# KeyCue™ ASI

# Application Shortcut Interface

**Version 1.1, April 2012**



## Contents

# Background

When we released the first version of KeyCue in 2004, KeyCue collected only keyboard shortcuts that were associated with menu commands. In 2008, we included shortcuts for Keyboard Maestro macros, and in 2009, we added support for QuicKeys. In these cases, we cooperated directly with the developers of these products and developed proprietary interfaces that allowed KeyCue to ask Keyboard Maestro and QuicKeys about currently available shortcuts.

Since the first versions of KeyCue, users asked us if it were possible to include application-specific shortcuts. Many applications use hard-wired keyboard shortcuts or user-definable hotkeys that are not accessible to KeyCue because there is no standard interface that would allow KeyCue to ask applications about such shortcuts … until now.

Starting with version 5.0, KeyCue can display arbitrary shortcuts found in other applications, but it needs some support by these applications. We have defined a simple interface that allows application developers to tell KeyCue about available keyboard shortcuts. To be more precise, applications just announce their ability to deliver information about shortcuts, and KeyCue subsequently asks these applications when it needs this information.

We hope that this interface will be adopted by many application developers. It is easy to implement and leads to an immediate benefit for both application developers and KeyCue users, as arbitrary applications can use KeyCue to display a "cheat sheet", and KeyCue can display all shortcuts found in an application, not only those that are tied to menu commands.

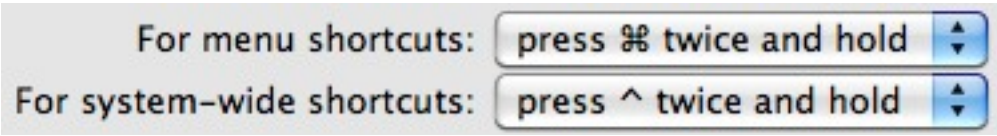
The following sections describe how the interface works and what application developers must do to support it.

# Overview

There are two types of shortcuts that applications can announce to KeyCue:

- *Application-specific shortcuts* are active only when an application is in the foreground. Typically, these shortcuts are handled during processing of keyboard events, and they are often active only in certain situations. For example, an application may use shift-F5 as a shortcut for opening a floating panel, but only when a certain type of document window is the key window. If such a keyboard combination is not associated with a menu command, KeyCue will not be able to "see" it when it scans the application's menus, but the application can tell KeyCue if and when the keyboard shortcut is available.
- *System-wide shortcuts* are active in all applications, as long as the supporting application is running. This is also the case for "daemon" applications that run silently in the background and provide keyboard shortcuts for utility functions. For example, a notepad application might provide a system-wide shortcut to quickly create a new note. Often, such shortcuts are implemented a "hotkeys" using the RegisterEventHotKey function found in CarbonEvents.h.

KeyCue has separate settings for menu shortcuts and system-wide shortcuts that can be invoked with different key combinations, as shown in this snapshot of KeyCue's settings:




KeyCue adds application-specific shortcuts to the list of menu shortcuts, and an application's system-wide shortcuts appear under the application's name in the system-wide shortcut table. The following snapshot shows how Typinator's shortcuts appear in the list of the system-wide shortcuts:


Since keyboard shortcuts are context-sensitive, KeyCue does not use static tables to learn about shortcuts of an application. Instead, KeyCue asks applications about shortcuts that are available right now. For example, Typinator's "Pause Expansions" shortcut changes to "Resume Expansions" when expansions are currently suspended.

KeyCue uses AppleEvents to ask applications about available keyboard shortcuts. Applications that support the KeyCue interface return a list of shortcuts descriptions.

There are two separate AppleEvents, one for application-specific and one for system-wide keyboard shortcuts. To save time, KeyCue sends these events only to applications that announce this feature in their Info.plist file.

The following sections explain in detail which steps you need to take to tell KeyCue about keyboard shortcuts in your application.

# Info.plist entries

To announce that your application supports the AppleEvents for querying keyboard shortcuts, you need to include one or both of the following entries in your Info.plist file:

```
<key>KeyCueApplicationSpecificKeys</key>
<string>1</string>
<key>KeyCueSystemWideKeys</key>
<string>1</string>
```

A string value of 1 tells KeyCue that your application supports application-specific or system-wide shortcuts. If your application supports only one kind of shortcut, omit the entry for the unsupported kind of shortcut or set the value to <string>0</string>.

The Info.plist entries must be present. When KeyCue does not find such an entry, it does not even try to send the corresponding AppleEvent.

# AppleEvents

KeyCue sends up to three types of AppleEvents to your application. Two of them ask your application to return a list of shortcut descriptions, and one tells your application to perform the action associated with a shortcut (see the section *Shortcut activation*).

The event class for all AppleEvents is '**KeyC**'. The event IDs for querying the available application-specific or system-wide shortcuts are '**AKey**' and '**SKey**', respectively. The event ID for shortcut activation is '**TKey**' (with "T" for "trigger").

