CRANK SOFTWARE

How to Improve Embedded GUI Development Efficiency Using a Collaborative Workflow

Best Practices Using Storyboard™

Vol. 2

Best Practices for Crank Storyboard

How to Improve Embedded GUI Development Efficiency Using a Collaborative Workflow

Taking advantage of all Crank Storyboard has to offer means understanding its workflows and deciding how to best fit them into your workday. While some of the practices presented here may be different than what you're used to, we firmly believe that the benefits of collaboration, rapid validation, and efficiency will be more than worth it.

At any time, we encourage you to visit our Help Center and video library for more detailed information on how to use Crank Storyboard.

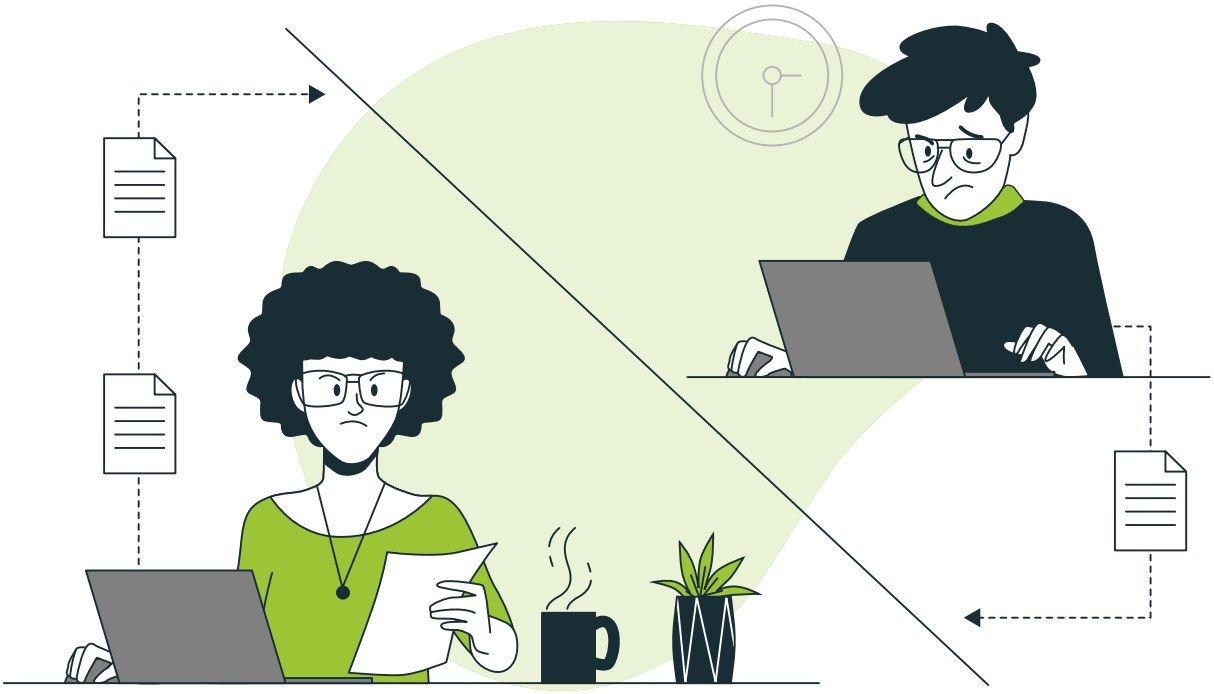
What you'll learn:

- Pitfalls of traditional embedded GUI development workflows
- Collaboration techniques between designers and developers
- Architecture of Storyboard applications
- Techniques for validating on real hardware
- Ways of verifying messages between the backend system and your GUI

