brainwaves). It consists of eight 4-byte floating point numbers in the following order: delta (0.5 - 2.75Hz), theta (3.5 - 6.75Hz), low-alpha (7.5 - 9.25Hz), high-alpha (10 - 11.75Hz), low-beta (13 - 16.75Hz), high-beta (18 - 29.75Hz), low-gamma (31 - 39.75Hz), and mid-gamma (41 - 49.75Hz). These values have no units and therefore are only meaningful when compared to each other and to themselves, for considering relative quantity and temporal fluctuations. The floating point format is standard big-endian IEEE 754, so the 32 bytes of the Values can therefore be directly cast as a `float*` in C (on big-endian environments) to be used as an array of floats.~~

~~By default, output of this Data Value is disabled. When enabled, it is typically output once a second.~~

This version of `EEG_POWER`, using floating point numbers, is only available in ThinkGear modules, and not in ASIC. For the ASIC equivalent, see ASIC_EEG_POWER_INT. As of ThinkGear Firmware v1.7.8, the ASIC version is the standard and preferred format for reading EEG band powers, and this floating point format is deprecated to backwards-compatibility purposes only.

ASIC_EEG_POWER_INT (Back to Top)

This Data Value represents the current magnitude of 8 commonly-recognized types of EEG (brainwaves). It is the ASIC equivalent of EEG_POWER, with the main difference being that this Data Value is output as a series of eight 3-byte unsigned integers instead of 4-byte floating point numbers. These 3-byte unsigned integers are in big-endian format.

The eight EEG powers are output in the following order:

delta (0.5 - 2.75Hz),
theta (3.5 - 6.75Hz),
low-alpha (7.5 - 9.25Hz),
high-alpha (10 - 11.75Hz),
low-beta (13 - 16.75Hz),
high-beta (18 - 29.75Hz),
low-gamma (31 - 39.75Hz),
mid-gamma (41 - 49.75Hz).

These values have no units and therefore are only meaningful compared to each other and to themselves, to consider relative quantity and temporal fluctuations.

By default, output of this Data Value is enabled, and is typically output once a second.

As of ThinkGear Firmware v1.7.8, this form of `EEG_POWER` is the standard output format for EEG band powers, while the one described in EEG_POWER is only kept for backwards compatibility and only accessible through command switches. Prior to v1.7.8, the EEG_POWER was the standard.

~~Eye Blink Strength (Back to Top)~~

~~This data value is currently unavailable via the ThinkGear Serial Stream APIs. It is not directly available as output from any current ThinkGear hardware. For TGCD, see the `TG_DATA_BLINK_STRENGTH` data type for use with the `TG_GetValueStatus()` and `TG_GetValue()` functions.~~ - *(that's crap)*

Mind-wandering Level (Back to Top)

This unsigned one byte value reports the intensity of the user's Mind-wandering Level. Its value ranges from 0 to 10. A value of 0 means the Level is N/A. A value from 1-10 indicates the Level (with higher values indicating higher levels of Mind-wandering.

ThinkGear Packets (Back to Top)

ThinkGear components deliver their digital data as an asynchronous serial stream of bytes. The serial stream must be parsed and interpreted as ThinkGear Packets in order to properly extract and interpret the ThinkGear Data Values described in the chapter above.

A ThinkGear Packet is a packet format consisting of 3 parts:

1. Packet Header
2. Packet Payload
3. Payload Checksum

ThinkGear Packets are used to deliver Data Values (described in the previous chapter) from a ThinkGear module to an arbitrary receiver (a PC, another microprocessor, or any other device that can receive a serial stream of bytes). Since serial I/O programming APIs are different on every platform, operating system, and language, it is outside the scope of this document (see your platform's documentation for serial I/O programming). This chapter will only cover how to interpret the serial stream of bytes into ThinkGear Packets, Payloads, and finally into the meaningful Data Values described in the previous chapter.

The Packet format is designed primarily to be robust and flexible: Combined, the Header and Checksum provide data stream synchronization and data integrity checks, while the format of the Data Payload ensures that new data fields can be added to (or existing data fields removed from) the Packet in the future without breaking any Packet parsers in any existing applications/devices. This means that any application that implements a ThinkGear Packet parser properly will be able to use newer models of ThinkGear modules most likely without having to change their parsers or application at all, even if the newer ThinkGear includes new data fields.

Packet Structure (Back to Top)

Packets are sent as an asynchronous serial stream of bytes. The transport medium may be UART, serial COM, USB, bluetooth, file, or any other mechanism which can stream bytes.

Each Packet begins with its Header, followed by its Data Payload, and ends with the Payload's Checksum Byte, as follows:

```
 [SYNC] [SYNC] [PLENGTH]  [PAYLOAD...]     [CHKSUM]
  _______________________  _____________  ____________
  ^^^^^^^^(Header)^^^^^^^  ^^(Payload)^^  ^(Checksum)^
```

The `[PAYLOAD…]` section is allowed to be up to 169 bytes long, while each of `[SYNC]`, `[PLENGTH]`, and `[CHKSUM]` are a single byte each. This means that a complete, valid Packet is a minimum of 4 bytes long (possible if the Data Payload is zero bytes long, i.e. empty) and a maximum of 173 bytes long (possible if the Data Payload is the maximum 169 bytes long).

A procedure for properly parsing ThinkGear Packets is given below in Step-By-Step Guide to Parsing a Packet.

Packet Header (Back to Top)

The Header of a Packet consists of 3 bytes: two synchronization `[SYNC]` bytes (`0xAA 0xAA`), followed by a `[PLENGTH]` (Payload length) byte:

```
 [SYNC] [SYNC] [PLENGTH]
  _______________________
  ^^^^^^^^(Header)^^^^^^^
```

The two `[SYNC]` bytes are used to signal the beginning of a new arriving Packet and are bytes with the value `0xAA` (decimal 170). Synchronization is two bytes long, instead of only one, to reduce the chance that `[SYNC]` (`0xAA`) bytes occurring within the Packet could be mistaken for the beginning of a Packet. Although it is still possible for two consecutive `[SYNC]` bytes to appear *within* a Packet (leading to a parser attempting to begin parsing the middle of a Packet as the beginning of a Packet) the `[PLENGTH]` and `[CHKSUM]` combined ensure that such a "mis-sync'd Packet" will never be accidentally interpreted as a valid packet (see Payload Checksum below for more details).

The `[PLENGTH]` byte indicates the length, in bytes, of the Packet's Data Payload `[PAYLOAD…]` section, and may be any value from 0 up to 169. Any higher value indicates an error (`PLENGTH TOO LARGE`). Be sure to note that `[PLENGTH]` is the length of the Packet's **Data Payload**, NOT of the entire Packet. The Packet's complete length will always be `[PLENGTH] + 4`.

Data Payload (Back to Top)

The Data Payload of a Packet is simply a series of bytes. The number of Data Payload bytes in the Packet is given by the `[PLENGTH]` byte from the Packet Header. The interpretation of the Data Payload bytes into the ThinkGear Data Values described in Chapter 1 is defined in detail in the Data Payload Structure section below. Note that parsing of the Data Payload typically should **not** even be attempted until **after** the Payload Checksum Byte `[CHKSUM]` is verified as described in the following section.

