

|  |
| --- |
|  |

#

# SUMMARY

Providing a default scrollable for unhandled ScrollActions

**Author: Kate Lovett (@Piinks)**

**Go Link: flutter.dev/go/default-scroll-action**

**Created:** 11/2020 / **Last updated:** 11/2020

Original proposal below, updated with new proposal 11/5/2020

WIP PR: <https://github.com/flutter/flutter/pull/69795>

# OBJECTIVE

For desktop and web applications, there is an expectation that common keyboard shortcuts like page up/down and arrow keys will carry out a scroll action without explicitly focusing on the main scrollable content.

# BACKGROUND

Prior to being invoked, a ScrollAction will first check if there is a Scrollable above the context of the current primaryFocus. If one is found, the Action will scroll the Scrollable by the amount configured in the ScrollIntent given to it.

This requires that the primaryFocus is currently below a given Scrollable in order for the ScrollAction to work. When a scrollable cannot be found, no action is completed. This causes an issue for developers that expect a keyboard shortcut to make a scroll view move by default without setting focus below it in their application.

If focus has not been requested elsewhere, upon execution of a simple application containing a ListView, the primaryFocus will be on the \_ModalScopeState by default. This is above any Scrollable that a user may have created in their application, and so when a ScrollAction is invoked, none is found. We want to enable users to set a default, or fallback, scrollable that will take a ScrollAction if it is not handled within the context of the current primaryFocus.

### Glossary

* **primaryFocus** - the current primaryFocus of the FocusManager.
* **ScrollAction** - an Action that scrolls the Scrollable that encloses the current primaryFocus by the amount configured in the ScrollIntent given to it.
* **Shortcuts** - a widget that establishes an ShortcutManager to be used by its descendants when invoking an Action via a keyboard key combination that maps to an Intent.
* **Intent** - a class for containing a description of a user action to be invoked.
* **Action** - a class for defining an invocation of a user action.
* **ScrollIntent** - an Intent that represents scrolling the nearest scrollable by an amount appropriate for the type specified.

# OVERVIEW

In order to provide a default scrolling Shortcut, we want to place a Scrollable context above the \_ModalScopeState that serves as a fallback for unhandled ScrollActions. This will result in the ScrollAction.isEnabled finding a valid Scrollable to invoke the Action on in the case that no other context is found. Even if the primaryFocus is elsewhere (below \_ModalScopeState), this default scrolling context will be available as a fallback.

To accomplish this, I would like to expand the PrimaryScrollController to serve as the fallback Scrollable reference. ScrollActions access the ScrollableState in order to invoke the action, and the same can be achieved with the PrimaryScrollController through the ScrollContext of the position attached to the controller.

The PrimaryScrollController is already used to facilitate scroll events and fallbacks like when animating after tapping the status bar in a Scaffold, or serving as the fallback controller in the CupertinoScrollbar, so using this existing inherited widget to create default ScrollActions seems to be a nice evolution of the existing constructs.

Asking users to wire up their own scroll views by providing keys or scroll controllers felt kludgy and bug-prone in earlier proposals. In a key based approach, the key needs to be below the \_ScrollableScope in order to access it, which seems unnecessarily complicated to wire up on your own. If you place a key in the scrollable content, it may be disposed of as you scroll. Setting the key to ListView.key or others in this way would not find the correct context.

A cleaner approach would be to allow users to specify a scrollable for default ScrollActions through a flag, and correctly handle the key privately. This can be done in a similar fashion to that of the ScrollNotification context.

We currently get the notificationContext by using the \_gestureDetectorKey that is placed below the \_ScrollableScope. From [ScrollableState.build](https://github.com/flutter/flutter/blob/9ec5d7131bfc91a52b42ee9a09866ddb6f1396fd/packages/flutter/lib/src/widgets/scrollable.dart#L666):

|  |
| --- |
| // \_ScrollableScope must be placed above the BuildContext returned by notificationContext// so that we can get this ScrollableState by doing the following://// ScrollNotification notification;// Scrollable.of(notification.context)//// Since notificationContext is pointing to \_gestureDetectorKey.context, \_ScrollableScope// must be placed above the widget using it: RawGestureDetector |

We can follow this existing convention by accessing the \_ScrollableScope in this way for a fallback ScrollAction. Instead of asking users to manage keys or their own scroll controllers, we can use the existing primary flag. This flag already indicates that a scroll view should use the PrimaryScrollController, and it can further indicate that the given Scrollable should be a fallback for a ScrollAction if it has not been handled.

From there, we can add consideration in ScrollAction.isEnabled for the PrimaryScrollController, looking up the tree to see if one is available.

In order to accomplish this, we will need to place a PrimaryScrollController higher up the tree than we already do. Currently, PrimaryScrollControllers are placed in Scaffolds and CupertinoPageScaffolds for the status bar animation. <https://github.com/flutter/flutter/pull/69795> relocates the default PrimaryScrollController to higher in the tree, where we might want to create one for each route. That way each route can have its own PrimaryScrollController and connected scroll view that will be used as a fallback for ScrollActions.

# OPEN QUESTIONS

* Can we alert users to a PrimaryScrollController conflict?
	+ Switching the defaultScrollAction should be allowed, but we might want to consider how to let users know that only one Scrollable can be registered as the DefaultScrollAction at a time. This could be covered by extensive documentation, but the ability to assert this kind of case would be nice.
* If the user instantiates their own PrimaryScrollController that controls the scroll view they intend for default ScrollActions elsewhere in the application, this will break since the scroll view will be connected with the lower PrimaryScrollController rather than the root.

