FridaLink User Manual

FridaLink - TCP/JSON based protocol and IDA plugin for establishing bridge between **Frida** client and **IDA Pro**. As a part of a **FRAPL** framework, it is designed to bring missing information to Frida from IDA disassembly and to monitor dynamic changes by controlling Frida directly from IDA.

Overview

FridaLink setup is **as simple as 1-2-3** and requires just three simple steps to actually start reverse engineering your target. Without a single line of code.

1. In IDA press **ALT+F7** and load **FridaLink.py**
2. In terminal run `$ ./create_project.sh -f ~/Projects/TargetApp ; cd ~/Projects/TargetApp` to create project
3. In terminal run `$ node ./client.js -l -n TargetApp server.js` to start FridaLink

As soon as FridaLink is established researcher is getting the best from the static and dynamic analisys. Full interactivity from IDA static analysis features plus the ability observe runtime state via CPU context, stack, backtraces and memory directly in IDA. It is possible to put custom code in hooks or replace functions entirely which is even easier with the FRAPL API. No maintenance code required, just the one for reverse engineering needs. Since FridaLink is highly integrated into IDA, all these features are available from popup menus and researcher can track as many things as they like as soon as there is a space on screen.

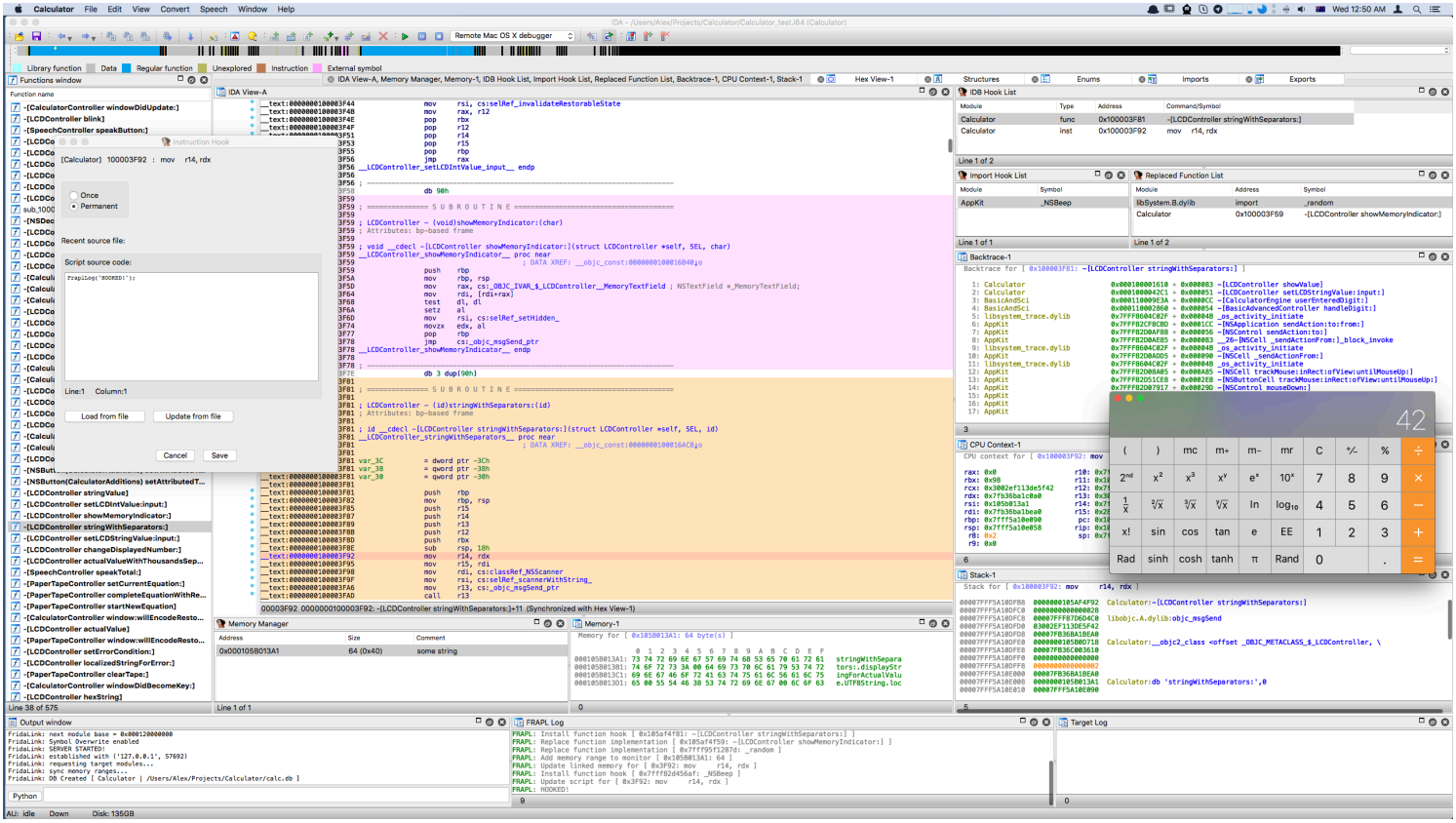


Table Of Contents

Loading FridaLink plugin into IDA

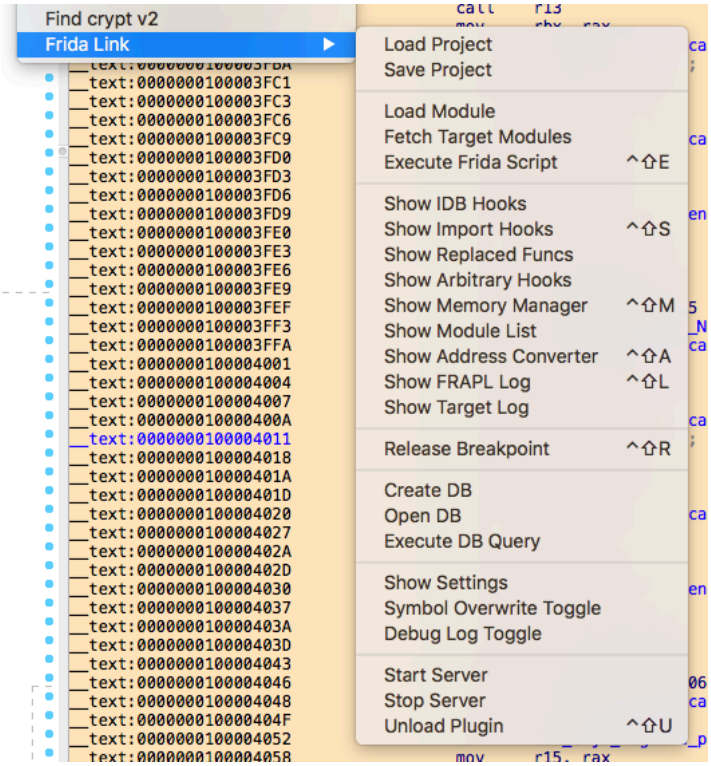
To load FridaLink plugin use **File/Script file...** menu or **ALT+F7** and choose **FridaLink.py** file.

Unloading FridaLink plugin

To unload FridaLink plugin use **Edit/Plugins/Frida Link/Unload Plugin** or **CTRL+SHIFT+U**.

FridaLink Main Menu

To assess general FridaLink functions and various FridaLink views, use **Edit/Plugins/Frida Link/** menu.



Server Contol

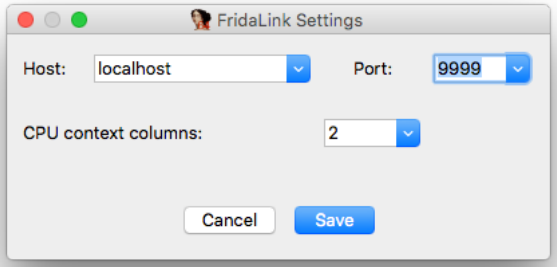
FridaLink server starts and stops automatically as soon as plugin is loaded/exited.
Additionally it can be started manually using **Edit/Plugins/Frida Link/Server Start** or stopped using **Edit/Plugins/Frida Link/Server Stop**.

FridaLink Project Settings

Settings window allows to set parameters like:
Host - FridaLink server host address (default: localhost)
Port - FridaLInk server port (default: 9999)
CPU context columns - number of columns in CPU context view

FridaLink project can be saved in any time while plugin is loaded. However, it can loaded ony when FridaLink connection is established which is required to setup all hooks into target. FridaLink project extension is "*.flp".

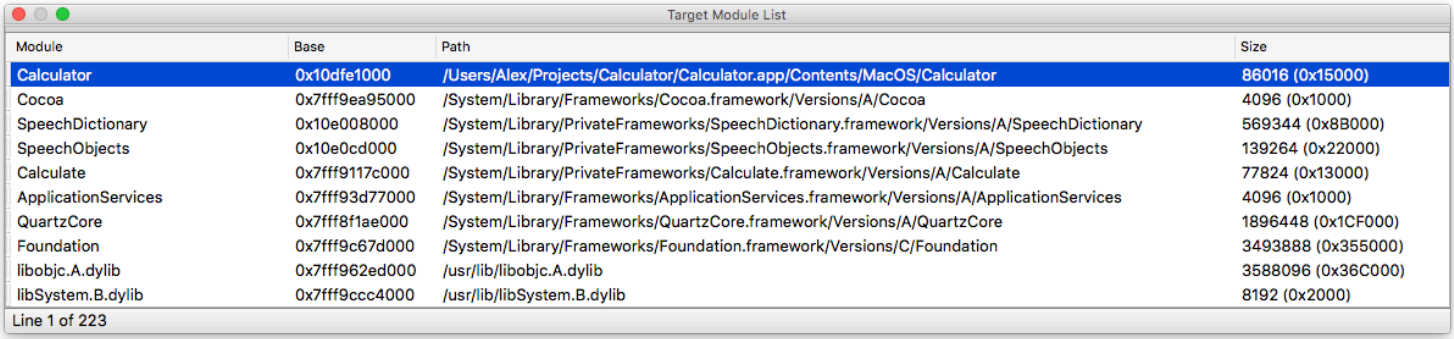
NOTE: FridaLink windows and their position will not be saved within IDA database and will dissapear when IDB is reopened.