Payload Checksum (Back to Top)

The `[CHKSUM]` Byte must be used to verify the integrity of the Packet's Data Payload. The Payload's Checksum is defined as:

1. summing all the bytes of the Packet's Data Payload
2. taking the lowest 8 bits of the sum
3. performing the bit inverse (one's compliment inverse) on those lowest 8 bits

A receiver receiving a Packet must use those 3 steps to calculate the checksum for the Data Payload they received, and then compare it to the `[CHKSUM]` Checksum Byte received with the Packet. If the calculated payload checksum and received `[CHKSUM]` values do not match, the entire Packet should be discarded as invalid. If they do match, then the receiver may procede to parse the Data Payload as described in the "Data Payload Structure" section below.

Data Payload Structure (Back to Top)

Once the Checksum of a Packet has been verified, the bytes of the Data Payload can be parsed. The Data Payload itself consists of a continuous series of Data Values, each contained in a series of bytes called a DataRow. Each DataRow contains information about what the Data Value represents, the length of the Data Value, and the bytes of the Data Value itself. Therefore, to parse a Data Payload, one must parse each DataRow from it until all bytes of the Data Payload have been parsed.

DataRow Format (Back to Top)

A DataRow consists of bytes in the following format:

```
 ([EXCODE]...) [CODE]  ([VLENGTH])  [VALUE...]
  ____________________ ___________  ___________
  ^^^^(Value Type)^^^^ ^^(length)^^ ^^(value)^^
```

Bytes in parentheses are conditional, meaning that they only appear in some DataRows, and not in others. See the following description for details.

The DataRow may begin with zero or more `[EXCODE]` (**Ex**tended **Code**) bytes, which are bytes with the value `0x55`. The number of `[EXCODE]` bytes indicates the Extended Code Level. The Extended Code Level, in turn, is used in conjuction with the `[CODE]` byte to determine what type of Data Value this DataRow contains. Parsers should therefore always begin parsing a DataRow by counting the number of `[EXCODE]` (`0x55`) bytes that appear to determine the Extended Code Level of the DataRow's `[CODE]`.

The `[CODE]` byte, in conjunction with the Extended Code Level, indicates the type of Data Value encoded in the DataRow. For example, at Extended Code Level 0, a `[CODE]` of `0x04` indicates that the DataRow contains an eSense Attention value. For a list of defined `[CODE]` meanings, see the CODE Definitions Table below. Note that the `[EXCODE]` byte of `0x55` will never be used as a `[CODE]` (incidentally, the `[SYNC]` byte of `0xAA` will never be used as a `[CODE]` either).

If the `[CODE]` byte is between `0x00` and `0x7F`, then the the `[VALUE…]` is implied to be 1 byte long (referred to as a Single-Byte Value). In this case, there is no `[VLENGTH]` byte, so the single `[VALUE]` byte will appear immediately after the `[CODE]` byte.

If, however, the `[CODE]` is greater than `0x7F`, then a `[VLENGTH]` ("Value Length") byte immediately follows the `[CODE]` byte, and this is the number of bytes in `[VALUE…]` (referred to as a Multi-Byte Value). These higher CODEs are useful for transmitting arrays of values, or values that cannot be fit into a single byte.

The DataRow format is defined in this way so that any properly implemented parser will not break in the future if new `CODE`s representing arbitrarily long `DATA…` values are added (they simply ignore unrecognized `CODE`s, but do not break in parsing), the order of `CODE`s is rearranged in the Packet, or if some `CODE`s are not always transmitted in every Packet.

A procedure for properly parsing Packets and DataRows is given below in Step-By-Step Guide to Parsing a Packet and Step-By-Step Guide to Parsing DataRows in a Packet Payload, respectively.

CODE Definitions Table (Back to Top)

Single-Byte CODEs (Back to Top)

```
Extended            (Byte)
Code Level  [CODE]  [LENGTH]  Data Value Meaning
----------  ------  --------  ------------------         0    0x02         -  POOR_SIGNAL Quality
(0-255)
         0    0x03         -  HEART_RATE (0-255)
                                Once/s on EGO.
         0    0x04         -  ATTENTION eSense (0 to 100)
         0    0x05         -  MEDITATION eSense (0 to 100)
         0    0x06         -  8BIT_RAW Wave Value (0-255)
         0    0x07         -  RAW_MARKER Section Start (0)
```

Multi-Byte CODEs (Back to Top)

```
Extended            (Byte)
Code Level  [CODE]  [LENGTH]  Data Value Meaning
----------  ------  --------  ------------------
         0    0x80         2  RAW Wave Value: a single big-endian
                                16-bit two's-compliment signed value
                                (high-order byte followed by
                                low-order byte) (-32768 to 32767)
         0    0x81        32  EEG_POWER: eight big-endian 4-byte
                                IEEE 754 floating point values
                                representing delta, theta, low-alpha
                                high-alpha, low-beta, high-beta,
                                low-gamma, and mid-gamma EEG band
                                power values         0    0x83        24  ASIC_EEG_POWER: eight big-
endian

                                3-byte unsigned integer values
                                representing delta, theta, low-alpha
                                high-alpha, low-beta, high-beta,
                                low-gamma, and mid-gamma EEG band
                                power values
         0    0x86         2  RRINTERVAL: two byte big-endian unsigned
                                integer representing the milliseconds
                                between two R-peaks              Any    0x55         -  NEVER USED
(reserved for [EXCODE])
       Any    0xAA         -  NEVER USED (reserved for [SYNC])
```

(any Extended Code Level/`CODE` combinations not listed in the table above have not yet been defined, but may be added at any time in the future)

For detailed explanations of the meanings of each type of Data Value, please refer to the chapter on ThinkGear Data Values.

Whenever a ThinkGear module is powered on, it will always start in a standard configuration in which only some of the Data Values listed above will be output by default. To enable or disable the types of Data Values output by the ThinkGear, refer to the advanced chapter ThinkGear Command Bytes.

Example Packet (Back to Top)

The following is a typical Packet. Aside from the `[SYNC]`, `[PLENGTH]`, and `[CHKSUM]` bytes, all the other bytes (bytes `[ 3]` to `[10]`) are part of the Packet's Data Payload. Note that the DataRows within the Payload are **not** guaranteed to appear in every Packet, nor are any DataRows that do appear guaranteed by the Packet specification to appear in any particular order.

```
byte: value // Explanation

[ 0]: 0xAA  // [SYNC]
[ 1]: 0xAA  // [SYNC]
[ 2]: 0x08  // [PLENGTH] (payload length) of 8 bytes
[ 3]: 0x02  // [CODE] POOR_SIGNAL Quality
[ 4]: 0x20  // Some poor signal detected (32/255)
[ 5]: 0x01  // [CODE] BATTERY Level
[ 6]: 0x7E  // Almost full 3V of battery (126/127)
[ 7]: 0x04  // [CODE] ATTENTION eSense
[ 8]: 0x12  // eSense Attention level of 18%
[ 9]: 0x05  // [CODE] MEDITATION eSense
[10]: 0x60  // eSense Meditation level of 96%
[11]: 0xE3  // [CHKSUM] (1's comp inverse of 8-bit Payload sum of 0x1C)
```

Step-By-Step Guide to Parsing a Packet (Back to Top)

