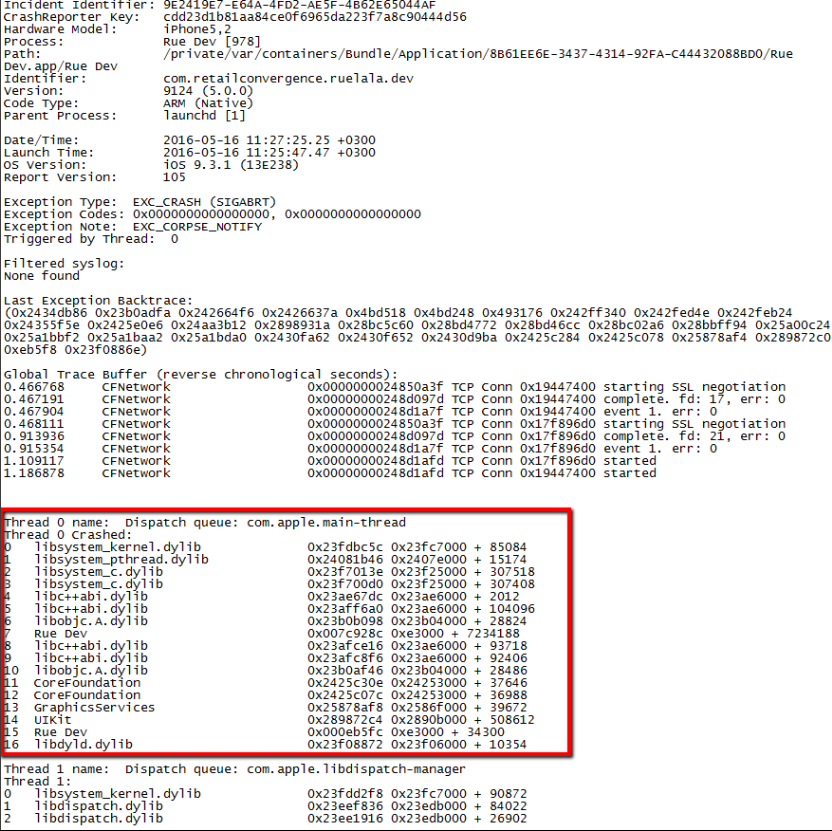
<https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/AnalyzingCrashReports/AnalyzingCrashReports.html>

<https://developer.apple.com/library/ios/technotes/tn2151/_index.html>

<https://habrahabr.ru/post/167805/>

В отличие от ОС Android, crash reports в iOS создаются автоматически. Но при этом в целях безопасности, чтобы злоумышленник не смог прочитать имена процедур, все их имена закодированы.

Вот как выглядит фрагмент обычного crash-лога:



В самом начале лога идёт общая информация: информация об устройстве, приложении, ОС, дата и время и т.п.

Первый раздел дает нам информацию о том процессе, который был аварийно остановлен.

* *Incident Identifier* - уникальный идентификатор журнала сбоя.
* *CrashReporter Key* - тоже уникальный ключ, который связан с идентификатором устройства. Хотя он анонимизирован, он дает вам очень полезную информацию: если вы видите, что все ваши 100 аварийных журналов имеют один и тот же параметр **CrashReporter Key** (или только два разных), это означает, что эта проблема - не широко распространенная проблема, а ограничена только одним или несколькими устройствами.
* *Hardware Model* - тип устройства. Если вы получаете много аварийных журналов от одной и той же модели устройства, это может означать, что ваше приложение не работает должным образом с конкретной моделью. В нашем примере это iPhone 4S.
* *Process* - имя приложения. Число в квадратных скобках – это идентификатор процесса приложения (PID) в момент сбоя.
* Следующие несколько строк должны быть понятны и так.

Далее идёт основная информация. Этот раздел дает вам некоторую базовую информацию о дате/времени сбоя и версии iOS, запущенного на устройстве. Если у вас много журналов сбоев от iOS 8.0, к примеру, это может означать, что эта проблема специфична для iOS 8.