The pitfalls of traditional development workflows for embedded GUIs

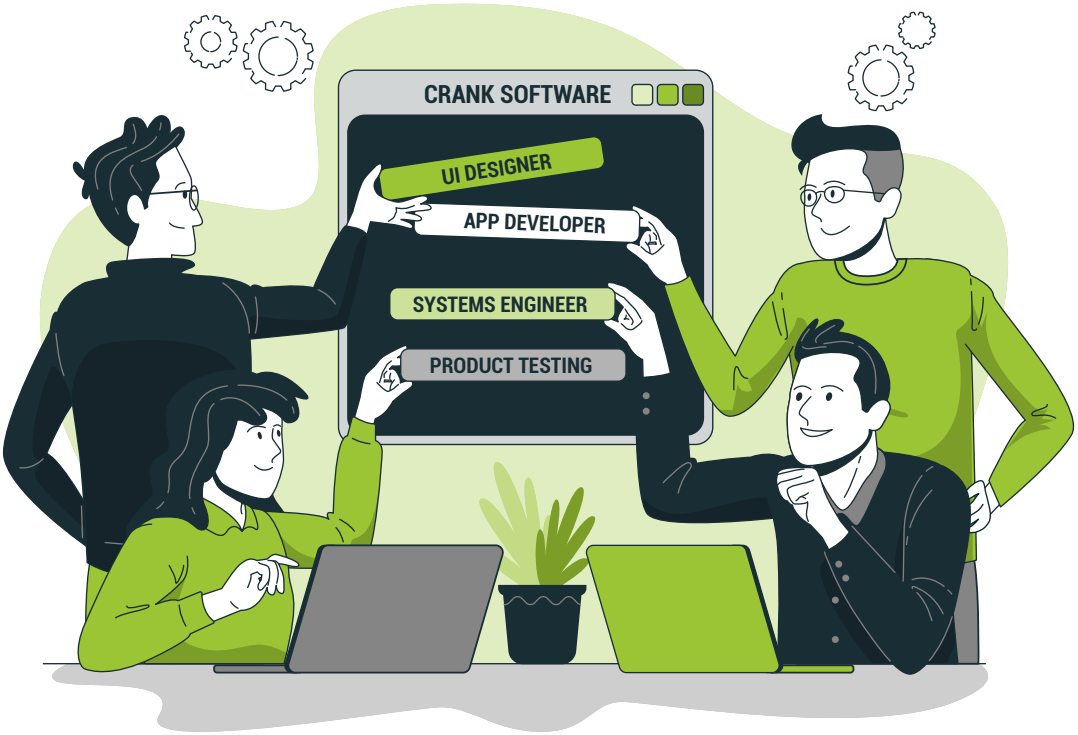
It makes sense to start by defining the traditional workflows performed by design and development teams and some of the pitfalls they encounter.

The process begins in the design stage, where designers create the look and feel of the GUI in isolation from the application developers and other project stakeholders (engineering, product managers, etc.). Once a high-fidelity design specification is complete, the stakeholders are engaged for feedback before GUI application development begins.



The problem with this workflow is that the user experience (UX) is designed without direct testing and feedback of the GUI application on hardware. This makes it challenging to uncover potential issues with the design, performance, or UX of the application until significant work has been put into development. The result is later-stage changes that are time-consuming and costly in terms of engineering and design effort.

What seems like a minor design change to one project stakeholder may actually be a significant exercise in tearing down and re-coding, leading to delays in getting the product to market or a less than optimal user experience.



If your GUI application is built using a development tool that allows for flexible and iterative workflows between designer, developer, and other stakeholders throughout the lifecycle, you can cut down on this linear feedback process and embrace changes at any time. All contributors can work together as early as possible to improve the product's GUI, validate it on the hardware, and ultimately benefit the end-user's experience.

Helping designers and developers work better together

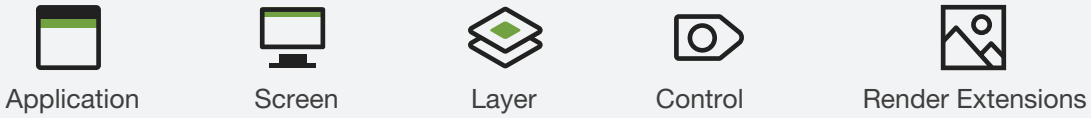
Storyboard is purpose-built to bridge the gap between design and development in their traditional GUI development workflows. It does so by enabling an open dialogue, or language, between both sides that begins in the early stages of design.

The goal of an optimal GUI development workflow isn't to force new processes onto designers and engineering groups, rather it's to create a way to translate design content seamlessly into development workflows. This is exactly what the Storyboard development framework sets out to provide.

