Skinetic SDK

Unity Plugin

Documentation

Date	Ver.	Ref.	Description	Author	Approved by
14/12/2022	1.0	SSUPDO001	Initial version	de Tinguy	
11/01/2023	1.1	SSUPDO001	Package Manager	de Tinguy	

Table of Content

1. Objective

The objective of this document is to provide a comprehensive overview as well as a quickstart for the **Skinetic SDK** plugin for *Unity 2019.4* and above.
The examples were tested on:

- 2019.4
- 2020.3
- 2021.3
- 2022.2

2. Disclaimer

3. Import and prerequisite

To import the **Skinetic SDK** package in any project, an archive is provided that you can decompress anywhere on your computer. Then you can manually import it using Unity's Package Manager using **Windows → Package Manager.** Then click on **+ → Add package from disk...** and select the folder SkineticSDK.
You can also directly add this folder into the Packages folder of your project which will add it automatically to your project as an **embedded package** (More details on Unity's documentation).

In order to work properly on Windows **x64**, the **Microsoft Visual C++ Redistribuable** need to be installed on the computer. They are often already available as various softwares depend on it. If they are missing from the computer, they can be obtained at:
https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170

4. Plugin General Overview

This plugin is designed to provide easy access to the **Skinetic SDK** inside *Unity* for both Android and Windows. When compiling your project, the corresponding platform is automatically enabled, no change is needed to handle the haptic layer.

The plugin relies on several key scripts:

- **SkineticDevice Component**
  A custom component to handle an instance of a Skinetic device.

- **PatternAsset ScriptableObject**
  A custom ScriptableObject containing a Skinetic haptic pattern (spn).

- **HapticEffect Component**
  A custom component representing an instance of the referenced pattern that can be configured to specify the haptic effect's rendering behavior.

These components are all using the **Skinetic** namespace.

5. SkineticDevice Component

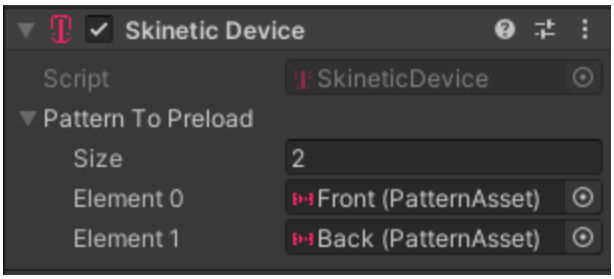
To enable haptic feedback using the **Skinetic SDK**, the first step is to set up a **Skinetic** device to connect to.
This can be done by creating a new *Data Asset* of class "**Skinetic Device**" (in *Miscellaneous/Data Asset*).

5.1. HapticDevice component overview

The HapticDevice component serves as the interface to the haptic rendering layer. This HapticDevice component also abstracts the target platform of your project: developing an application for android, is the same as for windows.

The HapticDevice component allows you to perform several kind of actions:

- Scan available devices
- Handle connection to a Skinetic device
- Get the current device information
- Device's global rendering state
- Handle haptic feedback



The last point will be detailed in section 6.

The HapticDevice component can be added to a gameobject from the menu following **Skinetic → Skinetic Device.**

The connection to a Skinetic device can be established through usb, bluetooth or wifi. Once connected, the communication channel has no impact on the commands that can be given through this component.

5.2. Scanning procedure

A Skinetic device is uniquely identified by its serial number, independently of the connection protocol. To connect to a specific Skinetic device, the serial number is required. It can be obtained by scanning the available devices.

The HapticDevice component provides 3 methods to handle the asynchronous scanning procedure:

- `ScanDevices` will start an asynchronous scanning procedure for all devices available with the connection type requested.

- `ScanStatus` returns the status of the scanning procedure or an error code.
- `GetScannedDevices` returns a list of devices to which it is possible to connect with, once the scanning procedure is complete.

The scanning procedure returns the list of all available devices while providing for each one: the supported output type (type of connection), the serial number, the device's type and the version of the embedded software. The same device can appear several times in the list if it is available through different connection protocols.

Note: The scanning procedure cannot be performed if the component is already connected.

5.3. Connection handling

The **SkineticDevice** component display means to manage the connection to an actual Skinetic device. The connection procedure is also asynchronous. To connect to a specific device, its serial number is required. However, it is possible to also connect to any skinetic device available.