The following method shows how you would install event handlers for all three events in a Cocoa application (typically in the applicationDidFinishLaunching: method of your application controller).

```
- (void) installAppleEventsForKeyCue
{
    NSAppleEventManager *manager = [NSAppleEventManager sharedAppleEventManager];
    [manager setEventHandler:self
        andSelector:@selector(getApplicationSpecificHotkeys:withReplyEvent:)
        forEventClass:'KeyC'
        andEventID:'AKey'];
    [manager setEventHandler:self
        andSelector:@selector(getSystemWideHotkeys:withReplyEvent:)
        forEventClass:'KeyC'
        andEventID:'SKey'];
    [manager setEventHandler:self
        andSelector:@selector(triggerHotkey:withReplyEvent:)
        forEventClass:'KeyC'
        andEventID:'TKey'];
}
```

The 'AKey' or 'SKey' events do not have any parameters. When your application receives one of these events, it must return a shortcut list, as described in the following section, *Shortcut list*.

The 'TKey' event has a string parameter that describes the shortcut to execute. See the section *Shortcut activation* for details.

# Shortcut list

The list of shortcuts must be a string representation of an array of dictionary entries in plist format. In a Cocoa application, create an NSArray of NSDictionary elements, where each dictionary describes a shortcut. To convert the array into a string representation, simply use the *description* method.

The following dictionary elements describe a shortcut:

| key | type | meaning |
|---|---|---|
| keycode | NSNumber | virtual key code |
| keystring | NSString | string representation of the key |
| modifiers | NSNumber | modifier mask |
| title | NSString | text to display for the shortcut |
| id | NSString | unique ID (used for activation of the shortcut) |

*modifiers*, *title* and *id* must be present. You can specify the key by means of its virtual key code or by a string. For example, you could specify the *esc* key by the key code 53 or by the string "esc". In general, we recommend using the virtual key code, as it often allows for a more compact display in the KeyCue table.

The modifiers are a bitwise combination of the modifier flag constants, as defined in NSEvent.h: NSCommandKeyMask, NSShiftKeyMask, NSAlternateKeyMask, NSControlKeyMask. A value of 0 means that no modifier is required.

The title of a shortcut should be a concise description of the action, in the same style as you would use for menu commands. KeyCue does not translate or otherwise manipulate this string, so you should return a localized form of the title, if possible.

If you wish, you can group keyboard shortcuts by inserting heading entries. A heading entry contains only a title and causes KeyCue to display all subsequent shortcuts as part of a group. For example, you could use the heading "Navigation" for shortcuts for moving around, and another heading "Sound" for keyboards associated with sound effects. To terminate a group without introducing a new group, insert a heading item with an empty title.

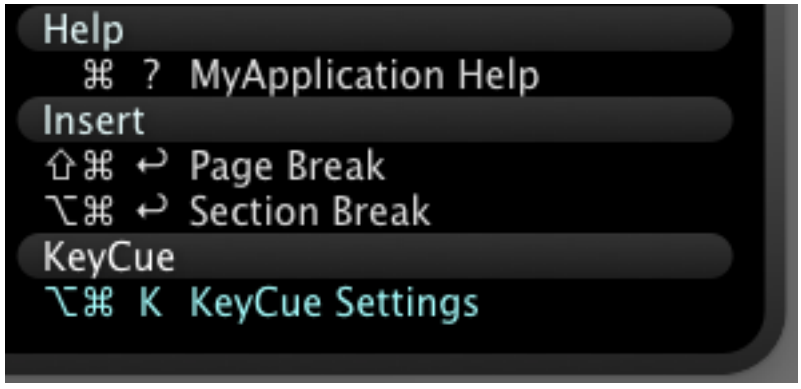
Note that KeyCue does not rearrange the items. You are responsible for arranging them in an appropriate order.

The following code fragment shows how an application would announce two application-specific shortcuts “Page Break” and “Section Break” in a group named “Insert”. In the example, the shortcuts are hard-wired as command-shift-return and command-option-return (36 is the virtual key code for the return key):

```
- (void) getApplicationSpecificHotkeys:(NSAppleEventDescriptor*)event
            withReplyEvent:(NSAppleEventDescriptor*)reply
{
    NSMutableArray *keyList = [NSMutableArray array];
    if ([self documentIsOpen]) {
        [keyList addObject:[NSDictionary dictionaryWithObjectsAndKeys:
            @"Insert", @"title",
            nil]];
        [keyList addObject:[NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:NSCommandKeyMask+NSShiftKeyMask], @"modifiers",
            [NSNumber numberWithInt:36], @"keycode",
            @"Page Break", @"title",
            @"newPage", @"id",
            nil]];
        [keyList addObject:[NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:NSCommandKeyMask+NSAlternateKeyMask], @"modifiers",
            [NSNumber numberWithInt:36], @"keycode",
            @"Section Break", @"title",
            @"newSection", @"id",
            nil]];
    }
    NSString *replyString = [keyList description];
    NSAppleEventDescriptor *replyDesc = [NSAppleEventDescriptor descriptorWithString:replyString];
    [reply setDescriptor:replyDesc forKeyword:keyAEResult];
}
```

This method assumes that the shortcuts are available only when a document is open. If this is not the case, it returns an empty list of shortcuts. The IDs "newPage" and "newSection" uniquely identify these shortcuts when the user clicks them in the KeyCue window. See the section on *Shortcut activation* for an example.