Target Modules

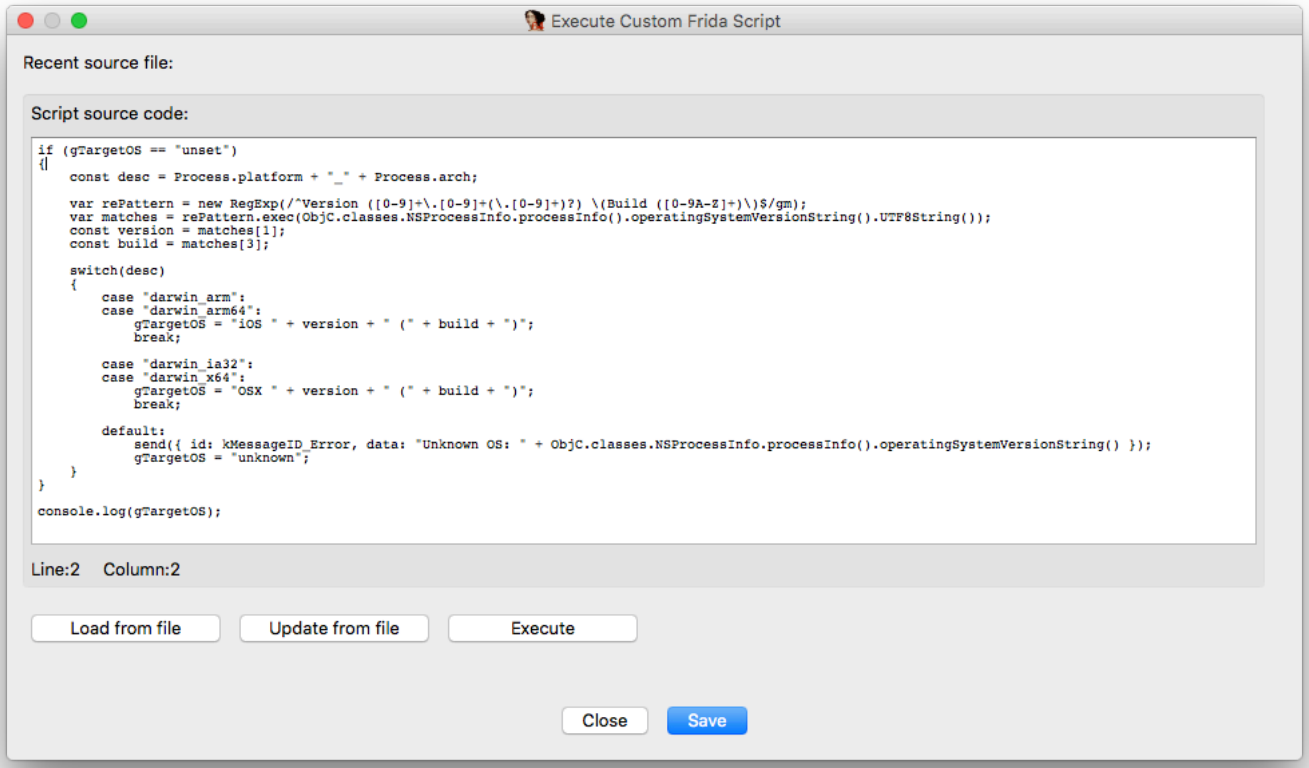
Usually all target modules are fetched as soon as connection with FRAPL is established, however it is possible to update it manually by using **Edit/Plugins/Frida Link/Fetch Target Modules**.

Accordingly, to display list of target modules, use **Edit/Plugins/Frida Link/Show Module List**.



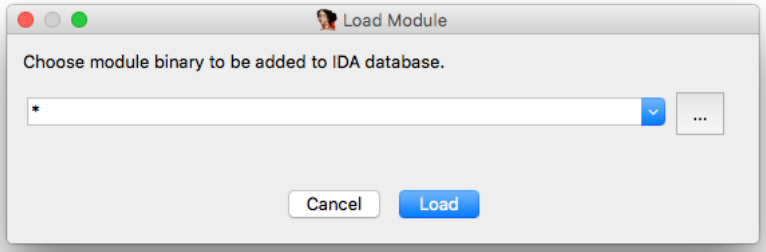
Executing custom Frida script on target

In order to execute custom Frida script on target, use **Edit/Plugins/Frida Link/Execute Frida Script**



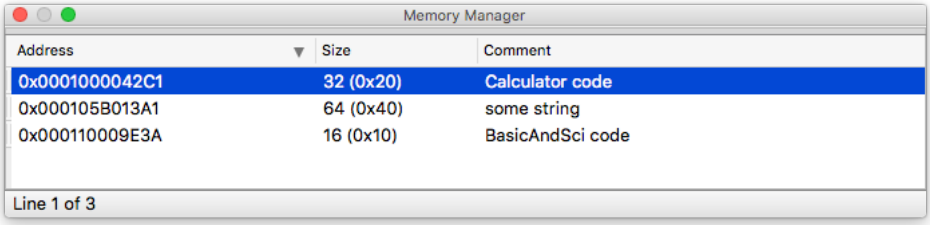
Load module

Usually user is prompted to load new module into IDB during processing backtrace, however this can also be done manually via **Edit/Plugins/Frida Link/Load Module**.

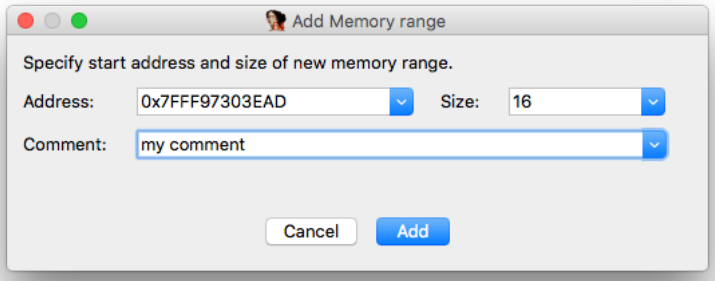


Memory Manager

To display all memory regions use **Edit/Plugins/Frida Link/Memory Manager** menu.