Fostering collaboration between designers and developers requires some understanding of the Storyboard application structure, and the conventions Storyboard uses to import content from design files. By understanding how content fits into the Storyboard application architecture, it's easier to structure your design in a way that seamlessly imports into Storyboard.

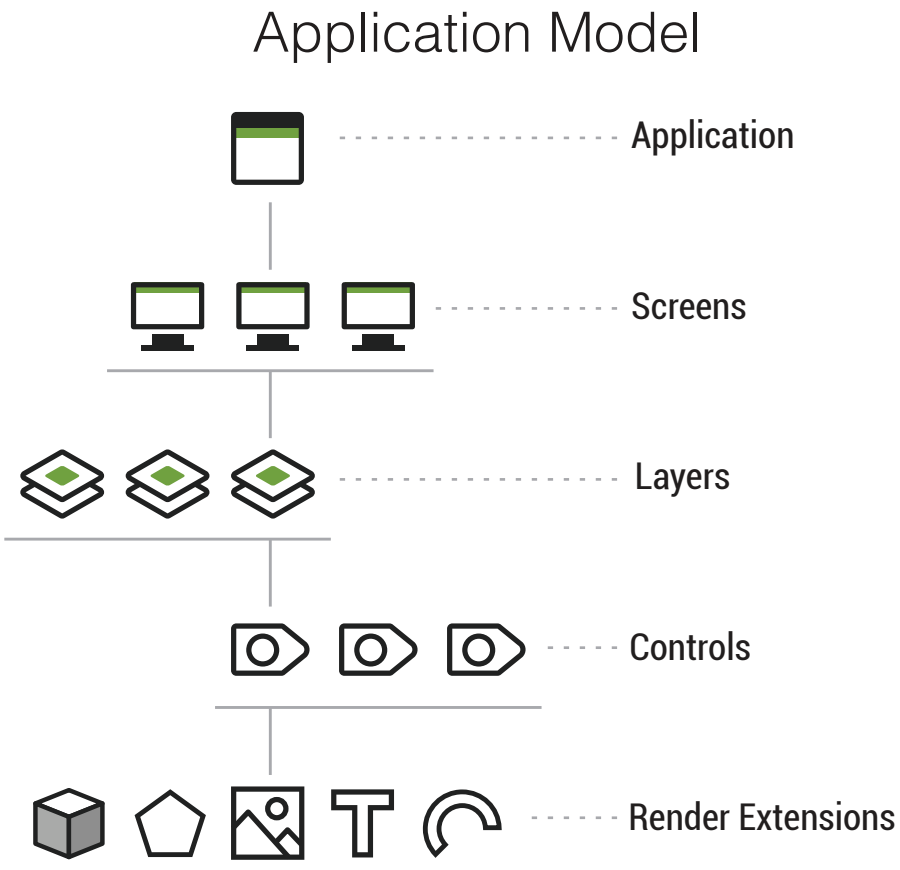
BEST PRACTICE

Understand the architecture of a Storyboard application and map design elements (screens, layers, controls) to this structure.



Picture a Storyboard-built GUI application like this, where the root node is the application and the branches of the tree are the main screens:

Here, "Application" contains information on the resolution, color depth, and other metadata relevant to the GUI.



Applications contain one screen at minimum and there's no limit as to how many you can have.

A **screen** must always be displayed and it contains content in the form of layers.

Layers are a finer-grained grouping of similar elements, such as navigation bars, popup dialogs, or the collection of elements that make up gauges. Each layer contains one or more controls that define where visual content (images, text, 3D objects, or a combination of them all) is positioned and displayed.

A **control** contains at least one render extension, used to perform the actual drawing of objects within the control's area. You can define these elements and mix and match them to create unique screens. Controls can be empty if you so choose.

A **render extension** defines the basic type of content that can be drawn, such as text, images, and rectangles.