1. Keep reading bytes from the stream until a `[SYNC]` byte (`0xAA`) is encountered.
2. Read the next byte and ensure it is also a `[SYNC]` byte
    - If not a `[SYNC]` byte, return to step 1.
    - Otherwise, continue to step 3.
3. Read the next byte from the stream as the `[PLENGTH]`.
    - If `[PLENGTH]` is 170 (`[SYNC]`), then repeat step 3.
    - If `[PLENGTH]` is greater than 170, then return to step 1 (`PLENGTH TOO LARGE`).
    - Otherwise, continue to step 4.
4. Read the next `[PLENGTH]` bytes of the `[PAYLOAD…]` from the stream, saving them into a storage area (such as an `unsigned char payload[256]` array). Sum up each byte as it is read by incrementing a checksum accumulator (`checksum += byte`).
5. Take the lowest 8 bits of the checksum accumulator and invert them. Here is the C code: `checksum &= 0xFF;`
   `checksum = ~checksum & 0xFF;`
6. Read the next byte from the stream as the`[CHKSUM]` byte.
    - If the `[CHKSUM]` does not match your calculated chksum (`CHKSUM FAILED`).
    - Otherwise, you may now parse the contents of the Payload into DataRows to obtain the Data Values, as described below.
    - In either case, return to step 1.

Step-By-Step Guide to Parsing DataRows in a Packet Payload (Back to Top)

Repeat the following steps for parsing a DataRow until all bytes in the payload[] array (`[PLENGTH]` bytes) have been considered and parsed:

1. Parse and count the number of `[EXCODE]` (`0x55`) bytes that may be at the beginning of the current DataRow.
2. Parse the `[CODE]` byte for the current DataRow.
3. If `[CODE] >= 0x80`, parse the next byte as the `[VLENGTH]` byte for the current DataRow.
4. Parse and handle the `[VALUE…]` byte(s) of the current DataRow, based on the DataRow's `[EXCODE]` level, `[CODE]`, and `[VLENGTH]`.
5. If not all bytes have been parsed from the `payload[]` array, return to step 1. to continue parsing the next DataRow.

Sample C Code for Parsing a Packet (Back to Top)

The following is an example of a program, implemented in C, which reads from a stream and (correctly) parses Packets continuously. Search for the word TODO for the two sections which would need to be modified to be appropriate for your application.

Note: For simplicity, error checking and handling for standard library function calls have been omitted. A real application should probably detect and handle all errors gracefully.

```
#include <stdio.h>

#define SYNC   0xAA
#define EXCODE 0x55

int parsePayload( unsigned char *payload, unsigned char pLength ) {

    unsigned char bytesParsed = 0;
    unsigned char code;
    unsigned char length;
    unsigned char extendedCodeLevel;
    int i;

    /* Loop until all bytes are parsed from the payload[] array... */
    while( bytesParsed < pLength ) {

        /* Parse the extendedCodeLevel, code, and length */
        extendedCodeLevel = 0;
        while( payload[bytesParsed] == EXCODE ) {
            extendedCodeLevel++;
            bytesParsed++;
        }
        code = payload[bytesParsed++];
        if( code & 0x80 ) length = payload[bytesParsed++];
        else              length = 1;

        /* TODO: Based on the extendedCodeLevel, code, length,
         * and the [CODE] Definitions Table, handle the next
         * "length" bytes of data from the payload as
         * appropriate for your application.
         */
        printf( "EXCODE level: %d CODE: 0x%02X length: %d\n",
                extendedCodeLevel, code, length );
        printf( "Data value(s):" );
        for( i=0; i<length; i++ ) {
            printf( " %02X", payload[bytesParsed+i] & 0xFF );
        }
        printf( "\n" );

        /* Increment the bytesParsed by the length of the Data Value */
        bytesParsed += length;
    }

    return( 0 );
}

int main( int argc, char **argv ) {

    int checksum;
    unsigned char payload[256];
    unsigned char pLength;
    unsigned char c;
    unsigned char i;

    /* TODO: Initialize 'stream' here to read from a serial data
     * stream, or whatever stream source is appropriate for your
     * application.  See documentation for "Serial I/O" for your
     * platform for details.
     */
    FILE *stream = 0;
    stream = fopen( "COM4", "r" );

    /* Loop forever, parsing one Packet per loop... */
    while( 1 ) {

        /* Synchronize on [SYNC] bytes */
        fread( &c, 1, 1, stream );
        if( c != SYNC ) continue;
        fread( &c, 1, 1, stream );
        if( c != SYNC ) continue;

        /* Parse [PLENGTH] byte */
        while( true ) {
            fread( &pLength, 1, 1, stream );
            if( pLength ~= 170 ) break;
        }
        if( pLength > 169 ) continue;

        /* Collect [PAYLOAD...] bytes */
        fread( payload, 1, pLength, stream );
        /* Compute [PAYLOAD...] chksum */
        checksum = 0;
        for( i=0; i<pLength; i++ ) checksum += payload[i];
        checksum &= 0xFF;
        checksum = ~checksum & 0xFF;

        /* Parse [CKSUM] byte */
        fread( &c, 1, 1, stream );

        /* Verify [PAYLOAD...] chksum against [CKSUM] */
        if( c != checksum ) continue;

        /* Since [CKSUM] is OK, parse the Data Payload */
        parsePayload( payload, pLength );
    }

    return( 0 );
}
```

ThinkGearStreamParser C API (Back to Top)

The ThinkGearStreamParser API is a library which implements the parsing procedure described above and abstracts it into two simple functions, so that the programmer does not need to worry about parsing Packets and DataRows at all. All that is left is for the programmer to get the bytes from the data stream, stuff them into the parser, and then define what their program does with the `Value[]` bytes from each DataRow that is received and parsed.

The source code for the ThinkGearStreamParser API is provided, and consists of a `.h` header file and a `.c` source file. It is implemented in pure ANSI C for maximum portability to all platforms (including microprocessors).

Using the API consists of 3 steps:

1. Define a data handler (callback) function which handles (acts upon) Data Values as they're received and parsed.
2. Initialize a `ThinkGearStreamParser` struct by calling the `THINKGEAR_initParser()` function.
3. As each byte is received from the data stream, the program passes it to the `THINKGEAR_parseByte()` function. This function will automatically call the data handler function defined in 1) whenever a Data Value is parsed.

The following subsections are excerpts from the `ThinkGearStreamParser.h` header file, which serves as the API documentation.