Right click in this reveals following opions:
Insert... - add new memory region by specifying address, size and comment (see below)



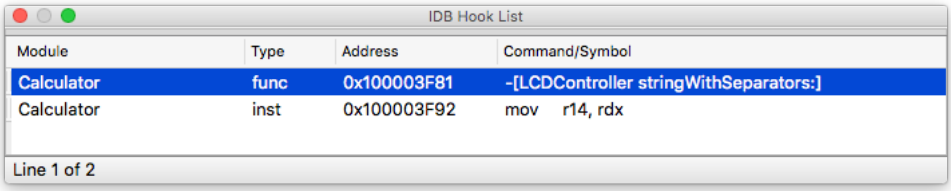
Delete - remove existing memory region from the list
Show - open selected memory region in a new window (see Memory View)

NOTE: it is actually possible to add multiple regions with the same address to be able to use it with different hooks and observe memory state in different points of time.

IDB Hook List

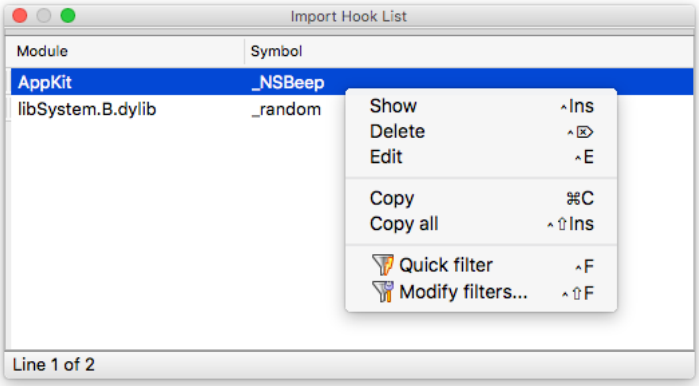
Edit/Plugins/Frida Link/Show IDB Hooks menu displays all hooks set within IDB address space.

Right click menu contains several options:
Jump To - move IDA cursor to the selected hook.
Delete - removes selected hook.
Edit - call instruction/function hook dialog (see Instruction Hooks and Function Hooks)



Import Hook List

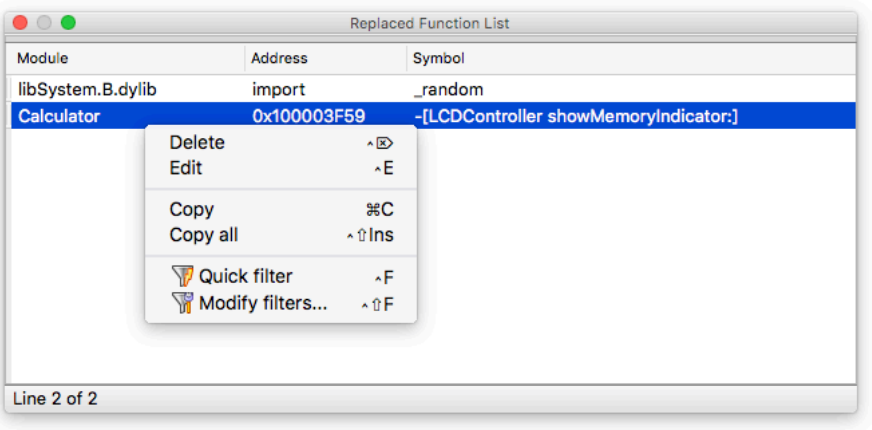
Use **Edit/Plugins/Frida Link/Show Import Hooks** to manage hooked import functions.



Show - will show CPU context and backtrace
Edit - opens function hook edit dialog
Delete - removes hook

Replaced Function List

In order to display all functions with replaced implementation select **Edit/Plugins/Frida Link/Show Replaced Funcs** from main main menu.



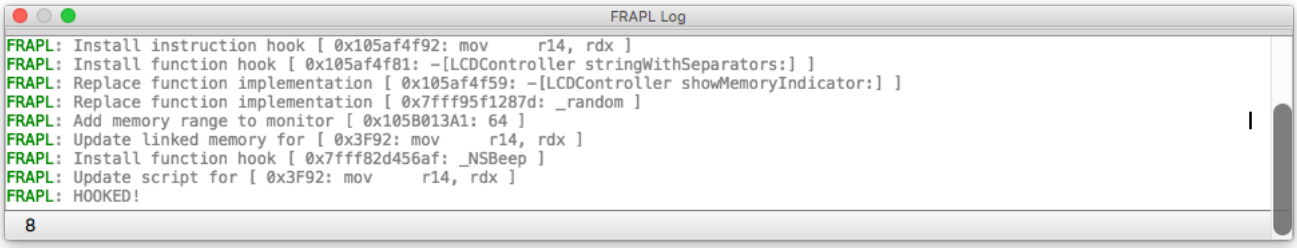
Delete - restore original implementation
Edit - open replace edit dialog

Arbitrary Hook List

*** coming soon ***

FRAPL Log

To check redirection of the FRAPL client log, use **Edit/Plugins/Frida Link/FRAPL Log**



To clear this window use "Clear" option in window's popup menu.
`FraplLog()` API adds regular log entry to this window
`FraplError()` API adds error log entry to this window

Target Log

To observe logs coming from target via **strout**, **stderr** or **NSLog** therse is a **Edit/Plugins/Frida Link/Target Log** menu.