[See our documentation and user guides to learn how to set up content within Photoshop or Sketch for a seamless import experience]

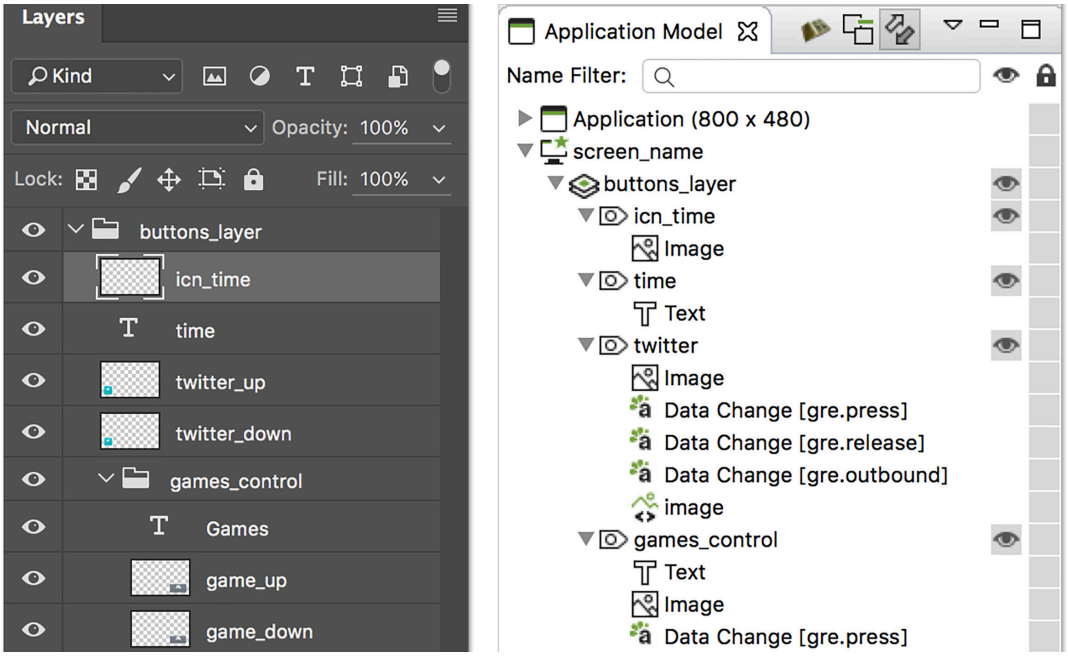
Having a strong understanding of the structure of a Storyboard GUI application is important for developers and designers alike. This application structure serves as a common understanding between both parties and allows them to think in a common way despite having different backgrounds and skill sets. Once the application architecture is understood, it's easy for GUI designers to start creating content within familiar design tools, such as Photoshop or Sketch, and importing it into Storyboard.

Once your application has been created, it's much easier to build up and structure content in a design tool for re-importing or adding additional content. A little knowledge means you can do this quickly and easily no matter where you are in your application development. We encourage you to do this frequently as it streamlines your feedback loop and provides opportunities to add behaviour to your design and test it out.

BEST PRACTICE

Use a consistent naming scheme and conventions for layers and assets in your design tool.