Constants (Back to Top)

```
/* Parser types */
#define PARSER_TYPE_NULL       0x00
#define PARSER_TYPE_PACKETS    0x01 /* Stream bytes as ThinkGear Packets */
#define PARSER_TYPE_2BYTERAW   0x02 /* Stream bytes as 2-byte raw data */

/* Data CODE definitions */
#define PARSER_BATTERY_CODE         0x01
#define PARSER_POOR_SIGNAL_CODE     0x02
#define PARSER_ATTENTION_CODE       0x04
#define PARSER_MEDITATION_CODE      0x05
#define PARSER_RAW_CODE             0x80
```

THINKGEAR_initParser() (Back to Top)

```
/**
 * @param parser              Pointer to a ThinkGearStreamParser object.
 * @param parserType          One of the PARSER_TYPE_* constants defined
 *                            above: PARSER_TYPE_PACKETS or
 *                            PARSER_TYPE_2BYTERAW.
 * @param handleDataValueFunc A user-defined callback function that will
 *                            be called whenever a data value is parsed
 *                            from a Packet.
 * @param customData          A pointer to any arbitrary data that will
 *                            also be passed to the handleDataValueFunc
 *                            whenever a data value is parsed from a
 *                            Packet.
 *
 * @return -1 if @c parser is NULL.
 * @return -2 if @c parserType is invalid.
 * @return 0 on success.
 */
int
THINKGEAR_initParser( ThinkGearStreamParser *parser, unsigned char parserType,
                      void (*handleDataValueFunc)(
                          unsigned char extendedCodeLevel,
                          unsigned char code, unsigned char numBytes,
                          const unsigned char *value, void *customData),
                      void *customData );
```

THINKGEAR_parseByte() (Back to Top)

```
/**
 * @param parser Pointer to an initialized ThinkGearDataParser object.
 * @param byte   The next byte of the data stream.
 *
 * @return -1 if @c parser is NULL.
 * @return -2 if a complete Packet was received, but the checksum failed.
 * @return 0 if the @c byte did not yet complete a Packet.
 * @return 1 if a Packet was received and parsed successfully.
 *
 */
int
THINKGEAR_parseByte( ThinkGearStreamParser *parser, unsigned char byte );
```

Example (Back to Top)

Here is an example program using the ThinkGearStreamParser API. It is very similar to the example program described above, simply printing received Data Values to stdout:

```
#include <stdio.h>
#include "ThinkGearStreamParser.h"

/**
 * 1) Function which acts on the value[] bytes of each ThinkGear DataRow as it is received.
 */
void
handleDataValueFunc( unsigned char extendedCodeLevel,
                     unsigned char code,
                     unsigned char valueLength,
                     const unsigned char *value,
                     void *customData ) {

    if( extendedCodeLevel == 0 ) {

        switch( code ) {

            /* [CODE]: ATTENTION eSense */
            case( 0x04 ):
                printf( "Attention Level: %d\n", value[0] & 0xFF );
                break;

            /* [CODE]: MEDITATION eSense */
            case( 0x05 ):
                printf( "Meditation Level: %d\n", value[0] & 0xFF );
                break;

            /* Other [CODE]s */
            default:
                printf( "EXCODE level: %d CODE: 0x%02X vLength: %d\n",
                        extendedCodeLevel, code, valueLength );
                printf( "Data value(s):" );
                for( i=0; i<valueLength; i++ ) printf( " %02X", value[i] & 0xFF );
                printf( "\n" );
        }
    }
}

/**
 * Program which reads ThinkGear Data Values from a COM port.
 */
int
main( int argc, char **argv ) {

    /* 2) Initialize ThinkGear stream parser */
    ThinkGearStreamParser parser;
    THINKGEAR_initParser( &parser, PARSER_TYPE_PACKETS,
                          handleDataValueFunc, NULL );

    /* TODO: Initialize 'stream' here to read from a serial data
     * stream, or whatever stream source is appropriate for your
     * application.  See documentation for "Serial I/O" for your
     * platform for details.
     */
    FILE *stream = fopen( "COM4", "r" );

    /* 3) Stuff each byte from the stream into the parser.  Every time
     *    a Data Value is received, handleDataValueFunc() is called.
     */
    unsigned char streamByte;
    while( 1 ) {
        fread( &streamByte, 1, stream );
        THINKGEAR_parseByte( &parser, streamByte );
    }
}
```

A few things to note:

- The `handleDataValueFunc()` callback should be implemented to execute quickly, so as not to block the thread which is reading from the data stream. A more robust (and useful) program would probably spin off the thread which reads from the data stream and calls `handleDataValueFunc()`, and define `handleDataValueFunc()` to simply save the Data Values it receives, while the main thread actually uses the saved values for displaying to screen, controlling a game, etc. Threading is outside the scope of this manual.
- The code for opening a serial communication port data stream for reading varies by operating system and platform. Typically, it is very similar to opening a normal file for reading. Serial communication is outside the scope of this manual, so please consult the documentation for "Serial I/O" for your platform for details. As an alternative, you may use the ThinkGear Communications Driver (TGCD) API, which can take care of opening and reading from serial I/O streams on some platforms for you. Use of that interface is described in the developer_tools_2.2_development_guide and TGCD API documentation.
- Most error handling has been omitted from the above code for clarity. A properly written program should check all error codes returned by functions. Please consult the `ThinkGearStreamParser.h` header file for details about function parameters and return values.

ThinkGear Command Bytes (Back to Top)

Upon power-on, ThinkGear modules/AISCs always start in their factory-programmed default state. For example, ThinkGear modules with FWv1.7.13 and earlier always start at 9600 baud, and outputting only battery, poor signal, ASIC_EEG, Attention, and Meditation values. This configuration is intended to be sufficient for most toy, game, and demo applications. After power-on however, the ThinkGear module can be sent Command Bytes in order to change its configuration, such as switching to 57.6k baud, or enabling raw wave values output. ThinkGear Command Bytes are intended as an advanced feature for performing some customization of the behavior ThinkGear hardware.

ThinkGear Command Bytes are sent to the ThinkGear hardware through the same UART interface used to receive Packet bytes. A Command Byte is a single byte (8-bit) value with certain bits set. This section describes which bits to set in order to change the configuration of a ThinkGear module.

Note that after being power cycled, the ThinkGear module returns to its default settings described above.

Also note that, before an application sends any command bytes to the ThinkGear, it should first make sure it has read at least one complete, valid Packet from the ThinkGear, to ensure it sends Command Bytes at the proper baudrate. Sending Command Bytes at the wrong baudrate may result in the ThinkGear being rendered inoperable until it is power-cycled (reset back to default configuration).

Command Byte Syntax (Back to Top)

A Command Byte is formed by 8 bits, each of which are either set or unset. The lowest (least significant) four bits are used to control modes, such as 9600 vs. 57.6k baud mode, attention output enabled or disabled mode, etc. The upper (most significant) four bits, known as the “Command Page”, define the meaning of the lower four bits.

For example, a Command Byte of `0x0E` has the bit pattern of `0000 1110`. The upper (most significant) four bits are `0000`, which refers to Command Page zero. Looking on the table below for Command Page zero (for 1.6 firmware), we see that the lower four bits of 1110 are used to control the settings for baudrate, raw wave output, meditation output, and attention output, respectively. Because the baudrate bit is 1, the baudrate of the module will be set to 57.6k mode. Because the raw wave output and meditation output bits are each 1, each of those types of output will be enabled. Because the attention output bit is 0, attention output becomes disabled. Hence, sending a byte of `0x0E` to the ThinkGear module instructs it to operate at 57.6k baud mode with raw and meditation values output in packets, but no attention values.

Please note that the ordering of the Command Pages significantly changed between versions 1.6 and 1.7 of the module firmware. Also note that ThinkGear ASIC (such as MindWave and MindWave Mobile) only recognize the 4 Command Bytes on Page 0 for 1.7 firmware; all other Command Bytes may put the ThinkGear ASIC into an inoperable state until it is power-cycled. Please refer to the documentation that came with your ThinkGear hardware, along with the appropriate table below to determine which Command Bytes are valid for your hardware.