The **SkineticDevice** component provides 4 methods to handle the asynchronous scanning procedure:

- `Connect` will start the asynchronous connection procedure.
- `Disconnect` will start the asynchronous disconnection routine. The disconnection is effective once all resources are released.
- `ConnectionStatus` returns the current connection status to monitor the connection or disconnection procedures.
- `SetConnectionCallback` Set a callback function fired upon connection changes. The callback is triggered at the end of the connection or disconnection procedure.

Note: The connection cannot be performed if the scanning procedure is ongoing.

Note: Setting a callback is an alternative to monitoring the connection state, but is not mandatory. The callback is not executed by the main thread. Hence, the actions that can be performed by it are restricted by Unity.

5.4. Device Introspection

From a **SkineticDevice** component that is already connected to a Skinetic device, several methods are provided to check or display device information such as SDK version (`GetSDKVersion`), Skinetic version (`GetDeviceVersion`), Skinetic device type (`GetDeviceType`) or Skinetic serial number (`GetDeviceSerialNumber`).

5.5. Global rendering state

The rendering of the device can be controlled with **`PauseAll`**, **`ResumeAll`** and **`StopAll`**. These calls affect all playing effects on the device, by pausing/resuming them or even stop and discharge all of them , freeing all resources.

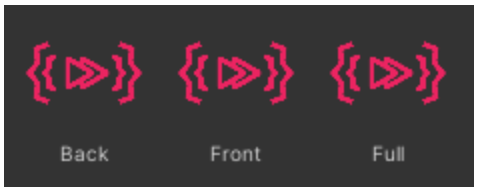
6. Haptic Feedback

The haptic feedback is based on: 1. **Pattern Asset** ScriptableObjects that can handle *.spn* patterns. 2. **HapticEffect** components that can create and manage a parametric instance of the pattern.

6.1. PatternAsset

Patterns are descriptors of sequences of haptic samples that can be created using Unitouch Studio, while haptic effects are the actual feedback rendered using the sequence descriptor.

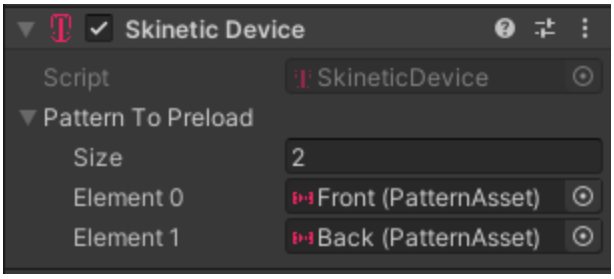
Every pattern created with **Skinetic Studio** can be used directly in *Unity*. Dragging the *.spn* file in the project window will automatically convert it to a **PatternAsset** ScriptableObject. These assets are then available for your project.



The PatternAsset has two fields: a Name that you can use to identify it and a json that the Skinetic SDK is able to process.

- **Load / Unload**
  All patterns to be played must be loaded beforehand into the connected device and unloaded once all interactions are completed (eg. while the game is stopping).
  A PatternAsset can be loaded and unloaded on the fly with the **`LoadPattern`** and **`UnloadPattern`** methods of the SkineticDevice component.

- **PreLoad**
  The SkineticDevice component also has a **PatternToPreload** public field which is a list of patterns that will be loaded during the **OnEnable** event.



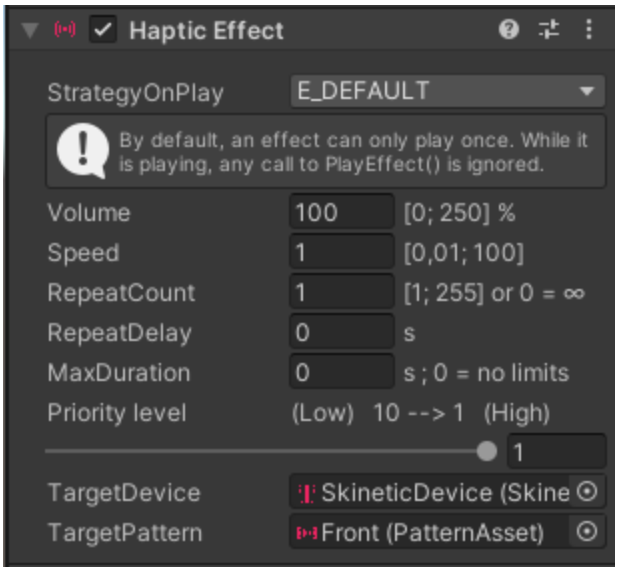
- **LoadedPatterns**
  A read-only list of all loaded patterns is available on the SkineticDevice component.

