

**A10 GUI Widgets**

***Engineering Design Document***

Version 1.0a

2018/01/10

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Author / Editor | Revision | Date | Comments |
| Steven Huang / Rui Zhang | 0.1 | 12/29/2017 | Initial draft |
| Steven Huang / Rui Zhang | 0.2 | 1/2/2018 | Initial draft |
| Steven Huang / Rui Zhang | 0.3 | 1/4/2018 | Initial draft |
| Steven Huang / Rui Zhang | 0.4 | 1/5/2018 | Initial draft |
| Zuoping Li / Peter Chuang / Steven Huang / Rui Zhang | 0.5 | 1/6/2018 | Initial draft |
| Peter Chuang | 0.6 | 1/6/2018 | Peter’s edits |
| Peter Chuang/  Zuoping | 1.0a | 1/10/2018 | First alpha release |

Contents

[Feature Summary 4](#_Toc503365869)

[The Role of A10 GUI Widgets 4](#_Toc503365870)

[Definitions of Widgets 5](#_Toc503365871)

[Structure of the Widget System 5](#_Toc503365872)

[Definition of Objects 6](#_Toc503365873)

[Characteristics of a A10Widget 7](#_Toc503365874)

[Difference between Presentational Components and Containers 7](#_Toc503365875)

[UML Diagram 8](#_Toc503365876)

[Definition of Objects 9](#_Toc503365877)

[Design Patterns Used 11](#_Toc503365878)

[Adapter Pattern 12](#_Toc503365879)

[Decorator Pattern 13](#_Toc503365880)

[Higher Order Components technique 14](#_Toc503365881)

[Command Pattern 16](#_Toc503365882)

[Future Enhancement (Not in the first release) 16](#_Toc503365883)

[Widget Theme 16](#_Toc503365884)

[Widget Locale 17](#_Toc503365885)

[References 17](#_Toc503365886)

# Feature Summary

A10 GUI Widgets (or sometimes referred to as “A10Widgets” or simply “Widgets” in this document) is dedicated to providing a set of shared and reusable Widget libraries that are designed to be widely used across different A10 applications. It is based on the A10 GUI Framework to maximize modularity and maintainability. The aims of the Widget and Framework together are to provide a good development experience for A10 and possibly outside contributing developers; as a result, make A10 products powerful and useful. This document captures the definition, design, project file structure, and examples to elaborate the concepts of Widgets.

# The Role of A10 GUI Widgets

A10 GUI consists of 5 layers:

1. A10 GUI Framework
2. A10 GUI Widgets: What this document covers.
3. A10 Common modules: AutoConfig, WAF, ...etc.
4. A10 Infra: Thunder, Harmony, or aGalaxy
5. A10 Apps: ACOS GUI, Harmony GUI, Object Explorer, and aGalaxy.

Figure 1 shows the structure of A10 GUI system. A10 GUI Widgets is based on A10 GUI Framework and is the library used by the common, Infra and App layers.