*** coming soon ***

Debug Log Toggle

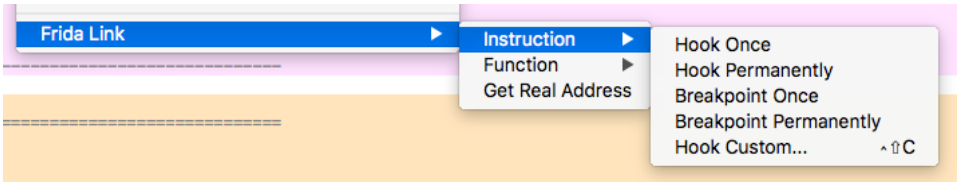
Edit/Plugins/Frida Link/Debug Log Toggle menu option enables/disables **dlog** used to debug FridaLink source.

Symbol Overwrite Toggle

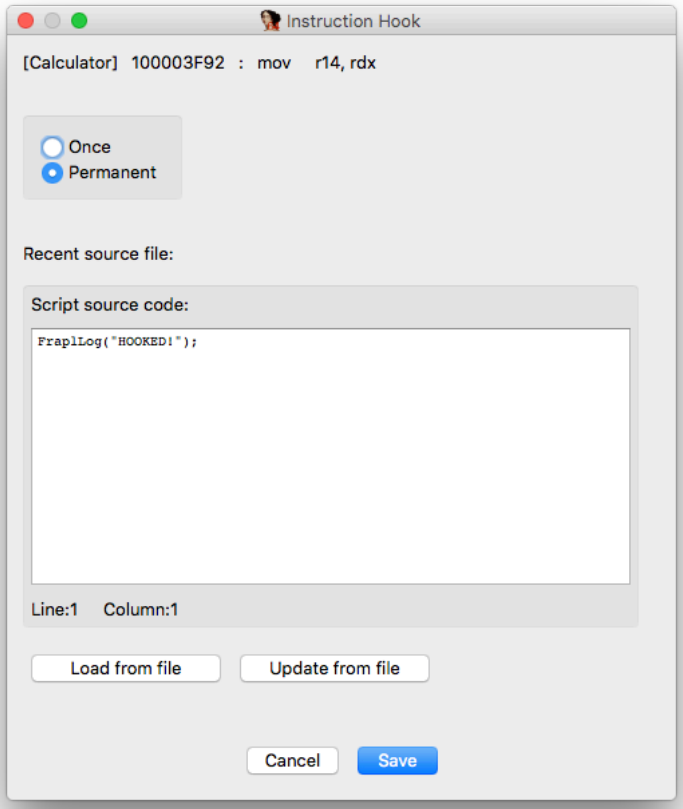
Edit/Plugins/Frida Link/Debug Log Toggle menu option allows to overwrite symbols in a backtrace from IDA database.

Instruction Hooks

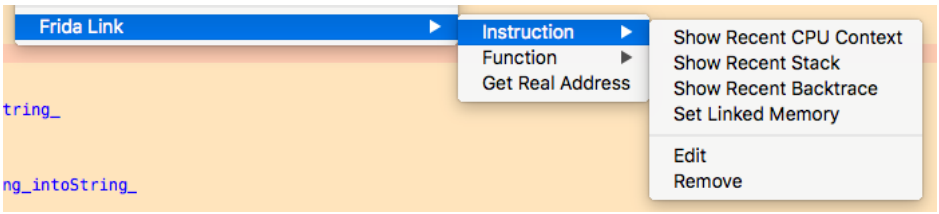
There are three options can be choosen for new instruction hook from popup menu.



Once - hooks instruction under cursor with trigger set to 'once', will be removed as soon as it is triggered
Permanent - hooks instruction under cursor permanently, i.e. hook remains after being triggered
Custom - opens ***Instruction Hook Dialog*** for additional customisation (scripts, etc)



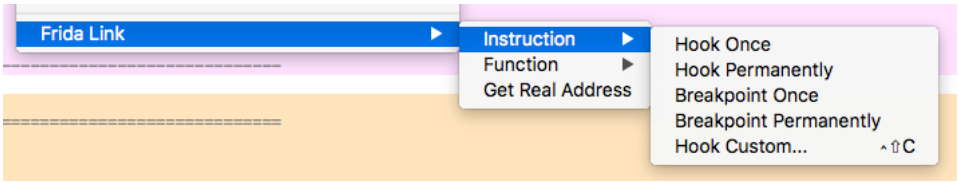
When hook is set popup menu allows to open CPU context view, Stack or Backtrace view. In order to dump particular memory range(s) when hook is triggered, use **Set Linked Memory** option.



NOTE: instruction hook on first function command will remove actual function hook if it exists

Instruction Breakpoints

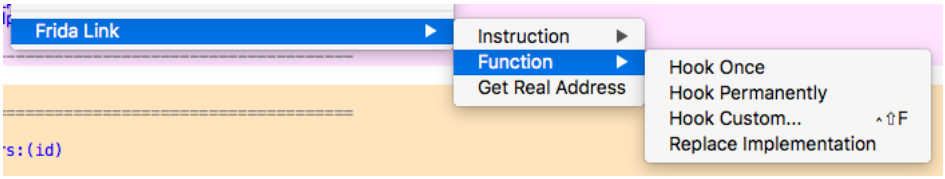
It is also possible to stop thread execution during instruction hook. To do that use **Breakpoint Once** or **Breakpoint Permanently** from instruction popup menu.