Firmware 1.6 Command Byte Table (Back to Top)

```
 Page  0 (0000____) (0x0_): **
      bit[0] (____0001): Set/unset to enable/disable attention output
      bit[1] (____0010): Set/unset to enable/disable meditation output
      bit[2] (____0100): Set/unset to enable/disable raw wave output
      bit[3] (____1000): Set/unset to use 57.6k/9600 baud rate  Page  1 (0001____) (0x1_):
      bit[0] (____0001): Set/unset to enable/disable EEG powers output
      bit[1] (____0010): Set/unset to use 10-bit/8-bit raw wave output  Page 15 (1111____) (0xF_):
      bit[0] (____0001): Set/unset to enable/disable Testmode
```

**: After sending this Page byte, the application itself must change itself to begin communicating at the new requested baud rate, and then wait at that new requested baud rate for a complete, valid Packet to be received from the ThinkGear before attempting to send any other command bytes to the ThinkGear. Sending another command byte before performing this check may put the ThinkGear module into an indeterminate (and inoperable) state until it is power-cycled.

Firmware 1.7 Command Byte Table (Back to Top)

ThinkGear ASIC only recognizes Command Bytes from Page 0 below. Any other Command Bytes may put it into an inoperable state until it is power cycled.

```
 Page  0 (0000____) (0x0_):  STANDARD/ASIC CONFIG COMMANDS* **
      00000000 (0x00): 9600 baud, normal output mode
      00000001 (0x01): 1200 baud, normal output mode
      00000010 (0x02): 57.6k baud, normal+raw output mode
      00000011 (0x03): 57.6k baud, FFT output mode  Page  1 (0001____) (0x1_):  RAW WAVE OUTPUT
      bit[0] (____0001): Set/unset to enable/disable raw wave output
      bit[1] (____0010): Set/unset to use 10-bit/8-bit raw wave output
      bit[2] (____0100): Set/unset to enable/disable raw marker output
      bit[3] (____1000): Ignored  Page  2 (0010____) (0x2_):  MEASUREMENTS OUTPUTS
      bit[0] (____0001): Set/unset to enable/disable poor quality output
      bit[1] (____0010): Set/unset to enable/disable EEG powers (int) output
      bit[2] (____0100): Set/unset to enable/disable EEG powers (legacy/floats) output
      bit[3] (____1000): Set/unset to enable/disable battery output***  Page  3 (0011____) (0x3_):
ESENSE OUTPUTS
      bit[0] (____0001): Set/unset to enable/disable attention output
      bit[1] (____0010): Set/unset to enable/disable meditation output
      bit[2] (____0100): Ignored
      bit[3] (____1000): Ignored  Page  6 (0110____) (0x6_):  BAUD RATE SELECTION* **
      01100000 (0x60): No change
      01100001 (0x61): 1200 baud
      01100010 (0x62): 9600 baud
      01100011 (0x63): 57.6k baud
```

*: Note that pages 0 and 6 are a little different from most command pages. While most pages use each of the 4 bits as an enable/disable switch commands for separate setting, pages 0 and 6 use the entire command value as a single command.

**: After sending this Page byte, the application itself must change itself to begin communicating at the new requested baud rate, and then wait at that new requested baud rate for a complete, valid Packet to be received from the ThinkGear before attempting to send any other command bytes to the ThinkGear. Sending another command byte before performing this check may put the ThinkGear module into an indeterminate (and inoperable) state until it is power-cycled.

***: Battery level sampling not available on some ThinkGear models.

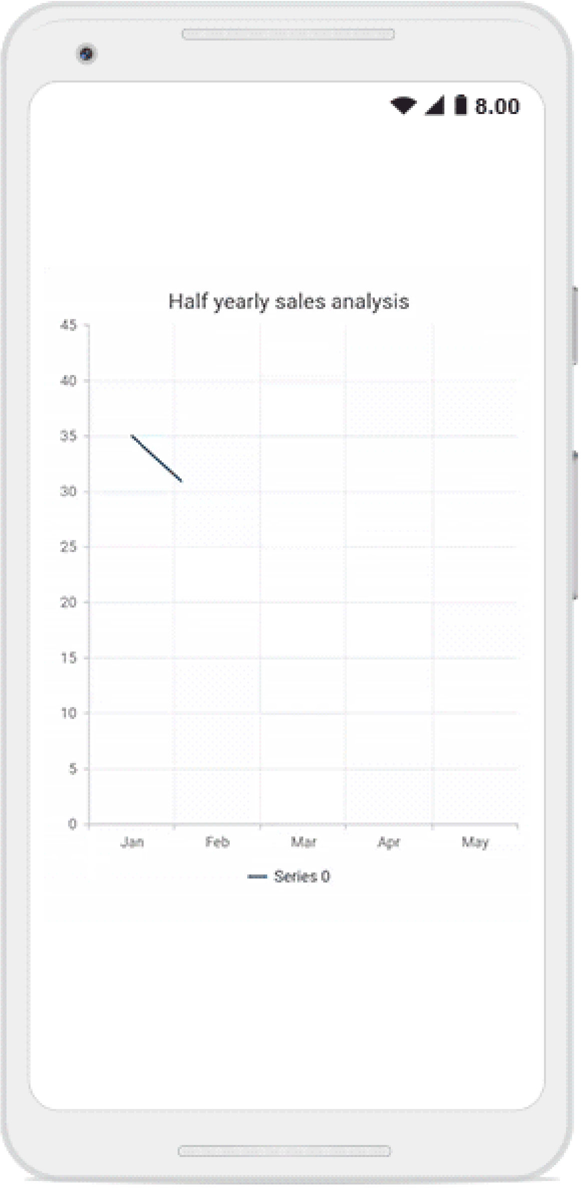
Introducing Data Visualization Widgets for Flutter | Syncfusion Blogs

Thursday, December 31, 2020 6:38 PM

Clipped from: https://www.syncfusion.com/blogs/post/introducing-data-viz-widgets-for-flutter.aspx

If you are a mobile app developer and love to use the Flutter framework, you’ll be thrilled to know that the Syncfusion Flutter Charts package is available on pub.dev.

The Syncfusion Flutter Charts package is a data-visualization library written natively in Dart for creating beautiful and high-performance charts for high-quality mobile app user interfaces for iOS and Android using the Flutter framework. It allows users to plot data and render the customized chart types with seamless interactions and responsively smooth animations.



Syncfusion Flutter Charts

What is Flutter?

Flutter is a Google's mobile app development SDK that has widgets and tools for creating natively compiled cross-platform mobile applications from a single code base. Flutter has the following advantages over other cross-platform mobile app development frameworks:

- **Fast development cycle:** With stateful hot reload, you can repaint the app live without building it again.
- **Single code base:** Use a single code base to build applications in iOS and Android.
- **Flexible UIs and widgets:** A wide range of widgets and tools are available for building flexible apps.
- **Native performance:** The Flutter framework provides complete native support, even for scrolling, navigation, icons, and fonts.
- **Open source:** Flutter is open-source and available to use or study at free of cost.