6.2. HapticEffect Component

HapticEffect components handle actual instances of PatternAssets. They provide several basic features:

- **Pattern Reference**
  In order to operate, a reference to a PatternAsset to play has to be set either in the inspector or through scripting under the field TargetPattern.

- **StrategyOnPlay**
  When the effect is triggered, several strategies are available:
  - **Default:** By default, an effect can only play once. While it is playing, any call to PlayEffect() is ignored.
  - **Forced:** Immediately stop the current playing instance and start a new one.
  - **Pulled:** Each time the effect is triggered a new independent instance of the effect starts.



- **Play Parameters**
  Several parameters are available to modify the way the effect will be rendered when triggered. These parameters can be set in the inspector or through scripting.
  - **Volume**: percentage of the base volume between [0; 250]%.
  - **Speed**: time scale between [0.01; 100]
  - **Repeat Count**: number of repetition of the effect.
  - **Repeat Delay**: pause in seconds between two repetitions of the effect.
  - **Max Duration**: maximum duration in seconds of the effect (regardless of the repeat count).
  - **Priority**: level of priority of the effect (from 1 to 10, 1 being the highest priority). The next section will go into details about the priority system.

- **Play / Stop**
  HapticEffect can be triggered and interrupted with play and stop commands. It is also possible to check the state of an effect to determine if it is still playing or if it has been muted (see next section). To trigger a haptic effect, two workflow are available:

- **From HapticEffect component:** If the **HapticEffect** component has a reference to a target **SkineticDevice** component, `PlayEffect`, `StopEffect`, and `GetEffectState` can be called from the HapticEffect component.
- **From HaptidDevice component**: `PlayEffect`, `StopEffect`, and `GetEffectState` are called from a **SkineticDevice** component by passing a reference to an HapticEffect component that need to be triggered.

6.3. Effect Priority Levels

HapticEffect instances have a priority level when they are triggered. The reason for this system is that there are a limited number of slots that can play simultaneously, i.e., only 5 independent haptic samples can be rendered simultaneously on the vest without regard to their spatialization. Extra samples are then “muted”, i.e., they are alive but not rendered.
Each pattern can use up to 5 simultaneous haptic samples, hence, it is a pattern instance (HapticEffect) as a whole which can be muted.

HapticEffect instances are then ordered by priority and only the effects with higher priority will be rendered, while the others will be muted.
The priority logic follows 3 rules to order two haptic effect:

- The priority effect is the one with the lowest priority level.
- The priority effect is the one using less number of slots, i.e., the smallest number of simultaneous samples (see Unitouch Studio).
- The priority effect is the most recent one, i.e., playing for the shortest amount of time.

Keep in mind that having too many effect playing simultaneously might:

- Make the actuators saturate and distort the rendered signals.
- Confuse the user as too many different sensations happen simultaneously.

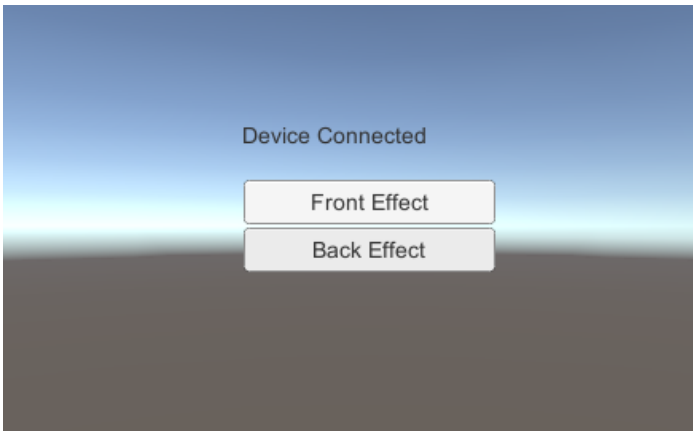
7. Errors & Logs

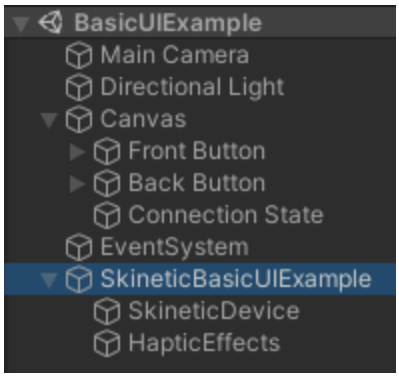
Most call the Skinetic SDK return negative error codes when issues arise, you can check the corresponding error using the static method `Skinetic.SkineticDevice.getSDKError`.
In addition the Skinetic SDK also log some information as it is running. The log can be found in the SkineticSDK.log file which is overridden after each session. On Android, the log are added to the **logcat** under SKINETICSDK.