The following snapshot shows how the result of this method appears in the shortcut table of KeyCue:




Shortcuts that appear before the first group heading (or when you do not insert group headings at all) are listed under a heading “Other XY Shortcuts” (where XY is the name of your application) to separate the shortcuts from the menu shortcuts.

System-wide shortcuts are added at the end of the shortcut list under the name of the application (as shown on page 3 for Typinator). Since multiple applications can provide system-wide shortcuts, this table can contain multiple such sections. If you use group headings for system-wide shortcuts, KeyCue displays them in the same style as submenus (with a separate heading and a frame around the items).

# Shortcut activation

When the user clicks an item in the shortcut table, KeyCue sends a 'TKey' AppleEvent to tell your application to execute the corresponding action. You can find the ID of the shortcut in the direct object of the event descriptor. The following method shows how your method would handle clicks on the “Page Break” and “Section Break” items in the KeyCue window:

```
- (void) triggerHotkey:(NSAppleEventDescriptor*)event
             withReplyEvent:(NSAppleEventDescriptor*)reply
{
    NSAppleEventDescriptor *parameter = [event descriptorForKeyword:keyDirectObject];
    NSString *hotKeyID = [parameter stringValue];
    if ([hotKeyID isEqual:@"newPage"]) {
        [self insertPageBreak];
    } else if ([hotKeyID isEqual:@"newSection"]) {
        [self insertSectionBreak];
    }
}
```

KeyCue does not expect any reply from the AppleEvent. Your method should therefore just perform the action and return.

The 'TKey' AppleEvent is used for both application-specific and system-wide shortcuts. If your application has both types of shortcuts, you can handle all of them in the same method.

# Hotkey feedback

Common techniques for providing system-wide shortcuts include the “hotkey” mechanism in CarbonEvents.h (RegisterEventHotKey, etc.) and event taps, as defined in Quartz Event Services. In both cases, your application will receive and handle the keyboard events, and then the events are removed from the event pipeline. When the user presses the key combination for your hotkey while KeyCue’s shortcut table is visible, KeyCue will not receive the event. However, KeyCue has an option “Typing a shortcut closes the KeyCue window”. When this option is turned on, the users will expect the shortcut window to disappear. To facilitate that, KeyCue provides an AppleEvent handler that lets your application tell KeyCue that a keyboard shortcut was pressed, so KeyCue can close the shortcut window.

The event class for this AppleEvent is '**KeyC**', and the event ID is '**KeyB**'. You can use the following function to send this event to KeyCue:

```
void SendAppleEventToKeyCue (OSType eventID)
{
    AEAddressDesc target;
    AppleEvent theAppleEvent;
    AppleEvent replyEvent;
    OSType KeyCue = 'KeyC';
    if (AECreateDesc(typeApplSignature, &KeyCue, sizeof(OSType), &target) == noErr) {
        if (AECreateAppleEvent(KeyCue, eventID, &target,
            kAutoGenerateReturnID, kAnyTransactionID, &theAppleEvent) == noErr) {
                AESendMessage(&theAppleEvent, &replyEvent, kAENoReply, 20); // ignore result
                AEDisposeDesc(&theAppleEvent);
        }
        AEDisposeDesc(&target);
    }
}
```

Whenever your application handles a hotkey, just call

```
SendAppleEventToKeyCue('KeyB');
```

Depending on the current state of KeyCue, this call has one of the following effects:

- When KeyCue is not installed or not running, nothing happens.
- When KeyCue is running and idle, nothing happens. When a user types the keyboard combination without the help of KeyCue, KeyCue simply ignores the event.
- When the user presses the keys for KeyCue's shortcuts, and the shortcut table has not appeared yet, KeyCue will cancel the request. Your application will handle the hotkey, and KeyCue will not display the shortcut table.
- When the shortcut table is visible and the option "Typing a shortcut closes the KeyCue window" is turned on, KeyCue closes the shortcut table.
- When the shortcut table is visible and the option "Typing a shortcut closes the KeyCue window" is turned off, KeyCue ignores the event, so the shortcut table remains visible.

Note that you can call SendAppleEventToKeyCue('KeyB'); unconditionally without checking for the presence of KeyCue or any other of the above states.

# Apple Events from the sandbox

According to Apple's rules for sandboxed applications, you cannot send AppleEvents to other applications unless you add an apple-events temporary exception to the entitlements of your application, as described in Apple's "Entitlement Key Reference" documentation:

http://tinyurl.com/apple-events-in-sandbox

To send Apple Events to KeyCue, specify the bundle identifier

com.macility.keycue

# Tell us

Please let us know when you are using the KeyCue interface in one of your applications. Just send a short note to keycue-support@ergonis.com and tell us which application and which version of that application supports the KeyCue interface. We will be happy to announce this on our web site, and we will keep you informed when we enhance or otherwise change the shortcut interface.

If you have any questions about KeyCue, need a license for testing the KeyCue interface with your application, or need support using the KeyCue interface, please contact us at keycue-support@ergonis.com. We will be glad to help.