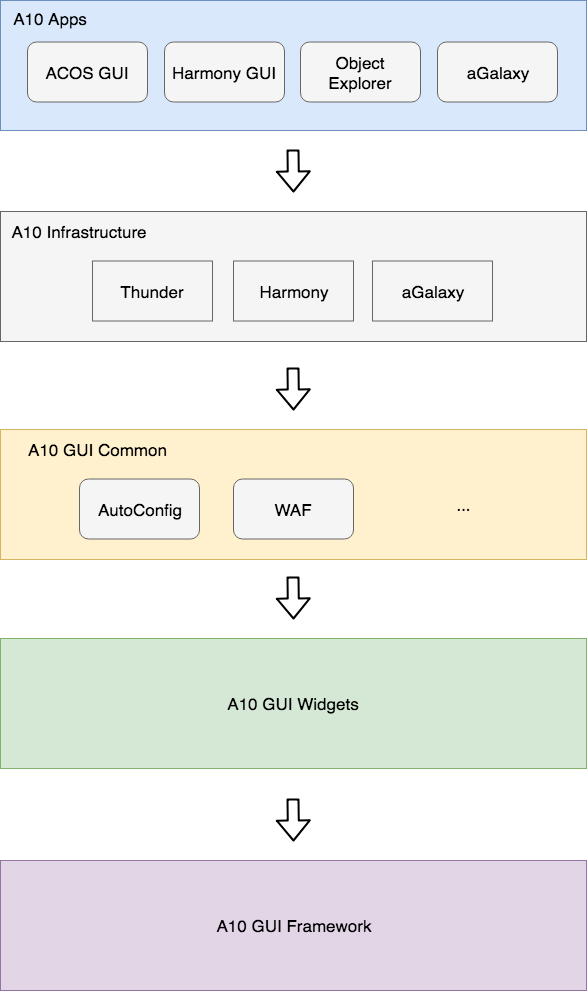


Figure 1: The structure of A10 GUI

# Definitions of Widgets

The shared and common presentation components are called Widgets. Widgets are stateless and are mainly concerned with how things look. Stateless here means that it has nothing to do with data or business logic.

## Structure of the Widget System

Figure 2 is the top-level design of the Widgets system. It illustrates that both Basic and Composite Widgets are created with the same interface through the same Widget wrapper. The A10 Widget wrapper is used to have uniform and consistent A10 behavior across different Widgets.

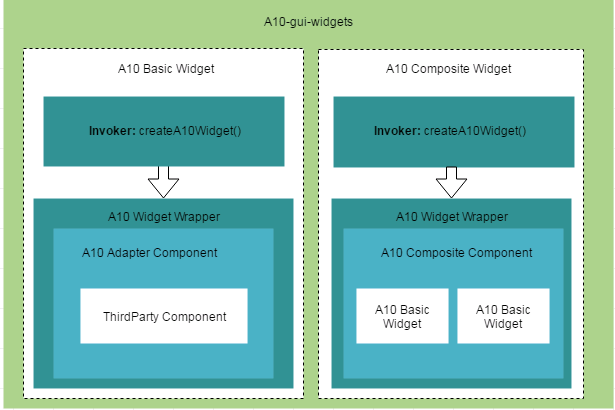
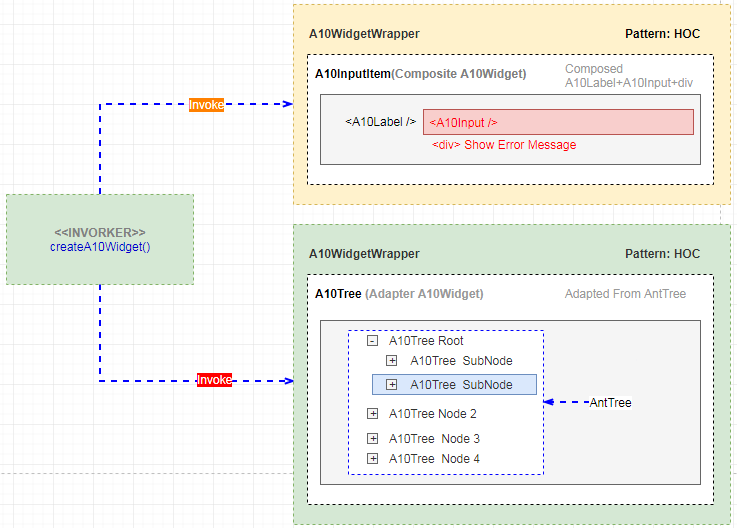


Figure 1 Widget Design: Top-level Design

Figure 2 Real case

### Definition of Objects

Let’s go through each object from top to bottom and left to right in Figure 3.

1. A10-gui-widget: The name of the A10 GUI Widget library in a repository. It aims to host source code of reusable Widgets for A10 developers. Any A10 developer may contribute his/her Widget into the shared library.
2. A10 Basic Widget: The fundamental Widget. Widgets may be derived from any existing library like an open source library made by others. Some example A10 Basic Widgets are: form, tree, search, context menu, tooltip, etc.
3. A10 Composite Widget: An A10 Composite Widget is composed of multiple A10 Basic Widgets. An example A10 Composite Widget is a tree-list Widget made from Basic Widgets like tree, context menu, and tooltip.
4. Invoker: The invoker provides createA10Widget() method.A10WidgetWrapper: Our design allows for the flexibility that other functionalities may be added in the future. Initially, however, four of React Component’s life cycle methods are overridden in A10 Widget Wrapper to provide basic logging:
5. A10WidgetWrapper: Our design allows for the flexibility that other functionalities may be added in the future. Initially, however, four of React Component’s life cycle methods are overridden in A10 Widget Wrapper to provide basic logging:
   1. componentWillMount()
   2. ComponentWillUnmount()
   3. componentDidMount()
   4. ComponentDidUpdate()
6. ThirdParty Component: The original implementation of the component. It could be the third-party component, such as Ant Design library. This ThirdParty Component is left unchanged.
7. A10 Composite Component: A construct for aggregating and organizing multiple A10 Basic Widgets as a single unit.