8. Examples Walkthrough

8.1. Basic UI Example

This example is a basic “hello-world” example. In playmode, the device will automatically connect to any Skinetic device available, and update the connection status. Additionally two buttons are displayed in the middle of the screen, pressing one of them triggers a predefined haptic effect on the vest on the front (or back) of the haptic device.





The scene is composed of a canvas with two buttons and a text and an empty gameobject SkineticBasicUIExample containing two gameobjects: one to handle the **SkineticDevice** component and on to handle the **HapticEffects**.

- **Automatic connection**

The connection is automatically performed by the **SkiEx_AutoConnect** script on the SkineticDevice gameobejct. This script has a reference set in the inspector to the SkineticDevice component and a Text on the canvas.
On the OnEnable event:

- A connection callback is set:
  >> m_device.SetConnectionCallback(ConnectionCallback);
  This callback will be called whenever the connection to the device is made or broken, and update the state variable **m_isConnected.**
- A connection to any Skinetic device is requested by passing **Skinetic.SkineticDevice.OutputType.E_AUTODETECT** and **0** to allow any kind of connection and pick the first available device.

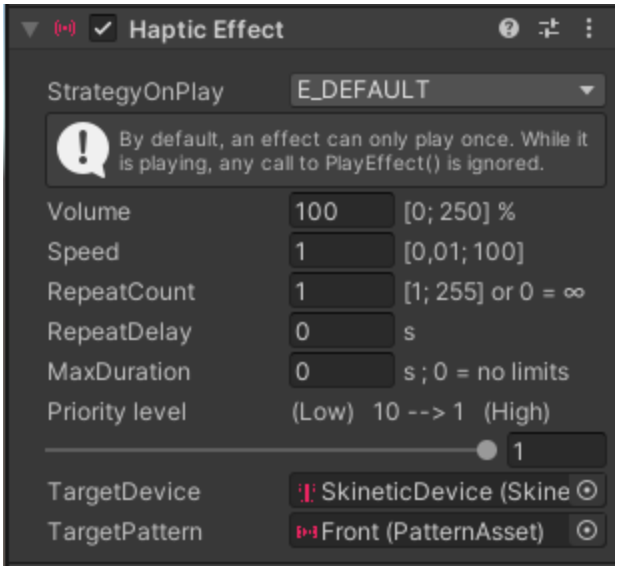
- Start a coroutine to update the connection status in the text field as **the callback being called from another thread cannot update the UI itself.**

During the OnDisable event, the disconnection routine is initiated and the function returns once the disconnection is completed.

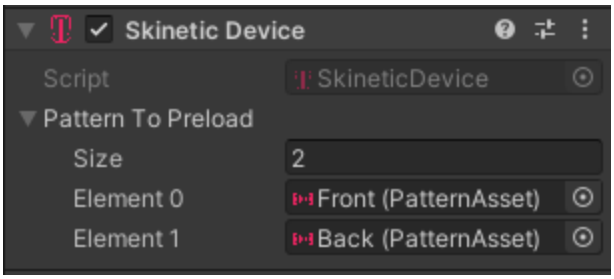
- **Basic Effects**

The **HapticEffects** gameobject contains 3 components:

- two preconfigured **HapticEffect** components whose parameters are predefined in the inspector and hold references to the **SkineticDevice** component in the scene and to the **PatternAsset** they have to use.
- a **SkiEx_BasicUIEffects** script which simply holds reference to the two **HapticEffect** and exposes 2 public functions registered in the inspector to each button's OnClick() event. These functions then call `PlayEffect` **from the HapticEffect** as the target device was set in the inspector.