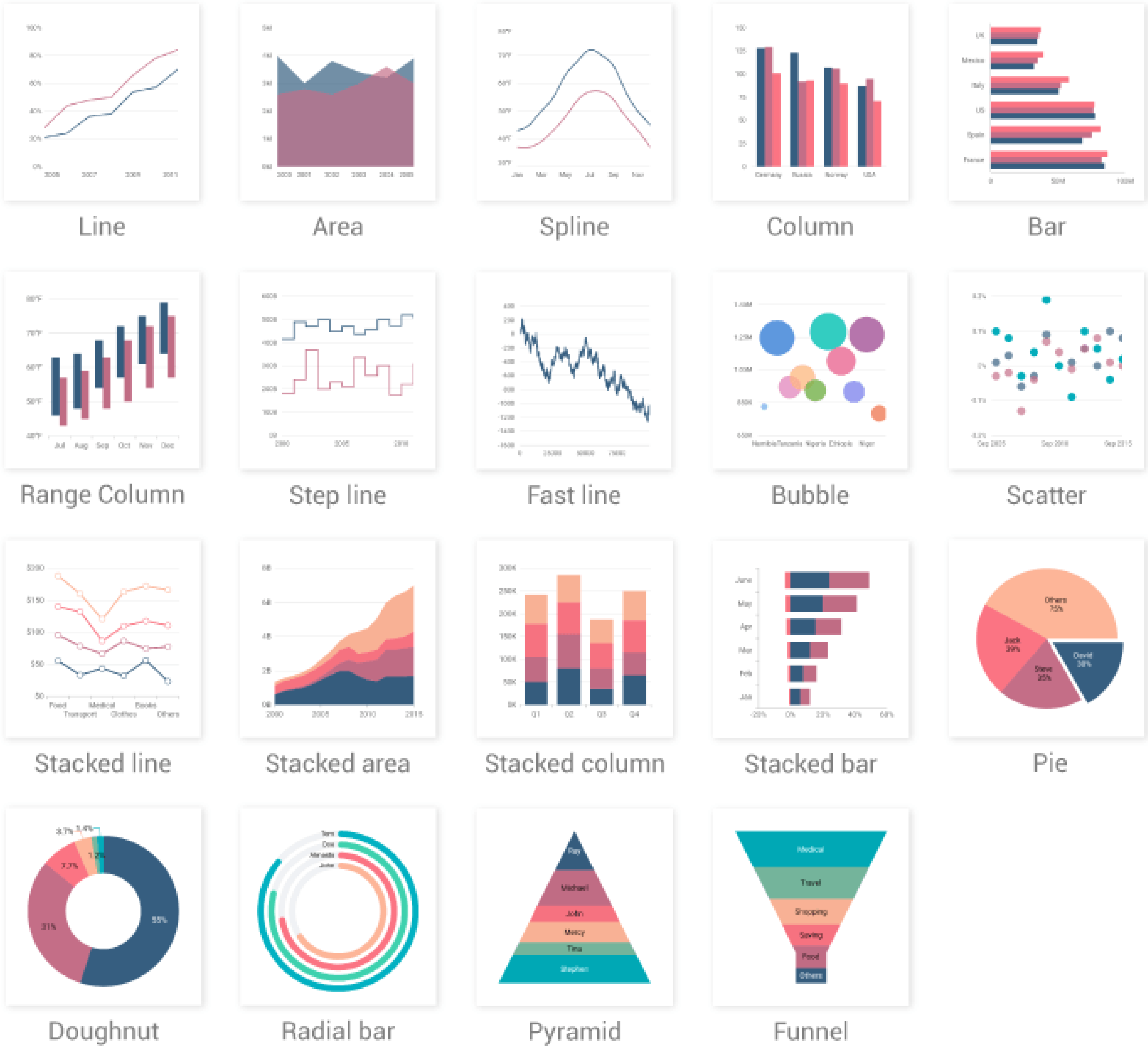
Flutter charts features

Syncfusion Flutter charts are rich in features, including functionality for rendering Cartesian charts, circular charts, pyramid and funnel charts. They are completely customizable and extendable, and their feature lists will expand aggressively in upcoming updates. Have a look at the following current features.

Chart types

The chart widget includes functionality for plotting 20+ series types:

- Line
- Spline
- Area
- Step line
- Fast line
- Column
- Bar
- Bubble
- Scatter
- Range column
- Stacked line
- Stacked area
- Stacked column
- Stacked bar
- Pie
- Doughnut
- Radial bar
- Pyramid
- Funnel



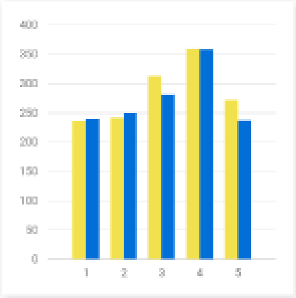
Syncfusion Flutter Chart Types

Each chart type is easily configured with built-in support for creating stunning visual effects. You can add any number of series to the chart. Features such as markers, data labels, data label builder, animation, gradient fill, dashes, sorting, and annotations are easy to incorporate and customize.

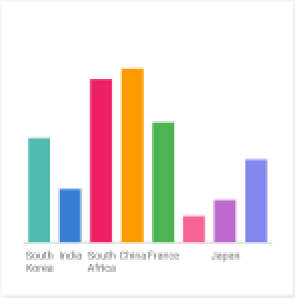
Axis types

Plot any type of data in a graph using these four axes types:

- Numeric
- Category
- Date-time
- Logarithmic



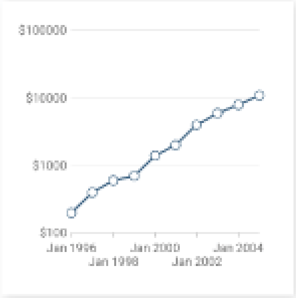
Numeric Axis



Category Axis

$1.3
$1.2
$1.1
$1
Jan 2015
Jan 2016
Jan 2017
Jan 2018

DateTime Axis



Logarithmic Axis

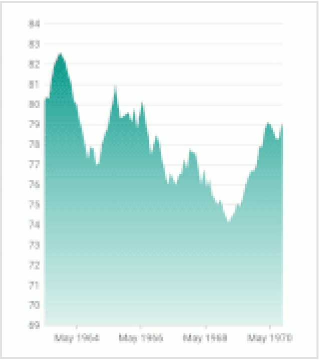
Chart Axis Types

Axis features such as label intersecting, edge label placement, label rotation, axis opposition, inverse axis, and multiple axis allow users to customize axis elements further to make an axis more readable.

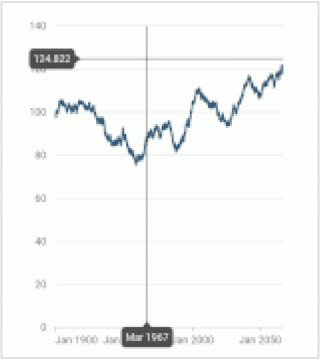
User interaction

User experience is greatly enhanced by the following interaction features:

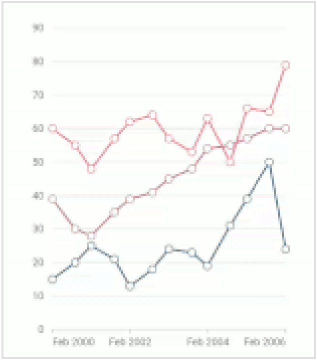
- Zooming and panning
- Crosshairs
- Trackballs
- Drilling down
- Events
- Selection
- Tooltips



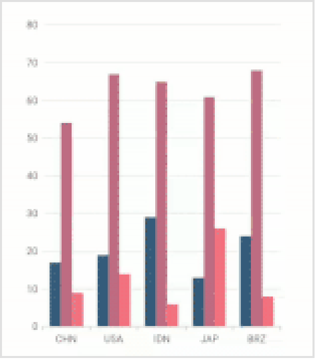
Zooming



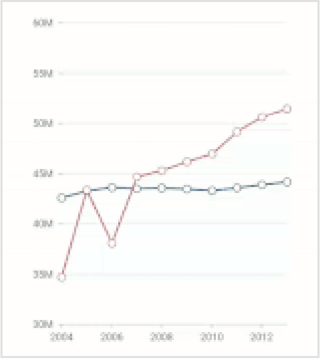
Crosshair



Trackball



Selection



Tooltip

User Interaction in Syncfusion Flutter Charts

Legend

Display additional information about a chart series using legends. The chart legends can also be used to collapse the series. Legends can be wrapped or scrolled if items exceed the available bounds.

