
pod **v1.11.2** Carthage compatible SwiftPM compatible

PLCrashReporter

PLCrashReporter is a reliable open source library that provides an in-process live crash reporting framework for use on iOS, macOS and tvOS. The library detects crashes and generates reports to help your investigation and troubleshooting with the information of application, system, process, thread, etc. as well as stack traces.

The easiest way to use PLCrashReporter is by using AppCenter. However, if you want to use PLCrashReporter directly, grab the latest release at [releases page](#).

Features

- Uses only supported and public APIs/ABIs for crash reporting.
- The most accurate stack unwinding available, using DWARF and Apple Compact Unwind frame data.
- First released in 2008, and used in hundreds of thousands of apps. PLCrashReporter has seen a tremendous amount of user testing.
- Does not interfere with debugging in lldb/gdb
- Backtraces for all active threads are provided.
- Provides full register state for the crashed thread.

Prerequisites

- Xcode 11 or above.
- Minimum supported platforms: iOS 11, macOS 10.9, tvOS 11, Mac Catalyst 13.0.

Decoding Crash Reports

Crash reports are output as protobuf-encoded messages, and may be decoded using the CrashReporter library or any Google Protocol Buffers decoder.

In addition to the in-library decoding support, you may use the included `plcrashutil` binary to convert crash reports to apple's standard iPhone text format:

```
1 plcrashutil convert --format=iphone example_report.plcrash
```

You can use `atos` command-line tool to symbolicate the output. For more information about this tool, see [Adding Identifiable Symbol Names to a Crash Report](#). Future library releases may include built-in re-usable formatters, for outputting alternative formats directly from the phone.

Adding PLCrashReporter to your project

PLCrashReporter can be added to your app via CocoaPods, Carthage, Swift Package Manager, or by manually adding the binaries to your project.

Integration via Cocoapods

1. Add the following line to your `Podfile`: `ruby pod 'PLCrashReporter'`
2. Run `pod install` to install your newly defined pod and open the project's `.xcworkspace`.

Integration via Swift Package Manager

1. From the Xcode menu, click **File > Swift Packages > Add Package Dependency**.
2. In the dialog that appears, enter the repository URL: `https://github.com/microsoft/plcrashreporter.git`.
3. In Version, select **Up to Next Major** and take the default option.

Integration via Carthage

1. Add the following line to your `Cartfile`: `ruby github "microsoft/plcrashreporter"`
2. Run `carthage update --use-xcframeworks` to fetch dependencies.
3. In Xcode, open your application target's **General** settings tab. Drag and drop **CrashReporter.xcframework** from the **Carthage/Build** folder into the **Frameworks, Libraries and Embedded Content** section. For iOS and tvOS, set **Embed** to **Do not embed**. For macOS, set **Embed** to **Embed and Sign**.

NOTE: Carthage integration doesn't build the dependency correctly in Xcode 12 with flag `"-no-use-binaries"` or from a specific branch. To make it work, refer to this [instruction](#).

Integration by copying the binaries into your project

1. Download the PLCrashReporter frameworks provided as a zip file.

-
2. Unzip the file and you'll see a folder called **PLCrashReporter** that contains subfolders for all supported platforms.
 3. Add PLCrashReporter to the project in Xcode:
 - Make sure the Project Navigator is visible (⌘+1).
 - Now drag & drop **PLCrashReporter.framework** (or **PLCrashReporter.xcframework**) from the Finder into Xcode's Project Navigator.
 - A dialog will appear, make sure your app target is checked and click **Finish**. > **NOTE:** > PLCrashReporter xcframework contains static binaries for iOS and tvOS, and dynamic binaries for macOS. When adding the framework to your project make sure that in **Frameworks, Libraries and Embedded Content** section **Embed** is selected to **Do not embed** for iOS and tvOS and **Embed and Sign** for macOS. **PLCrashReporter-Static-{version}.zip** is an exception - it contains static frameworks for all platforms.

Example

The following example shows a way how to initialize crash reporter. Please note that enabling in-process crash reporting will conflict with any attached debuggers so make sure the **debugger isn't attached** when you crash the app.