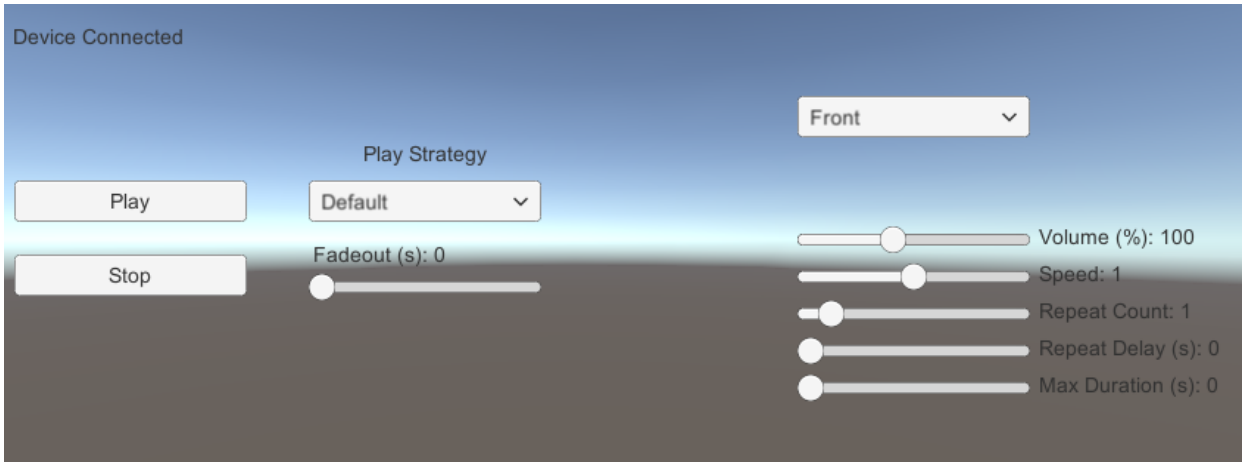


Additionally the two **PatternAssets** are added to the preload list of the **HapticDevice** so there is no need to register them through scripting as they will be loaded right at the beginning.



8.2. Editable Parameters Example

In this example the haptic effect parameters can be edited from a UI. In playmode, the device will automatically connect to any Skinetic device available, and update the connection status. Additionally the UI exposes 3 different **PatternAsset** that the user can select. Then, the user can change the parameters of the effect that are passed to the `PlayEffect` function.

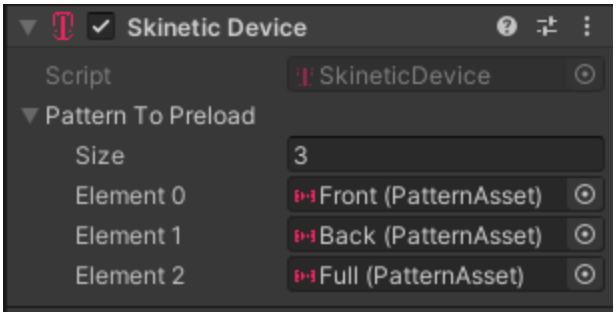


- **Automatic connection**

The automatic connection is the same as the Basic UI Example using the same **SkiEx_AutoConnect** script.

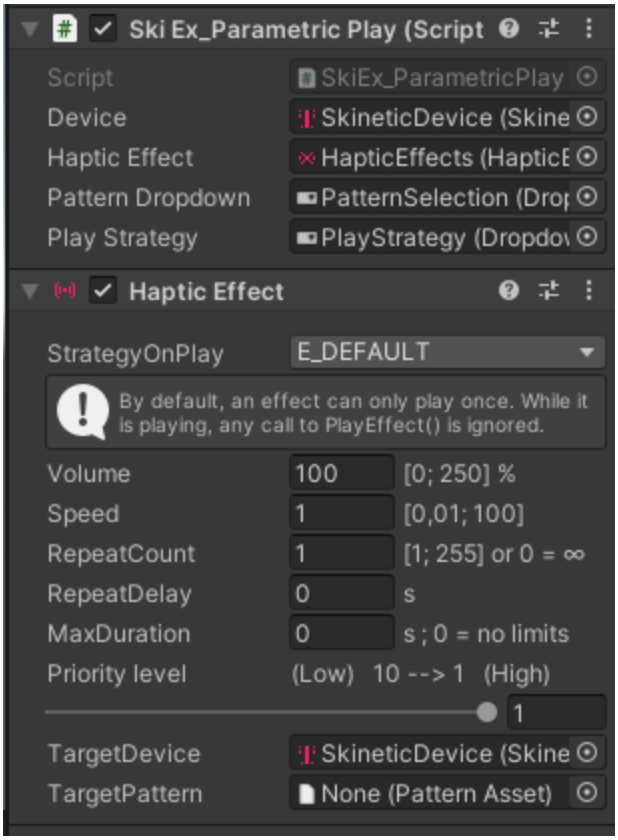
- **PatternAssets**

In this example, 3 **PatternAssets** are added to the preload list of the **SkineticDevice** component. After the OnEnable event, they will be loaded and accessible through **LoadedPatterns,** a read-only list of all loaded patterns available on the **SkineticDevice** component. The component then acts as a storage that another script will use.