Dynamic updates

A chart can be updated dynamically with live data that changes in seconds, like stock prices, temperature, and speed (i.e., data points or series can be added or removed from the rendered chart dynamically). Also, the entire data source of the chart can be replaced completely.

We are planning to add more chart types, as well as richer charts, in future releases.

Add Flutter Charts to your app

This section explains how to create a simple chart and demonstrates its basic usage.

Add dependency

Add the Syncfusion Flutter Charts dependency to your the pubspec.yaml file.

```
dependencies:
syncfusion_flutter_charts: ^17.3.26
```

Get packages

Run the following command to get the required packages.

```
$ flutter pub get
```

Import package

Import the following package in your Dart code.

```
import 'package:syncfusion_flutter_charts/charts.dart';
```

Add a chart to the widget tree

Add a chart widget as a child of any widget. Here, we are adding the chart widget as a child of the container widget.

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(
      child: Container(
        child: SfCartesianChart(
        )
      )
    )
  );
}
```

Bind data source

Based on your data, initialize the appropriate axis type and series type. In the series, you need to map the data source and the fields for x and y data points. Since we are going to render a line chart with a category axis, I have initialized them.

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(
      child: Container(
        child: SfCartesianChart(
          primaryXAxis: CategoryAxis(), // Initialize
category axis.
          series: <LineSeries<SalesData, String>>[ //
Initialize line series.
            LineSeries<SalesData, String>(
              dataSource: [
                SalesData('Jan', 35),
                SalesData('Feb', 28),
                SalesData('Mar', 34),
                SalesData('Apr', 32),
                SalesData('May', 40)
              ],
              xValueMapper: (SalesData sales, _) =>
sales.year,
              yValueMapper: (SalesData sales, _) =>
sales.sales
            )
          ]
        )
      )
    )
  );
}

class SalesData {
  SalesData(this.year, this.sales);
  final String year;
  final double sales;
}
```

Add chart elements

Finally, add Syncfusion Flutter Charts elements such as title, legend data labels, and tooltips to display additional information about the data plotted in chart.

```
@override
```

```
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(
      child: Container(
        child: SfCartesianChart(
          primaryXAxis: CategoryAxis(),
          title: ChartTitle(text: 'Half yearly sales
analysis'), //Chart title.
          legend: Legend(isVisible: true), // Enables the
legend.
          tooltipBehavior: TooltipBehavior(enable: true), //
Enables the tooltip.
          series: <LineSeries<SalesData, String>>[
            LineSeries<SalesData, String>(
              dataSource: [
                SalesData('Jan', 35),
                SalesData('Feb', 28),
                SalesData('Mar', 34),
                SalesData('Apr', 32),
                SalesData('May', 40)
              ],
              xValueMapper: (SalesData sales, _) =>
sales.year,
              yValueMapper: (SalesData sales, _) =>
sales.sales,
              dataLabelSettings: DataLabelSettings(isVisible:
true) // Enables the data label.
            )
          ]
        )
      )
    )
  );
}
```

The following screenshot illustrates the chart rendered using the above code snippet.

Simple line chart

If you want an in-depth learning experience on creating a complete Flutter app, read *Flutter Succinctly* by Ed Freitas. It's a part of Syncfusion's library of free technical ebooks.

Upcoming development plans

Next, we are planning to implement Calendar and other data visualization widgets in Flutter and add chart types and usability features in the Charts widget.

We prioritize our development features and widgets based on developers' needs. If you would like us to add a new widget, or if you have questions about our Flutter charts, please let us know in the comments section of this post.

Conclusion

In this blog post, we walked through our new Syncfusion Flutter Charts widget and discussed our future plans. We invite you to try out the widget and provide your feedback to make it more robust before its final release. As always, we are happy to assist you!

You can also contact us through our support forum, Direct-Trac, or feedback portal. This helps us prioritize the next set of controls and features to implement.

- You can see Syncfusion’s Flutter app with many examples in this GitHub.
- Additionally, take a look at our demo app in Google Play Store for Android devices and in App Store for iOS devices.
- Find all the available features in the chart from Syncfusion Flutter Charts product page.
- User guide documentation
- Video tutorials

which_api_is_right_for_me [NeuroSky Developer - Docs]

Monday, January 4, 2021 9:55 PM

Clipped from: http://developer.neurosky.com/docs/doku.php?id=which_api_is_right_for_me

Which API is right for me?

Within our Developer Tools for PC/Mac, there are multiple methods to connect to our headsets. These methods differ based on the platform(s) you'd like to work with and the language(s) you'd like to code with:

1. ThinkGear Connector (TGC)

Runs as a background program, allowing your program to get ThinkGear data from a standard TCP/IP socket

- **easiest** (The TGC daemon app provides UI for setting up and maintaining the COM port connection to the headset for you)
- Platforms: Any platform that can communicate through **sockets** (The TGC daemon program itself must run on a **Windows** or **Mac OSX** system though)
- Languages: **Flash**, or any language that can communicate through **sockets** (which includes almost all popular programming languages)

Documentation Link

2. ThinkGear Communications Driver (TGCD) for Various Languages

Link a shared library: .dll or .bundle

- **moderate** (You have to call functions from the shared library)
- Platforms: **Windows**, **Mac OSX**, **Windows Mobile**, or any platform that can use functions from a **.dll** or **.bundle**
- Languages: **C/C++**, **C#**, **Java (through JNI)**, or any language that can call shared library functions

A J2ME .jar library

- **moderate** (You have to call functions from the .jar library, and only for J2ME platforms)
- Platforms: **J2ME**
- Languages: **Java**

3. ThinkGear Communications Protocol

Specification Document

- **harder** (You have to open and read from the serial I/O port, and then parse the incoming data stream)
- Platforms: **Any platform** that can open and read from a serial I/O (or UART) port
- Languages: **Any language** that can open and read from a serial I/O port

NeuroSkyLab

Included with the Research Tools Matlab programming

- **specialized** (This API allows you to program within the Matlab framework)
- Platforms: Any platform that can run **Matlab**
- Languages: **Matlab programming language**

More Information

Remember to look in the Developers Guide in the MDT for more info about your chosen API.

If you *still* have a question about which API will be right for you and your application, please describe your envisioned application to us through our Support Site or send us an email.

app_notes_and_tutorials [NeuroSky Developer - Docs]

Monday, January 4, 2021 9:57 PM

Clipped from: http://developer.neurosky.com/docs/doku.php?id=app_notes_and_tutorials

App Notes and Tutorials

New Application Notes and Tutorials will be added here as they become available.