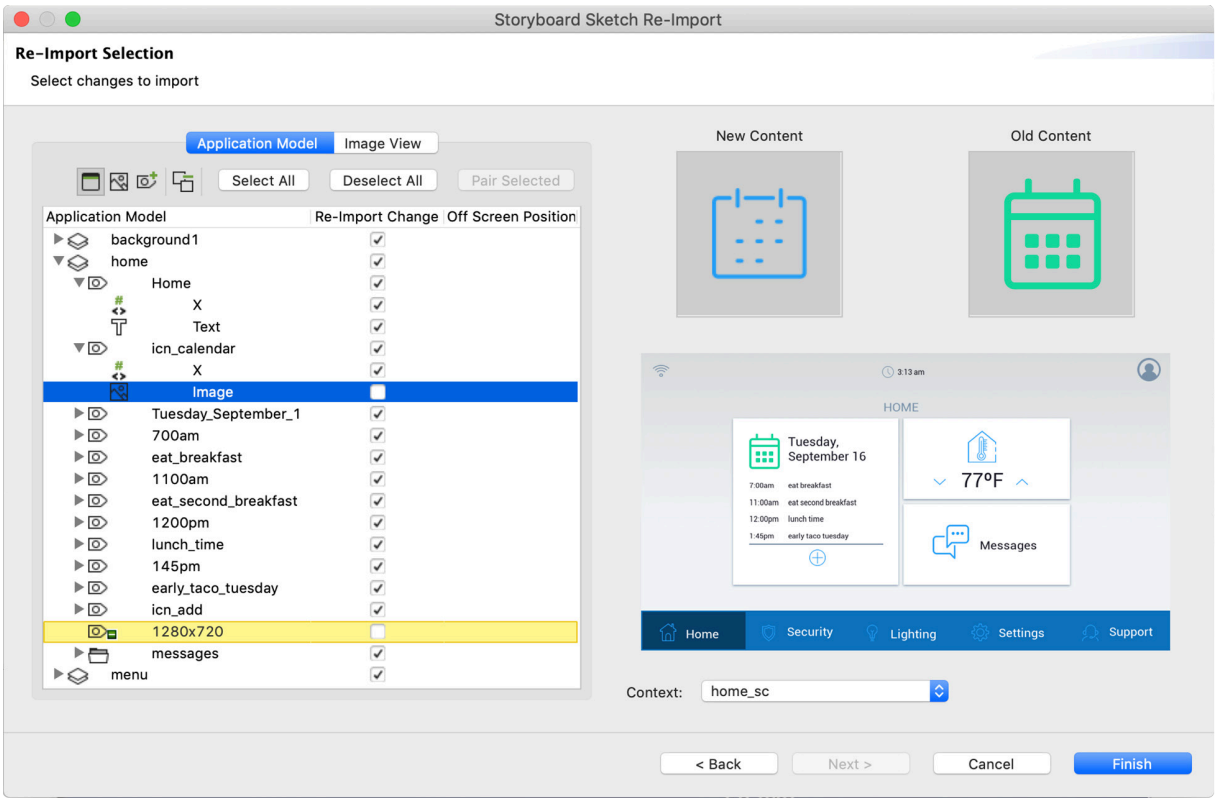
A best practice for maximizing the design import workflow is to develop a consistent naming schema and conventions for the layers and assets created with the design tool. This allows for logical mappings of elements and the bootstrapping of simple behaviors, such as button logic, to occur once



The naming and organisation of content in the Photoshop Layers panel on the left determines how content is named and structured using Storyboard's Photoshop Import feature. Learn More

imported into Storyboard. It also simplifies the design iteration process of importing new content into the project, in particular the re-import of changes to elements that already had behaviour defined.

We recommend that any changes to the names and structure of elements in your design should be done consciously and intentionally after the initial import into Storyboard — either in the tool or the design files. This will avoid breaking the mapping between existing application content and the design file content. For example, if you decide to retheme and rename a button called "confirm_btn" to "accept_button", you will need to map the new assets to the old controls during the import process if the intent is to replace the existing content.



Making significant changes to the design of the UI without impacting functionality of an application in development is possible with the Sketch and Photoshop Re-Import Feature. Learn More

BEST PRACTICE

Consider changes to the names and structures of design elements carefully, as this may break the mappings created during the initial import into Storyboard.

Additionally, using Storyboard's model Compare and Merge tools that link seamlessly with GIT, SVN, and other version control systems, developers can easily work in parallel to ensure that the UI can be modified, shared, and improved by multiple contributors. Any potential issues will be highlighted within the tool, allowing you to review, triage, and either accept or reject them. Storyboard allows for the testing of changes made to the GUI by simulating without the need to deploy to hardware.

BEST PRACTICE

Import and iterate the design frequently with Storyboard's Import and Re-Import features. Muliple users can leverage the Compare and Merge tool to work in parallel.

Validating changes and enhancements on target hardware

Regardless of how effective a simulator is, there is no better method than running it on the actual hardware to validate how the GUI application performs. Having a seamless method for deploying to hardware helps you validate changes more frequently, allowing you to detect any potential issues earlier, resulting in time and effort saved.

To test as early as possible, and with the least impact on the release cycle, start validation on hardware when you have a skeleton for the GUI application in place or have reached a point where animations are ready to be added. We recommend this because development machines are more powerful and always have access to a GPU, whereas the intended embedded hardware may not.

BEST PRACTICE

Test and validate your GUI frequently on the target hardware.