Objective-c

```
1 @import CrashReporter;
2
3 ...
4
5 // Uncomment and implement isDebuggerAttached to safely run this code
   with a debugger.
6 // See: https://github.com/microsoft/plcrashreporter/blob/2
   dd862ce049e6f43feb355308dfc710f3af54c4d/Source/Crash%20Demo/main.m#
   L96
7 // if (![self isDebuggerAttached]) {
8
9 // It is strongly recommended that local symbolication only be enabled
   for non-release builds.
10 // Use PLCrashReporterSymbolicationStrategyNone for release versions.
11 PLCrashReporterConfig *config = [[PLCrashReporterConfig alloc]
   initWithSignalHandlerType: PLCrashReporterSignalHandlerTypeMach
12                                     symbolicationStrategy:
   PLCrashReporterSymbolicationStrategyNone]
```

```

13 PLCrashReporter *crashReporter = [[PLCrashReporter alloc]
    initWithConfiguration: config];
14
15 // Enable the Crash Reporter.
16 NSError *error;
17 if (![crashReporter enableCrashReporterAndReturnError: &error]) {
18     NSLog(@"Warning: Could not enable crash reporter: %@", error);
19 }
20 // }
];

```

Checking collected crash report can be done in the following way:

```

1 if ([crashReporter hasPendingCrashReport]) {
2     NSError *error;
3
4     // Try loading the crash report.
5     NSData *data = [crashReporter
        loadPendingCrashReportDataAndReturnError: &error];
6     if (data == nil) {
7         NSLog(@"Failed to load crash report data: %@", error);
8         return;
9     }
10
11     // Retrieving crash reporter data.
12     PLCrashReport *report = [[PLCrashReport alloc] initWithData: data
        error: &error];
13     if (report == nil) {
14         NSLog(@"Failed to parse crash report: %@", error);
15         return;
16     }
17
18     // We could send the report from here, but we'll just print out
        some debugging info instead.
19     NSString *text = [PLCrashReportTextFormatter
        stringValueForCrashReport: report withTextFormat:
        PLCrashReportTextFormatiOS];
20     NSLog(@"%@", text);
21
22     // Purge the report.
23     [crashReporter purgePendingCrashReport];
24 }

```

Swift

```

1 import CrashReporter
2
3 ...

```

```

4 // Uncomment and implement isDebuggerAttached to safely run this code
  with a debugger.
5 // See: https://github.com/microsoft/plcrashreporter/blob/2dd862ce049e6f43feb355308dfc710f3af54c4d/Source/Crash%20Demo/main.m#L96
6 // if (!isDebuggerAttached()) {
7
8     // It is strongly recommended that local symbolication only be
      enabled for non-release builds.
9     // Use [] for release versions.
10    let config = PLCrashReporterConfig(signalHandlerType: .mach,
      symbolicationStrategy: .all)
11    guard let crashReporter = PLCrashReporter(configuration: config) else
      {
12        print("Could not create an instance of PLCrashReporter")
13        return
14    }
15
16    // Enable the Crash Reporter.
17    do {
18        try crashReporter.enableAndReturnError()
19    } catch let error {
20        print("Warning: Could not enable crash reporter: \(error)")
21    }
22 // }

```

Checking collected crash report can be done in the following way:

```

1 // Try loading the crash report.
2 if crashReporter.hasPendingCrashReport() {
3     do {
4         let data = try crashReporter.
          loadPendingCrashReportDataAndReturnError()
5
6         // Retrieving crash reporter data.
7         let report = try PLCrashReport(data: data)
8
9         // We could send the report from here, but we'll just print out
          some debugging info instead.
10        if let text = PLCrashReportTextFormatter.stringValue(for: report,
          with: PLCrashReportTextFormatiOS) {
11            print(text)
12        } else {
13            print("CrashReporter: can't convert report to text")
14        }
15    } catch let error {
16        print("CrashReporter failed to load and parse with error: \(error
          )")
17    }
18 }
19

```

```
20 // Purge the report.  
21 crashReporter.purgePendingCrashReport()
```

Building

Prerequisites

- A Mac running macOS compliant with Xcode requirements.
- Xcode 11 or above.

Also, next optional tools are used to build additional resources:

- Doxygen to generate the documentation. See the official Doxygen repository for more information or use Homebrew to install it.
- GraphViz to generate the documentation. See the official GraphViz website for more information or use Homebrew to install it.
- [protobuf-c](#) to convert Protocol Buffer [.proto](#) files to C descriptor code. See the official [protobuf-c](#) repository for more information or use Homebrew to install it.

Building

- Open a new window for your Terminal.
- Go to PLCrashReporter's root folder and run

```
1 xcodebuild -configuration Release -target 'CrashReporter'
```

to create binaries for all platforms.

Contributing

We are looking forward to your contributions via pull requests.

To contribute to PLCrashReporter, you need the tools mentioned above to build PLCrashReporter for all architectures and [protobuf-c](#) to convert Protocol Buffer [.proto](#) files to C descriptor code.

Code of Conduct

This project has adopted the Microsoft Open Source Code of Conduct. For more information see the Code of Conduct FAQ or contact opencode@microsoft.com with any additional questions or comments.