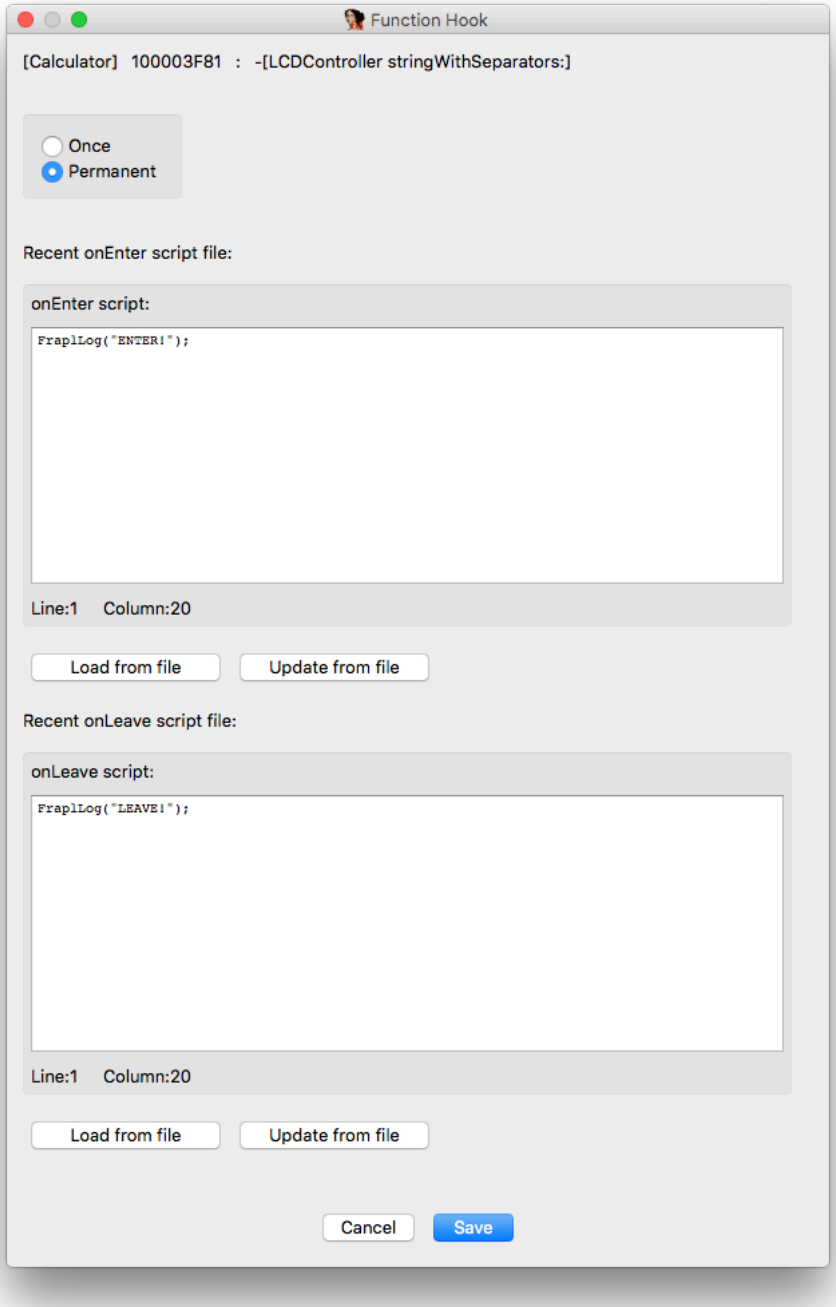
To continue execution, press **Release Breakpoint** from Main menu

Function Hooks

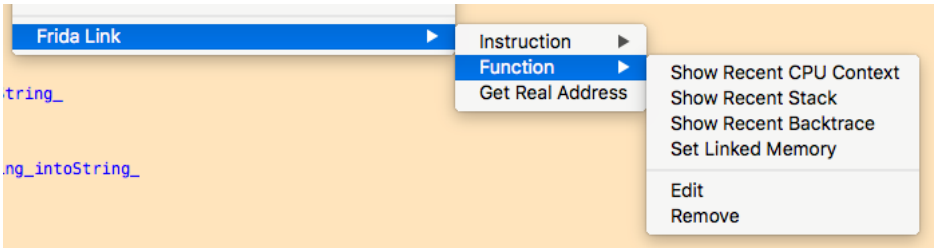
There are three option can be choosen for new function hook from popup menu.