To prepare your application for deployment to hardware, you must first configure a resource export configuration for the project that is used in conjunction with the export packager:

- This configuration contains information on what assets will be exported and in which format. This configuration only needs to be done once (per target configuration) and adjusted only when changes to the platform are made.
- The export packager allows you to choose exporting to an MPU platform, MCU platform, or a mobile device. Each packager has its own nuances that are covered in detail in the Storyboard documentation.

When working on smaller embedded platforms, such as microcontrollers, you need to rebuild and flash the system based on the exporter output. To help streamline the export process, it's recommended to export the application to the RTOS project location, allowing quicker rebuilds to be performed.

RESOURCES

Exporting to your target configuration?

The Storyboard Resource Export Configuration editor provides the ability to create configurations for how resources should be exported from Designer.

Storyboard Resource Export Configuration Editor

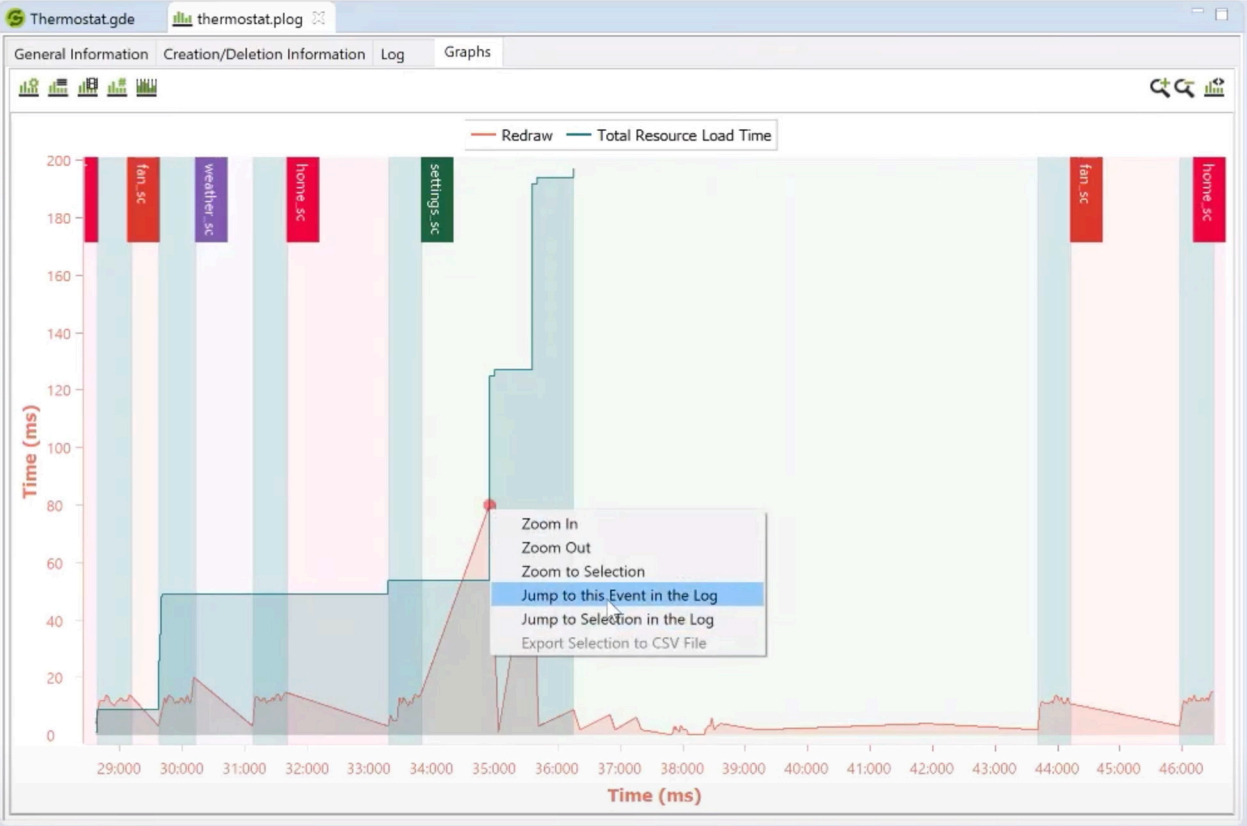
Take greater control over how and when resources should be exported from the design environment.