Информация об исключении (exception): в этом разделе вы видите тип исключения, который был получен в момент сбоя. Вы также видите код исключения и поток, который допустил исключение. В зависимости от типа аварийного отчета, в этом разделе может быть некая дополнительная информация. В строке "**Exception type:**" видим значение **EXC\_CRASH**. Это означает, что это лог именно крэша, некорректного завершения приложения вследствие какой-то ошибки в коде или ресурсе ([SIGABRT](https://ru.wikipedia.org/wiki/SIGABRT)— [сигнал](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB%D1%8B_(UNIX)), посылаемый процессом самому себе при выполнении функции abort(), для аварийного останова, в случае невозможности дальнейшего продолжения программы). Проще говоря, приложение получает сигнал SIGABRT, когда есть вызов несуществующего метода. iOS генерирует crash reports и для завершения приложений системой, когда нужно освободить ресурсы аппарата для более приоритетного приложения. Такие логи не являются для нас интересными.

Далее идёт трассировка потоков (stack trace). Этот раздел содержит трассировку для всех потоков приложения. Трассировка представляет собой список всех активных фреймов в момент сбоя. Мы видим, какие функции вызывались, когда произошёл сбой. Красным прямоугольником выделен наиболее интересующий нас фрагмент лога. Именно здесь пошагово описываются процессы и процедуры, вызвавшие отказ приложения. Но как видим, они закодированы с использованием 16-ричной системы.

**Что это вообще за колонки в стеке:**

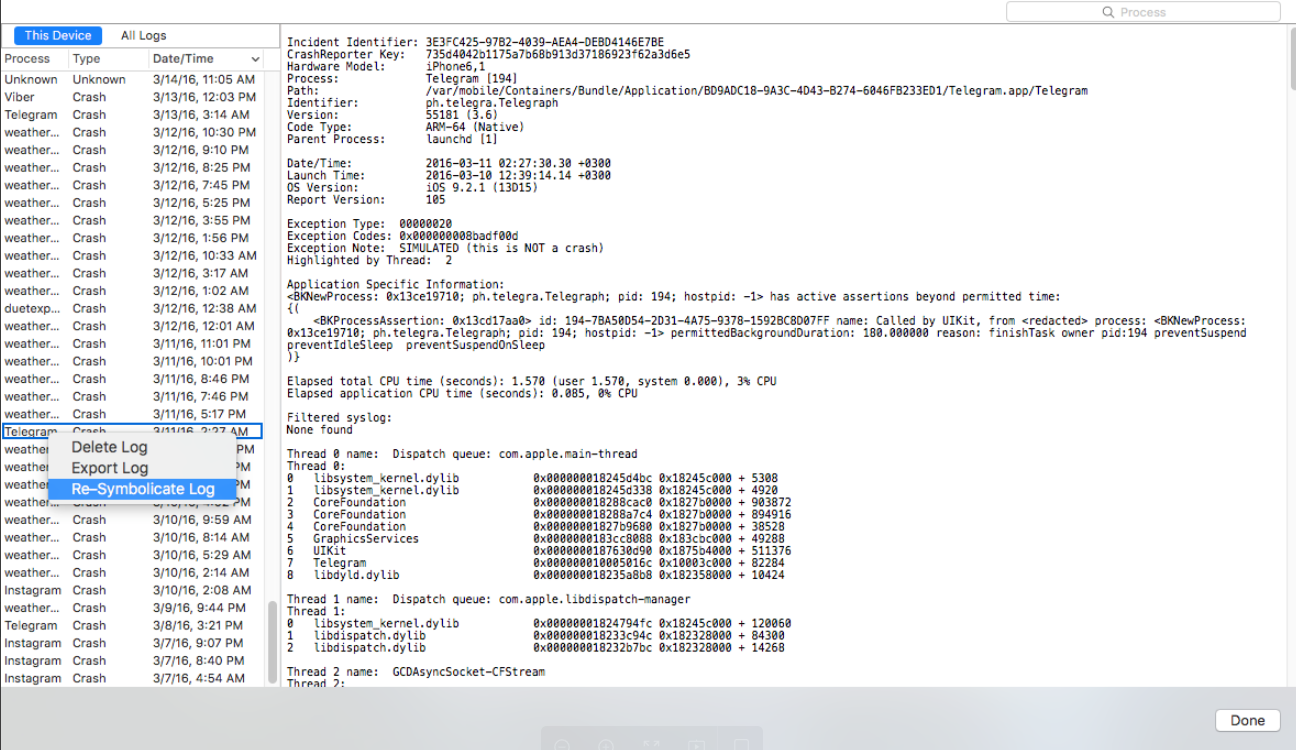
* В первой колонке содержится порядковый номер записи (номер фрейма). Важно знать, что нумерование в стеке идет в обратном порядке, т.е. от большего к меньшему. Таким образом, вызов с порядковым номером 0 и есть первопричина падения приложения.
* Во второй колонке содержится название фреймворка или системной библиотеки.
* Далее следует абсолютный адрес точки входа функции (функция = вызов).
* В четвертой колонке две части: базовый адрес и смещение. Тут это 0×23fc7000 + 85084, где первое число указывает на файл, а второе - на строку кода в файле.