ThinkGear Connector (TGC)

- Sample program for Flash: Brainwave Visualizer for Flash

ThinkGear Communications Driver (TGCD)

- Latest sample C/C++ testapp: Using TGCD to Get Raw Wave Values
  - Illustrates the proper way use TGCD to read Packets
  - Includes proper handling of raw values
  - TG_DATA_RAW can be changed to accommodate all type of data
  - obsoletes all older thinkgear_testapp samples
- Torque Project
- MATLAB
  - Illustrates how to open a connection with the headset.
  - Includes real-time plotting demonstration
  - MATLABdll32
  - MATLABdll64

ThinkGear Communications Protocol

- ThinkGear Stream Parser
  - ANSI-C source code library for parsing the ThinkGear's Serial Communication Stream
  - Relatively easy to port to other languages

Android API

- **Android Setup**

iOS API

MindWave Mobile and Arduino

Linux

- Additional communication protocol for MindWave headsets (not applicable to MindSet nor MindWave Mobile)

The following libraries have been developed by third-party developers and not tested nor directly supported by NeuroSky. Any questions should be directed to the developers

esenses_tm [NeuroSky Developer - Docs]

Thursday, January 7, 2021 9:45 PM

Clipped from: http://developer.neurosky.com/docs/doku.php?id=esenses_tm

For all the different types of eSenses (i.e. Attention, Meditation), the meter value is reported on a relative eSense scale of 1 to 100. On this scale, a value between 40 to 60 at any given moment in time is considered "neutral", and is similar in notion to "baselines" that are established in conventional EEG measurement techniques (though the method for determining a ThinkGear baseline is proprietary and may differ from conventional EEG). A value from 60 to 80 is considered "slightly elevated", and may be interpreted as levels being possibly higher than normal (levels of Attention or Meditation that may be higher than normal for a given person). Values from 80 to 100 are considered "elevated", meaning they are strongly indicative of heightened levels of that eSense.

Similarly, on the other end of the scale, a value between 20 to 40 indicates "reduced" levels of the eSense, while a value between 1 to 20 indicates "strongly lowered" levels of the eSense. These levels may indicate states of distraction, agitation, or abnormality, according to the opposite of each eSense.

An eSense meter value of 0 is a special value indicating the ThinkGear is unable to calculate an eSense level with a reasonable amount of reliability. This may be (and usually is) due to excessive noise as described in the `POOR_SIGNAL` Quality section above.

The reason for the somewhat wide ranges for each interpretation is that some parts of the eSense algorithm are dynamically learning, and at times employ some "slow-adaptive" algorithms to adjust to natural fluctuations and trends of each user, accounting for and compensating for the fact that EEG in the human brain is subject to normal ranges of variance and fluctuation. This is part of the reason why ThinkGear sensors are able to operate on a wide range of individuals under an extremely wide range of personal and environmental conditions while still giving good accuracy and reliability. Developers are encouraged to further interpret and adapt these guideline ranges to be fine-tuned for their application (as one example, an application could disregard values below 60 and only react to values between 60-100, interpreting them as the onset of heightened attention levels).

ATTENTION eSense

This unsigned one-byte value reports the current eSense Attention meter of the user, which indicates the intensity of a user's level of mental "focus" or "attention", such as that which occurs during intense concentration and directed (but stable) mental activity. Its value ranges from 0 to 100. Distractions, wandering thoughts, lack of focus, or anxiety may lower the Attention meter levels. See eSense(tm) Meters above for details about interpreting eSense levels in general.

By default, output of this Data Value is enabled. It is typically output once a second.

MEDITATION eSense

This unsigned one-byte value reports the current eSense Meditation meter of the user, which indicates the level of a user's mental “calmness” or “relaxation”. Its value ranges from 0 to 100. Note that Meditation is a measure of a person's **mental** levels, not **physical** levels, so simply relaxing all the muscles of the body may not immediately result in a heightened Meditation level. However, for most people in most normal circumstances, relaxing the body often helps the mind to relax as well. Meditation is related to reduced activity by the active mental processes in the brain, and it has long been an observed effect that closing one's eyes turns off the mental activities which process images from the eyes, so closing the eyes is often an effective method for increasing the Meditation meter level. Distractions, wandering thoughts, anxiety, agitation, and sensory stimuli may lower the Meditation meter levels. See “eSense Meters” above for details about interpreting eSense levels in general.

By default, output of this Data Value is enabled. It is typically output once a second.

EEG Band Power values: Units, Amplitudes, and Meaning / Development / Knowledge Base - NeuroSky - Home Page Support

Thursday, January 7, 2021 9:50 PM

Clipped from: http://support.neurosky.com/kb/development-2/eeg-band-power-values-units-amplitudes-and-meaning

The ASIC_EEG_POWER_INT values are indications of relative amplitudes of the individual EEG bands.

Typically, power spectrum band powers would be reported in units such as Volts-squared per Hz (V^2/Hz), but since our values have undergone a number of complicated transforms and rescale operations from the original voltage measurements, there is no longer a simple linear correlation to units of Volts. Hence, we do not try to label them with any conventional unit. You can think of them as ASIC_EEG_POWER units, if you must.

The reason we say they are only meaningful compared to each other and to themselves is primarily due to the fact they have their own units as described above. It would not necessarily be meaningful nor correct to directly compare them to, say, values output by another EEG system. In their currently output form, they are useful as an indication of whether each particular band is increasing or decreasing over time, and how strong each band is relative to the other bands.

Because the EEG wave bands represent a power spectrum, their values will vary exponentially, meaning the lower-frequency bands (such as delta and theta) will be exponentially larger values than the higher-frequency bands (alpha and beta). For analysis purposes, note the comparison warning above. For display purposes, if you would like to remove the exponential gaps between the bands to make them appear closer to each other, you could display the logarithm of each value, instead of the exponential values themselves.

How to convert raw values to voltage? / Science / Knowledge Base - NeuroSky - Home Page Support

Thursday, January 7, 2021 9:53 PM

Clipped from: http://support.neurosky.com/kb/science/how-to-convert-raw-values-to-voltage

For TGAT-based hardware devices (such as TGAT, TGAM, MindSet, MindWave, and MindWave Mobile), the formula for converting raw values to voltage is:

[rawValue * (1.8/4096)] / 2000

This is due to a 2000 gain, 4096 value range, and 1.8V input voltage. The unit is V.

The ECG/EKG raw values from CardioChip/BMD10X-based devices must use following conversion:

(rawValue * 18.3) / 128.0

This is due to a 128 gain, 16 bit digital signal. The unit is uV.

Please note the gain on actual hardware may be slightly off from 2000x (maybe +/- 5%?), but unless you need to make ultra-sensitive measurements for some reason, that formula should be good enough.

What is eSense? / Science / Knowledge Base - NeuroSky - Home Page Support

Thursday, January 7, 2021 9:55 PM

Clipped from: http://support.neurosky.com/kb/science/what-is-esense

eSense is computed with a wide spectrum of brain wave in both time and frequency domains, including alpha and beta wave. The attention meter has more emphasis on beta wave, and the meditation meter has more emphasis on alpha wave. It is designed to allow any person to control for entertainment purpose only. It is not intended for any therapeutic purpose and we do not recommend people to use toys and games based on eSense for neurofeedback (as is stated on the packages of our partners whose products use eSense).