Original Proposal

WIP PR: <https://github.com/flutter/flutter/pull/69616>

# OBJECTIVE

For desktop and web applications, there is an expectation that common keyboard shortcuts like page up/down and arrow keys will carry out a scroll action without explicitly focusing on the main scrollable content.

# BACKGROUND

Prior to being invoked, a ScrollAction will first check if there is a Scrollable above the context of the current primaryFocus. If one is found, the Action will scroll the Scrollable by the amount configured in the ScrollIntent given to it.

This requires that the primaryFocus is currently below a given Scrollable in order for the ScrollAction to work. When a scrollable cannot be found, no action is completed. This causes an issue for developers that expect a keyboard shortcut to make a scroll view move by default without setting focus below it in their application.

If focus has not been requested elsewhere, upon execution of a simple application containing a ListView, the primaryFocus will be on the \_ModalScopeState. This is above any Scrollable that a user may have created in their application, and so when a ScrollAction is invoked, none is found. We want to enable users to set a default, or fallback, scrollable that will take a ScrollAction if it is not handled within the context of the current primaryFocus.

### Glossary

* **primaryFocus** - the current primaryFocus of the FocusManager.
* **ScrollAction** - an Action that scrolls the Scrollable that encloses the current primaryFocus by the amount configured in the ScrollIntent given to it.
* **Shortcuts** - a widget that establishes an ShortcutManager to be used by its descendants when invoking an Action via a keyboard key combination that maps to an Intent.
* **Intent** - a class for containing a description of a user action to be invoked.
* **Action** - a class for defining an invocation of a user action.
* **ScrollIntent** - an Intent that represents scrolling the nearest scrollable by an amount appropriate for the type specified.

# OVERVIEW

In order to provide a default scrolling Shortcut, we want to place a Scrollable context above the \_ModalScopeState that serves as a default focus. This will result in the ScrollAction.isEnabled finding a valid Scrollable to invoke the Action on in the case that no other context is found. Even if the primaryFocus is elsewhere (below \_ModalScopeState), this default scrolling context will be available as a fallback.

To accomplish this, I would like to create a DefaultScrollAction, an inheritedWidget that will maintain a key that is enclosed in a Scrollable. ScrollActions access the ScrollableState in order to invoke the action, so using a key will allow us to access the currentState of the default scrollable.

Asking users to set this up by providing their own key felt kludgy and bug-prone. This key needs to be below the \_ScrollableScope in order to access it, which seemed unnecessarily complicated to wire up on your own. If you place a key in the scrollable content, it may be disposed of as you scroll. Setting the key to ListView.key or others in this way would not find the correct context.

A cleaner approach would be to allow users to specify a scrollable for default ScrollActions through a flag, and correctly handle the key privately. This can be done in a similar fashion to that of the ScrollNotification context.

We currently get the notificationContext by using the \_gestureDetectorKey that is placed below the \_ScrollableScope. From [ScrollableState.build](https://github.com/flutter/flutter/blob/9ec5d7131bfc91a52b42ee9a09866ddb6f1396fd/packages/flutter/lib/src/widgets/scrollable.dart#L666):

|  |
| --- |
| // \_ScrollableScope must be placed above the BuildContext returned by notificationContext// so that we can get this ScrollableState by doing the following://// ScrollNotification notification;// Scrollable.of(notification.context)//// Since notificationContext is pointing to \_gestureDetectorKey.context, \_ScrollableScope// must be placed above the widget using it: RawGestureDetector |

We can follow this existing convention by accessing the \_ScrollableScope in this way for a DefaultScrollAction. Instead of asking users to manage keys, we can add a flag to Scrollables, isDefaultScrollAction. This flag will indicate that the given Scrollable should connect with the DefaultScrollAction higher in the tree to receive a ScrollAction if it has not been handled. Setting a flag seemed a cleaner, and simpler approach for users, and in managing the keys ourselves, we can leave most of the implementation private.

In <https://github.com/flutter/flutter/pull/69616>, this is proposed in Scrollable.didChangeDependencies:

|  |
| --- |
| if (widget.isDefaultScrollAction) { final \_DefaultScrollActionState? defaultScrollAction =  DefaultScrollAction.maybeOf(context); // Similar to notificationContext, \_ScrollableScope is placed above the // widget using it: RawGestureDetector, which allows us to access this // ScrollableState defaultScrollAction?.\_defaultScrollKey = \_gestureDetectorKey;} |

From there, we can add consideration in ScrollAction.isEnabled for the DefaultScrollAction, looking for this key if it is available.

# OPEN QUESTIONS

* Should this be an independent widget?
	+ Originally, the idea for this feature was tied into Scrollbars. Scrollbars are not always in use, so I went an independent route for the feature instead of integrating with Scrollbars.
* Can we alert users to a DefaultScrollAction conflict?
	+ Switching the defaultScrollAction should be allowed, but we might want to consider how to let users know that only one Scrollable can be registered as the DefaultScrollAction at a time. This could be covered by extensive documentation, but the ability to assert this kind of case would be nice.
* What’s in a name?
	+ DefaultScrollAction and Scrollable.isDefaultScrollAction are two of several name iterations. The intention is for the API to be easily understood by users, and hopefully easily discovered when looking to solve this use case.