Нам нужно понять, что же такое закодировано в строке с номером 0. Для этого нужно применить процесс так называемой символизации. Есть как минимум 2 способа (оба для MacOS):

Xcode

Все крэш-логи можно просмотреть при помощи утилит, идущих с Xcode.

Xcode -> Window -> Devices -> View Device Logs

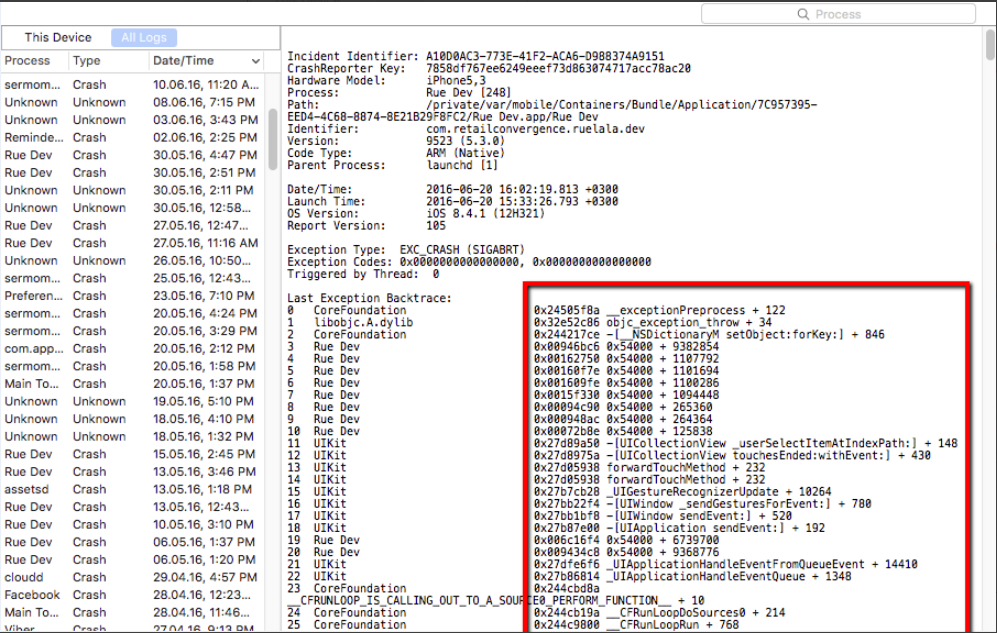


Слева в списке логов устройства находим интересующий нас, кликаем правой кнопкой мыши и выбираем "Re-Symbolicate Log".

Для полной десимволизации необходим не только сам крэш-лог, но и так называемый .dsym файл приложения, а также бинарный файл приложения. Если у вас есть .ipa файл приложения, то распаковав обычным архиватором, вы сможете найти бинарник. Например, если ваше приложение называется "HeloWorld.ipa", то распаковав его, вы должны увидеть папку Payload, а внутри нее HelloWorld.app - это и есть необходимая Вам папка.

Что же касается .dsym файла, то он не всегда может быть в .ipa файле. Обычно билды приложений для iOS приходят на тесты без dev подписи, а как релизная или предрелизная версия. В таких сборках .dsym файла не будет, и вам понадобится спросить его у разработчика.