Once - hooks function under cursor with trigger set to 'once', will be removed as soon as it is triggered
Permanent - hooks function under cursor permanently, i.e. hook remains after being triggered
Custom - opens ***Function Hook Dialog*** for additional customisation (scripts, etc)

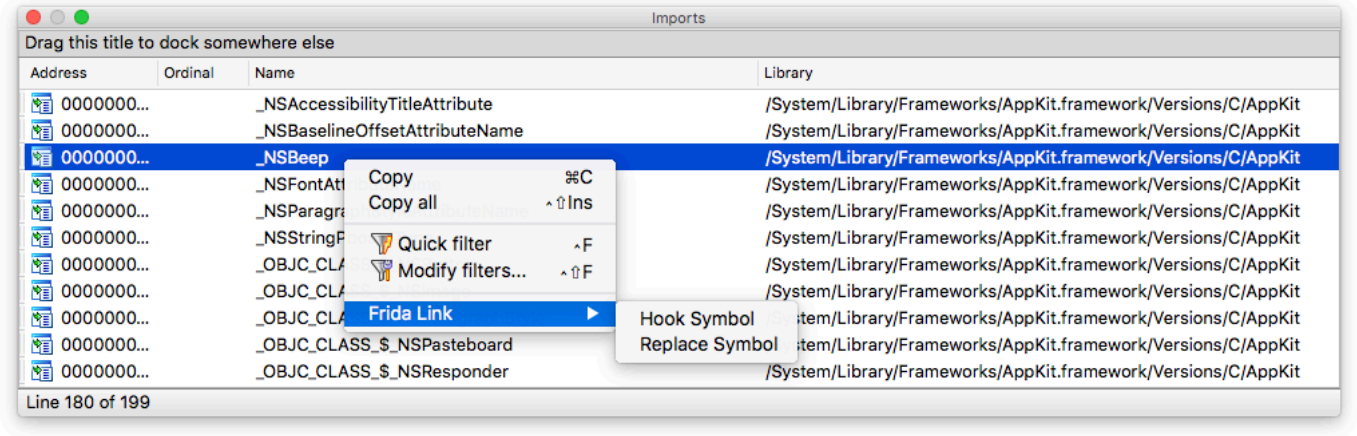


When hook is set popup menu allows to open CPU context view, Stack or Backtrace view. In order to dump particular memory range(s) when hook is triggered, use **Set Linked Memory** option.

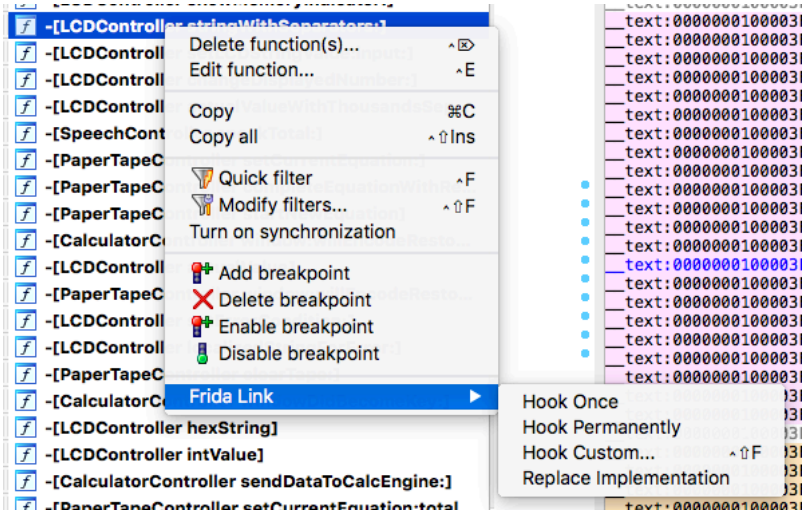


NOTE: function hook will remove instruction hook on first function command if it exists

Function hook can be set directly from **Imports View**

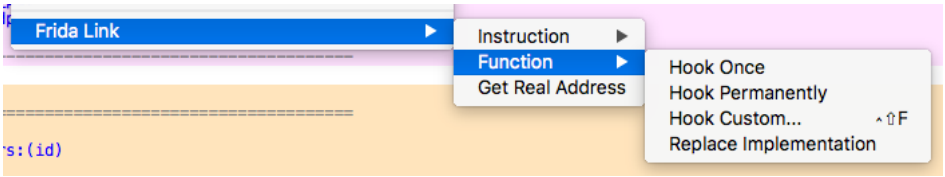


or **Functions View** popup menu.

