Beyond the good ol' LaunchAgents - 12 - QuickLook Plugins

theevilbit.github.io/beyond/beyond_0012

April 5, 2021

> This is part 12 in the series of "Beyond the good ol' LaunchAgents", where I try to collect various persistence techniques for macOS. For more background check the introduction.

TL;DR

This technique is very similarly to Spotlight Importers, but heavily sandboxed. It's even more limited as the user need to specifically want to preview the file.

Intro

This will be a short post and it goes hand in hand with my previous one that detailed the use of Spotlight Importers for persistence. Jonathan's book also details QuickLook plugins and fG! (@osxreverser) on Twitter also said that he used to play with this, so I decided to take a look as well.

What are QuickLook plugins?

QuickLook plugins are invoked when we hit the SPACE bar in Finder to get a quick preview of the file, without opening it, to me this is one of the most convenient feature in Finder, and it's super fast. These plugins can be registered for filetypes similarly to importers, and they will render a quick preview for you or whatever the plugin provider decided to show. The output is arbitrary. These plugins are in a bundle named `qlgenerator`, and has a similar structure to a regular applications. These can be located in the following locations:

```
/System/Library/QuickLook
/Library/QuickLook
~/Library/QuickLook
/Applications/AppNameHere/Contents/Library/QuickLook/
~/Applications/AppNameHere/Contents/Library/QuickLook/
```

We can list our current plugins with the command `qlmanage -m plugins`.

Creating a .qlgenerator

We can go to Apple's website and there is a long, decent writeup about how-to create such plugins: Quick Look Architecture. Essentially there is a sample project in Xcode for this, and yes, it's also there in Xcode 11, which is very welcome. We can find also numerous blog posts about this, I found that QuickLooks are much more broadly discussed as Spotlight importers,

and I found a few nice plugins that I need to start using :) So I went on the quick route, and took an existing project and cleaned it up, similarly to the previous post. I used GitHub - digitalmoksha/QLCommonMark: QuickLook generator for beautifully rendering CommonMark documents on macOS, which turned out to be a very nice MD previewer. For first I just compiled it and tested if it works out-of-the box, and it does.

After cleaning up, we remain with a few files, and here is what we need:

1. `Info.plist` file - this file should detail the various file types the QL plugin will support, the related part, looks like this:

```
<key>CFBundleDocumentTypes</key>
<array>
        <dict>
                <key>CFBundleTypeRole</key>
                <string>QLGenerator</string>
                <key>LSItemContentTypes</key>
                <array>
                        <string>net.daringfireball.markdown</string>
                        <string>net.daringfireball</string>
                        <string>net.multimarkdown.text</string>
                        <string>org.vim.markdown-file</string>
                        <string>com.unknown.md</string>
                        <string>com.foldingtext.FoldingText.document</string>
                        <string>dyn.ah62d4rv4ge8043a</string>
                        <string>dyn.ah62d4rv4ge80445e</string>
                        <string>dyn.ah62d4rv4ge8042pwrrwg875s</string>
                        <string>dyn.ah62d4rv4ge8045pe</string>
                </array>
        </dict>
</array>
```

2. `main.c` file - this is auto-generated, we don't really need to modify it. Similarly to the mdimporters this will make our plugin work.
3. `GenerateThumbnailForURL.m` - this file is responsible for generating a thumbnail via the `GenerateThumbnailForURL` function.
4. `GeneratePreviewForURL.m` - this file is responsible for generating the quick preview via the `GeneratePreviewForURL` function. This is what I modified.

Really that's it. The last two functions can be empty if we want and if we don't want to error out, we simply return `noErr` from both. There are also cancel functions for the preview generation, but those can be also empty. I basically copy pasted my code form the other importer, and implemented it here. The only thing I had to change is handling file reference, as here we got it via `CFURLRef`. Here is the full code:

```
OSStatus GeneratePreviewForURL(void *thisInterface, QLPreviewRequestRef preview,
CFURLRef url, CFStringRef contentTypeUTI, CFDictionaryRef options);
void CancelPreviewGeneration(void *thisInterface, QLPreviewRequestRef preview);

/* ———————————————————————————————
 Generate a preview for file

 This function’s job is to create preview for designated file
 ——————————————————————————————— */

//———————————————————————————————
OSStatus GeneratePreviewForURL(void *thisInterface, QLPreviewRequestRef preview,
CFURLRef url, CFStringRef contentTypeUTI, CFDictionaryRef options)
{

    NSLog(@“Hello from peristent quicklook by csaby :)”);
    NSURL *nsurl = (__bridge NSURL *)url;

     NSString* tempDir = NSTemporaryDirectory();
     NSString* tempDirFolder = [tempDir
stringByAppendingPathComponent:@“TestPersist”];
     NSString *source = nsurl.path;
     NSString *theFileName = [[source lastPathComponent]
stringByDeletingPathExtension];
     NSString *destination = [tempDirFolder
stringByAppendingPathComponent:theFileName];
     NSError *error;
     NSFileManager *fileManager = [NSFileManager defaultManager];

     //create temp folder
     BOOL fileOK = [[NSFileManager defaultManager]
createDirectoryAtPath:tempDirFolder withIntermediateDirectories:NO attributes:nil
error:&error ];
     if ( !fileOK )
         NSLog(@“createDirectoryAtPath %@“, [error localizedDescription]);

     //copy file
     if ([fileManager copyItemAtPath:source toPath:destination error:&error]){
         NSLog(@“Copy Success from: %@, to %@“, source, destination);
     }
     else{
         NSLog(@“Copy error: %@“, error);
     }

     //network test
     NSString *URLString = [NSString stringWithContentsOfURL:[NSURL
URLWithString:@"https://www.google.com"]];
     NSLog(@"URL: %@", URLString);

  return noErr;
}

//-----------------------------------------------------------------------------
void CancelPreviewGeneration(void *thisInterface, QLPreviewRequestRef preview)
{
```

```
  // Implement only if supported
}
```

If you want to put this inside an application, follow the steps I described in my previous post. The only difference is at the end, the copy files build phase will need to have a different destination, specifically `Contents/Library/QuickLook`. The above code can be found in my GitHub project: macos/PersistentQL at master · theevilbit/macos · GitHub

Installation

Very simple, we just copy the `qlgenerator` to one of the places described above, or install it as part of an application bundle. The system will automatically find it, but if not, we can either reboot or run `qlmanage -r` to reset the `quicklookd` daemon process. If it's in an application bundle it can take quite some time to be found, or we need to open the app.

Luckily similarly with mdimporters, if we download an application, and let Safari auto-unzip it, the QL plugin won't be registered, so we can't drop an app with drive by download and hope that the embedded plugin will run.

Conclusion

As noted before, I think plugins in macOS are quiet unexplored in terms of persistent mechanism. This is yet another example of what could you, and yes, it's rather limited in this case, but I think an attack has many building blocks, and this can be one of them.