- **Parametric Effects**

The **HapticEffects** gameobject possesses a single **HapicEffect** component and a **SkiEx_ParametricPlay** script that handles the UI elements.



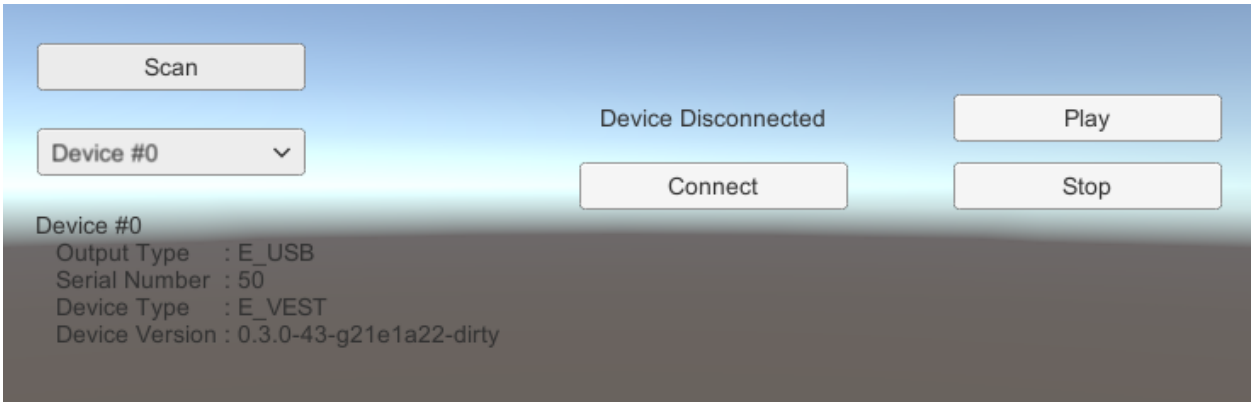
The **HapticEffect** component has a reference to the **SkineticDevice** as there is only one in the scene. However, the **TargetPattern** field is not set in the inspector as it will be set through scripting by the **SkiEx_ParametricPlay** script.

For each property of the haptic effect, the setter is registered to the **OnValueChanged** event of the corresponding slider. Except for the repeat count and the play strategy that are set through function of the **SkiEx_ParametricPlay** script as their type could not be directly handled by the UI's event.

Any modification of the **HapticEffect** properties is then taken into account on the next `PlayEffect`. In addition to the play button, a stop button is also added to call `StopEffect.`

8.3. Scan Device Example

In this example the user can trigger a scan of all available Skinetic devices and display the result of the scan. He can then select one of the devices and connect to it. Play and stop buttons enable to trigger an effect in order to test the connection.



- **Basic Effects**

The **HapticEffects** gameobject contains only a basic predetermined effect as to test the connection. The effect is handled and triggers with buttons just like in the previous example. The only difference is that the **PatternAsset** is not in the **Preload** list of the **SkineticDevice** component and is registered through scripting in the **SkiEx_BasicSingleEffect** script that handles the effect and the triggers.

- **Scanning Procedure**

The Scanning (and connection) procedure is done in the **SkiEx_DeviceScan** script. The scan is triggered by pressing on the button Scan which triggers the ScanDevices coroutine.

  - This coroutine first disconnects any device that might still be connected, as the scan can only be done if no device is connected. Since no callback was set in this example, the connection state is monitored by calling `ConnectionStatus` every 0.2s until the state **E_DISCONNECTED** is obtained.
  - Then the scanning procedure is triggered to find Skinetic devices using any kind of output type. Then every 0.2s the scanning procedure status is checked with `ScanStatus` until it returns 0 for a successful scan.
  - Finally the result is obtained with `GetScannedDevices`, the dropdown menu is updated with the available devices and the first one in the list is displayed.

- **Displaying Device Info**

The Scanning procedure returns a list of **Skinetic.SkineticDevice.DeviceInfo**. A dropdown menu allows the user to select one of the devices. By doing so, the device information is displayed in a text field: OutputType, Serial Number, Device Type, Device Version.

- **Connection Procedure**

As the user select a device from the scanned device list, he can connect to it by clicking on the Connect button which triggers the connection routine:

  - This coroutine first calls `Connect` by passing as argument the **output type** and the **serial number** of the selected device in the scan list.
  - As there is no connection callback set, the connection state is monitored by calling `ConnectionStatus` every 0.2s until the state **E_CONNECTED** or **E_DISCONNECTED** is obtained.