Keycode Generator

This directory contains a keycode generator that can generate Dart code for the `LogicalKeyboardKey` and `PhysicalKeyboardKey` classes.

It generates multiple files across Flutter. For framework, it generates

- `keyboard_key.dart`, which contains the definition and list of logical keys and physical keys; and
- `keyboard_maps.dart`, which contains platform-specific immutable maps used for the `RawKeyboard` API.

For engine, it generates one key mapping file for each platform.

It draws information from various source bases, including online repositories, and manual mapping in the `data` subdirectory. It incorporates this information into a giant list of physical keys (`physical_key_data.json`), and another for logical keys (`logical_key_data.json`). The two files are checked in, and can be used as the data source next time so that output files can be generated without the Internet.

Running the tool

The tool can be run based on the existing database. To do this, run:

```
/PATH/TO/ROOT/bin/gen_keycodes
```

The tool can also be run by rebuilding the database by drawing online information anew before generating the files. To do this, run:

```
/PATH/TO/ROOT/bin/gen_keycodes --collect
```

This will generate `physical_key_data.json` and `logical_key_data.json`. These files should be checked in.

By default this tool assumes that the gclient directory for flutter/engine and the root for the flutter/flutter are placed at the same folder. If not, use `--engine-root=/ENGINE/GCLIENT/ROOT` to specify the engine root.

Other options can be found using `--help`.

Logical Key ID Scheme

To provide logical keys with unique ID codes, Flutter uses a scheme to assign logical keycodes which keeps us out of the business of minting new codes ourselves. This only applies to logical key codes: Flutter's physical key codes are just defined as USB HID codes.

The logical codes are meant to be opaque to the user, and should never be unpacked for meaning, since the coding scheme could change at any time and the meaning is likely to be retrievable more reliably and correctly from the API.

However, if you are porting Flutter to a new platform, you should follow the following guidelines for specifying logical key codes.

The logical key code is a 52-bit integer (due to the limitation of JavaScript). The entire namespace is divided into 32-bit *planes*. The upper 20 bits of the ID represent the plane ID, while the lower 32 bits represent values in the plane. For example, plane 0x1 refers to the range 0x1 0000 0000 - 0x1 FFFF FFFF. Each plane manages how the values within the range are assigned.

The planes are planned as follows:

- **Plane 0x00**: The Unicode plane. This plane contains keys that generate Unicode characters when pressed (this includes dead keys, but not e.g. function keys or shift keys). The value is defined as the Unicode code point corresponding to the character, lower case and without modifier keys if possible. Examples are Key A (0x61), Digit 1 (0x31), Colon (0x3A), and Key Ù (0xD9). (The "Colon" key represents a keyboard key that prints the ":" character without modifiers, which can be found on the French layout. On the US layout, the key that prints ":" is the Semicolon key.) This plane also contains key None (0x0).
- **Plane 0x01**: The unprintable plane. This plane contains keys that are defined by the Chromium key list and do not generate Unicode characters. The value is defined as the macro value defined by the Chromium key list. Examples are CapsLock (0x105), ArrowUp (0x304), F1 (0x801), Hiragata (0x716), and TVPower (0xD4B). Some keys that exist in the Chromium key list are not present in Flutter in this plane, most notably modifiers keys (such as Shift). See the Flutter plane below for more information.
- **Plane 0x02**: The Flutter plane. This plane contains keys that are defined by Flutter. The values are also manually assigned by Flutter. Modifier keys are placed in this plane, because Flutter distinguishes between sided modifier keys (for example "ShiftLeft" and "ShiftRight"), while the web doesn't (only has "Shift"). Other examples are numpad keys and gamepad keys.
- **Plane 0x03-0x0F**: Reserved.
- **Plane 0x10-0x1F**: Platform planes managed by Flutter. Each platform plane corresponds to a Flutter embedding officially supported by Flutter. The platforms are listed as follows:

Code	Platform
0x11	Android
0x12	Fuchsia
0x13	iOS
0x14	macOS
0x15	Gtk

Code	Platform
0x16	Windows
0x17	Web
0x18	GLFW

Platform planes store keys that are private to the Flutter embedding of this platform. This most likely means that these keys have not been officially recognized by Flutter.

The value scheme within a platform plane is decided by the platform, typically using the field from the platform's native key event that represents the key's logical effect (such as `keycode`, `virtual key`, etc).

In time, keys that originally belong to a platform plane might be added to Flutter, especially if a key is found shared by multiple platforms. The values of that key will be changed to a new value within the Flutter plane, and all platforms managed by Flutter will start to send the new value, making it a breaking change. Therefore, when handling an unrecognized key on a platform managed by Flutter, it is recommended to file a new issue to add this value to `keyboard_key.dart` instead of using the platform-plane value. However, for a custom platform (see below), since the platform author has full control over key mapping, such change will not cause breakage and it is recommended to use the platform-plane value to avoid adding platform-exclusive values to the framework.

- **Plane 0x20-0x2F**: Custom platform planes. Similar to Flutter's platform planes, but for private use by custom platforms.