## Characteristics of a A10Widget

The typical characteristics of a Widget are described below:

1. It may contain other presentation component and container inside and usually has view, which contains DOM markups and associated CSS style.
2. It has no dependencies on the rest of the App.
3. It does not need to know how data is loaded or mutated (handled by the container).
4. It rarely has any state; if it does, it will be UI state rather than data state.

# UML Diagram

Figure 3 is the UML diagram that shows relationships of ingredients in a Basic Widget. Figure 4 is taking the Form as an example to represent an instance of figure 3. The definition and explanations of each object are listed below.

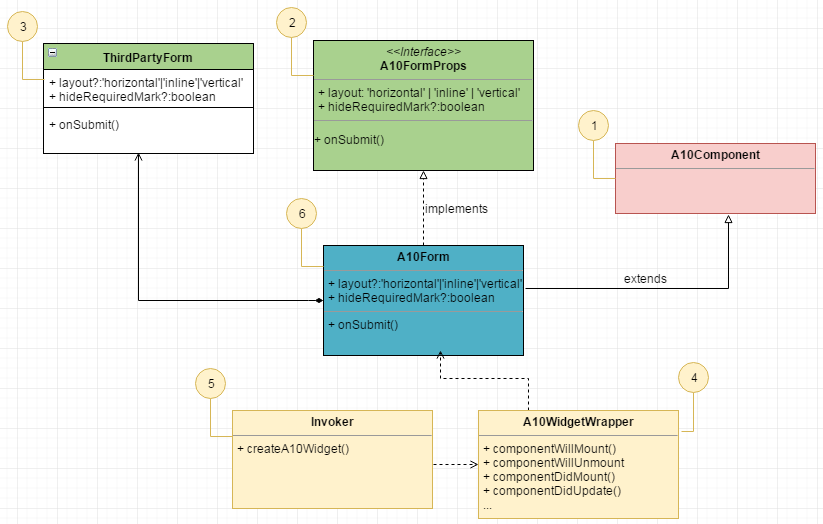


Figure 3 The UML of Widget structure

## Definition of Objects

1. A10Component: This is the base class for every A10Widget. A Widget could directly extend the A10Component.
2. A10WidgetProps: The interface for A10Widget. Every Widget has specific props that need be implemented.
3. ThirdPartyComponent: A component derived from third-party open source or commercial library.
4. A10WidgetWrapper: Our design allows for the flexibility that other functionalities may be added in the future. Initially, however, four of React Component’s life cycle methods are overridden in A10 Widget Wrapper to provide basic logging:
   1. componentWillMount()
   2. componentWillUnmount()
   3. componentDidMount()
   4. componentDidUpdate()
5. Invoker: The invoker provides createA10Widget() method to inject some common behavior.
6. A10Widget:
7. Could be composed of other A10Components.
8. Implements the A10WidgetProps interface.
9. Is composed of some selected properties (props) and methods from ThirdPartyComponent through Adapter pattern.
10. Injects extended functionalities through invoker with Command pattern and A10WidgetWrapper through Decorator pattern.