LEARN MORE

When working with a microprocessor, you can use the Secure Copy Protocol (SCP) to transfer the GUI application over the network and have it configured to run automatically. This process of deploying the application to hardware for testing and validation is as fast and simple as a few mouse clicks.

BEST PRACTICE

When working with a microprocessor, use the SCP protocol for the automatic transfer and run of your GUI application on the target.



Performance logging is a tracing and recording mechanism built into the Storyboard engine, allowing the capture of detailed execution information about operations performed within the engine. Learn More

Another benefit of exporting the configured application quickly, is that Storyboard allows you to collect real metrics on performance and resource consumption. When working with devices that have limited memory, such as a single-core MCU device, early awareness of the application's resource footprint and performance is critical. This is especially important when you consider the traditional alternative of sifting through lines of code of a completed GUI application to isolate and understand a performance bottleneck.

Storyboard's performance logs, which contain the data surrounding the performance of your application, can be extremely helpful. Once generated, they can be imported into Storyboard and broken down into multiple sections, such as rendering performance, event flow, and the creation and deletion of elements, to make the process of fine tuning the application's performance a whole lot easier.

BEST PRACTICE

Use the Storyboard performance logs to baseline and optimize GUI resource footprint and performance metrics.

Frequent testing on your hardware allows you to validate your application as it's being developed and gather real metrics when you need to understand what's happening in Storyboard. Making this is a part of your day to day workflow ensures you'll end up with a solid product and efficient development processes.

Verifying system messages to and from the GUI

Once the layout of the user interface, animations, transitions, and event interactions are complete, it's time to inject real data into the application. Similar to the benefits of testing on actual hardware, injecting simulated events into the GUI application early in development helps ensure it performs as intended in the real-world (simulated or live) before changes become too costly.

Even if the backend application responsible for feeding data into your GUI is not available, all you require are the events and their data structures to start injecting. By adding them to the Storyboard project, these event definitions are used to generate a Storyboard IO event header file that's used by the backend application. This creates the basis for the application programming interface (API), or data contract, between your frontend GUI and the backend logic. This keeps the data and event flow consistent and allows you to reduce the integration effort when connecting your GUI to the real backend for the first time.

BEST PRACTICE

Define events and their data structures early and use Storyboard Connector to inject them into your application for testing.

Another benefit of defining the event structures is that they can be used to simulate data from the Storyboard Connector. Storyboard Connector allows simulation data to be generated and injected into a running application on your desktop or embedded hardware. This helps you stress test the application before the live data source is ready and provides the development team with a clearer understanding of how the GUI will perform when it's added to the embedded product.

RESOURCES

How to connect data with events.

Learn how to use data-driven events to update your embedded GUI project and export your events to a C/C++ Header File for system engineers and developers to use.

Using the Storyboard IO Connector

The Event Editor is used to add and edit event definitions. Learn More

Adding an Event Definition

Learn about the different ways new events are added. Learn More

Developing an efficient and collaborative GUI development workflow

While the tools and features available within Storyboard are powerful, they are only helpful when deliberately planned and used intentionally. Discipline and consistency are two key pieces of creating powerful and efficient development workflows.

By leveraging the best practices discussed in this document you will have the flexibility to speed up GUI development, adapt to changes in design, and improve your application validation cycle.

Best Practices Summarized

1.	Understand the architecture of a Storyboard application and map design elements (screens, layers, controls) to this structure.
2.	Use a consistent naming scheme and conventions for layers and assets in your design tool.
3.	Consider changes to the names and structures of design elements carefully, as this may break the mappings created during the initial import into Storyboard.
4.	Import and iterate the design frequently with Storyboard's Import and Re-Import features. Muliple users can leverage the Compare and Merge tool to work in parallel.
5.	Test and validate your GUI frequently on the target hardware.
6.	When working with a microprocessor, use the SCP protocol for the automatic transfer and run of your GUI application on the target.
7.	Use the Storyboard performance logs to baseline and optimize GUI resource footprint and performance metrics.
8.	Define events and their data structures early and use Storyboard Connector to inject them into your application for testing.