Ниже приведен фрагмент частично десимволизированного iOS crash-лога:

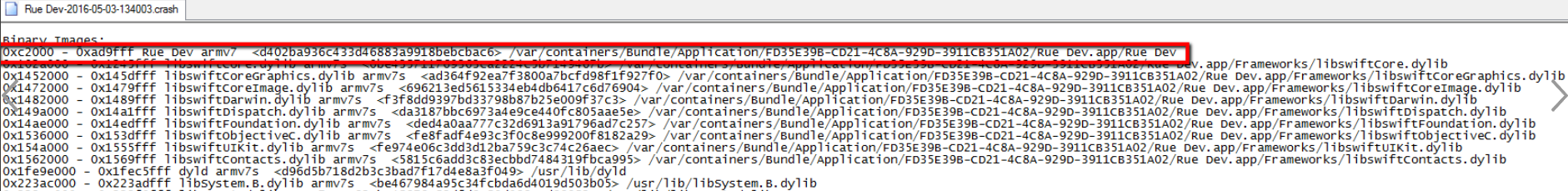


Итого, чтобы полностью символизировать crash-лог с использованием Xcode, вам необходимы

* Xcode
* Сам Crash-log
* Файл приложения, которое было загружено в App Store
* DSYM-файл

Тут должно быть точное соответствие версий, в противном случае аварийный журнал не может быть полностью символизирован.

То есть ваш crash report, iOS app binary и DSYM-файл должны иметь один и тот же UUID. Посмотреть какой именно UUID вам нужен, можно в самом crash-логе в секции Binary Images:



В примере на снимке экрана необходимый нам UUID - это d402ba936c433d46883a9918bebcbac6.

# Утилита atos

atos (Address to Symbol) - на вход ей дается имя бинарного файла и интересующий адрес, а на выходе получаем название класса, метода и номер строки в исходном коде.

Ситаксис использования прост:

# atos [-o <binary-image-file>] [-p <pid> | <partial-executable-name>] [-arch architecture] [-l <load-address>] [-s <slide>] [-printHeader] [-f <address-input-file>] [<address> ...]

# atos -arch armv7 -o APPNAME.app/APPNAME MEMORY\_LOCATION\_OF\_CRASH

Examples:

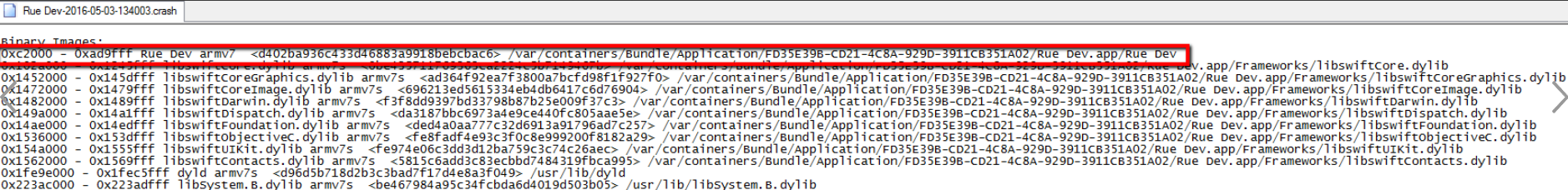
# atos -arch armv6 -o MyApp 0x00004ee2

# atos -arch armv7 -o 'Vimeo.app'/'Vimeo' 0x00004ee2

где

# — символ приглашения командной строки  
-arch armv6 — указание таргет-платформы. тут важно понимать, под какую платформу собирался проект — это также можно узнать в Build Settings. Если указать неправильную платформу, можно получить не корректный результат.  
-o MyApp — название исполняемого файла. Он находится там же, где и .dSYM файл, то есть в папке бандла приложения (MyApp.app)  
-0x00004ee2 — собственно, виновник торжества — адрес крэш инструкции.

На скриншоте ниже мы можем сразу найти и версию архитектуры (armv7), а также имя исполняемого файла ('Rue Dev.app'/'Rue Dev')



* Для символизации вам всё равно понадобится бинарный файл приложения и .dsym-файл.
* Верно указывайте версию архитектуры, т.к. даже с неверной версий atos может выдать вам символьные имена функций, которые будут абсолютно неверны для вашего приложения.