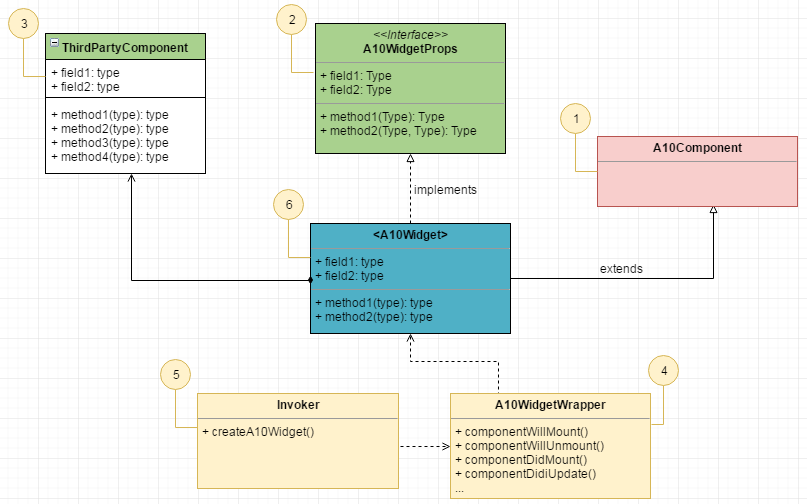


Figure 4 The example UML of Widget structure with Form component

# Design Patterns Used

Figure 5 shows the design patterns being used in a Widget. The Adapter, Decorator, and Commander patterns are applied. Detail explanation is described below.

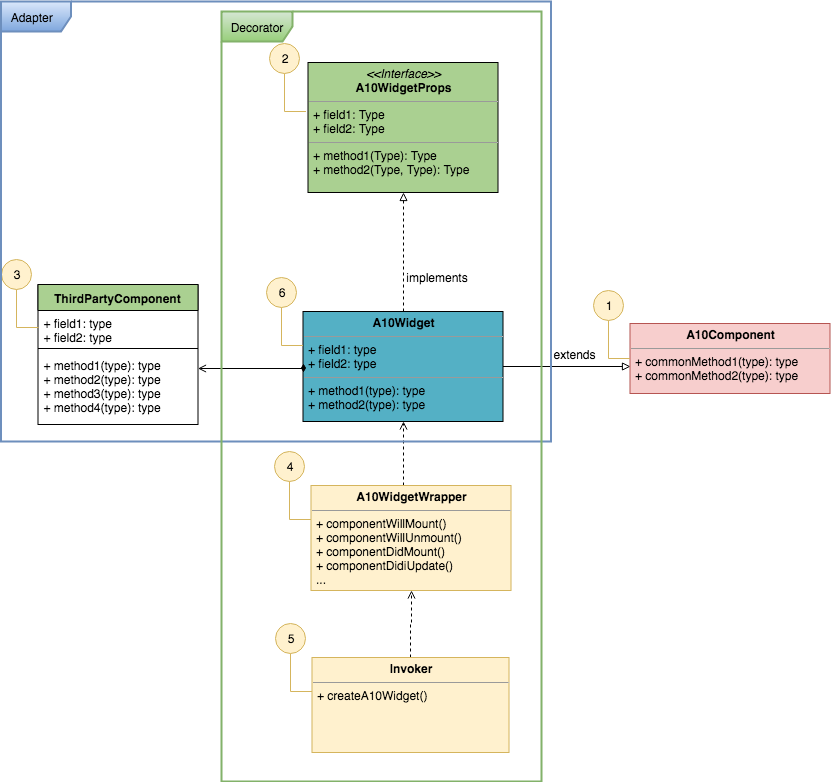


Figure 5 Design patterns of used in A10Widget

## Adapter Pattern

When a Widget is being created, the ThirdPartyComponent, derived from open source library, will be adapted through A10Widget as shown in figure 6. As a result, some selected properties and methods from ThirdPartyComponent will be further defined through A10Widget . Those props which are not selected will be imported directly without any change. In Figure 6, method3 and method4 from ThirdPartyComponent are such props that will be imported directly without any change. Benefits of this Adapter approach:

1. Allowing the interface of the ThirdPartyComponent to be used in A10WidgetsWrapper.
2. There is no need to modify ThirdPartyComponent when interacting with A10WidgetWrapper.

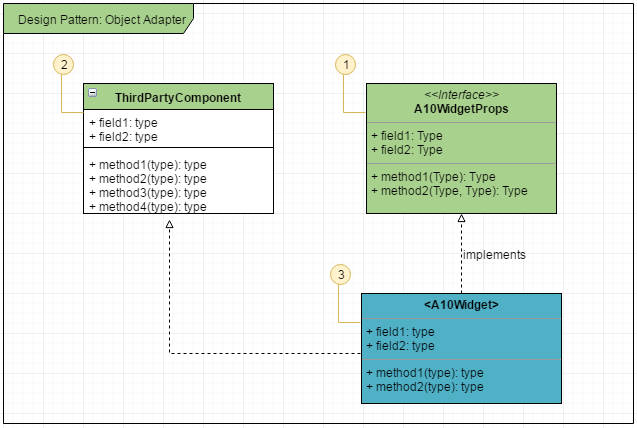


Figure 6 The UML of Design Pattern: Object Adapter

## Decorator Pattern

A10WidgetWrapper adopts the Decorators design pattern. It allows us to enhance a Widget by composition between classes rather than by inheritance of a base class. Benefits of this decorator approach:

1. Allowing functionality to be added to A10Widget, either statically or dynamically without coupling.
2. The original behavior of A10Widget is not affected at all. A10Widget still has all of its own behavior. At the same time, new functionality is added without changing the class which is being decorated.

Decorator allow us to add additional methods to the existing A10Widget . These four methods that currently support logging are added:

* 1. componentWillMount()
  2. componentWillUnmount()
  3. componentDidMount()
  4. componentDidUpdate()

Taking the figure 7 as an example, we are decorating the A10Widget with A10WidgetWrapper. After decorating, the A10Widget still has the original methods (method1, method2) unchanged. And new methods from A10WidgetWrapper will be injected into the A10Widget as well.

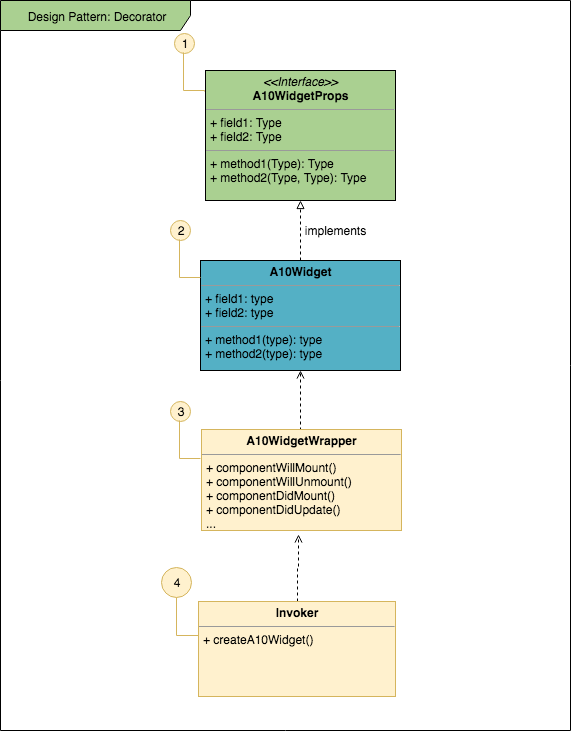


Figure 7 The UML of design pattern: decorator in a Widget

### Higher Order Components technique

It’s worth mentioning that the technique of Higher Order Components (HOCs) is also frequently used in our system. A HOC is a pure function that takes an old component and returns a new component. HOCs in React are very similar to Decorators in object-oriented programming. Benefits of this HOCs approach:

1. Reusing component logic: We could define and share the same logic across many components through abstraction in a single place.
2. Control over Inputs: Using HOC, we can have control over the inputs passed to the composed component. We can restrict any input, mask it or transform it even.
3. Establish Connection to a Store/API Service: If there is a generic store/ API service like Analytics or A/B which we want multiple components to be connected to, we can do that using HOCs.
4. Intercept rendering: It can be used in places where we want to intercept the rendering method. For example: If we are creating a HOC for managing Access to a certain component, we can disable Render, partial render etc. for that component according to Access permissions from some store.

## Widget Theme

Following the framework design, we need to define the widget theme under the folder of a Widget. For example, Form/style/index.less, in this file, we could import framework defined less variables and functions.

## Widget Locale

Following the framework design, all widgets show the corresponding language as long as setting up language in app code by props.a10Context.changeLocale() provided from a10-gui-framework.

# References

<https://ant.design/components/form/>

<https://ant.design/components/button/>

<https://reactjs.org/docs/higher-order-components.html>

[https://medium.com/@mappmechanic/react-utility-higher-order-components-as-Decorators-tc39-stage-2-9e9f3a17688a](https://medium.com/@mappmechanic/react-utility-higher-order-components-as-decorators-tc39-stage-2-9e9f3a17688a)