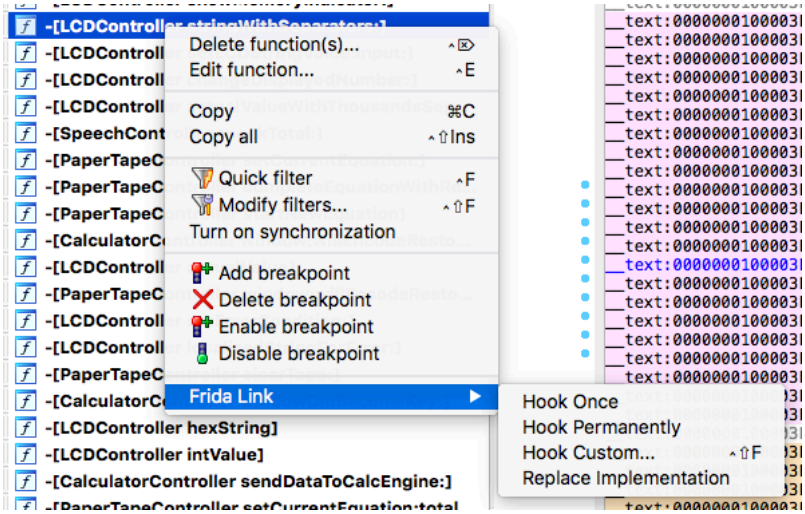


Function Implementation Replace

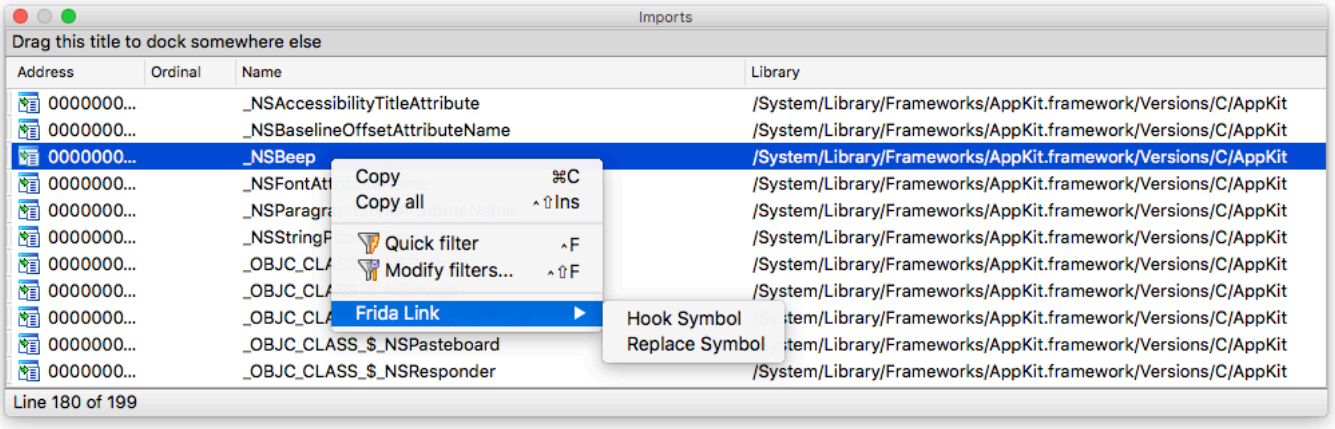
To replace function implementation completely use ***Replace Implementation*** option from **IDA View**



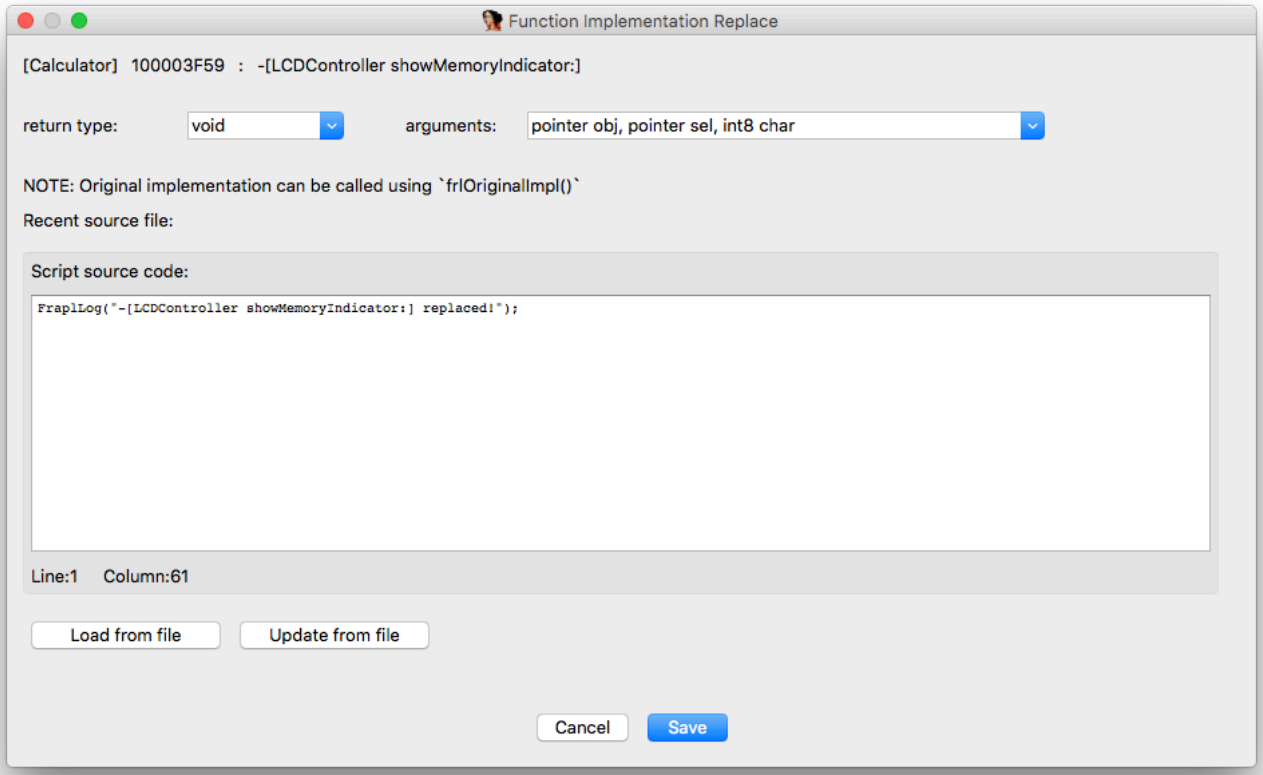
or **Functions View** popup menu.



It is also possible to replace import symbol directly by choosing **Replace Symbol** from **Imports View**.



In the function replacement dialog use **Frida** types for **return** and **agruments** fields.



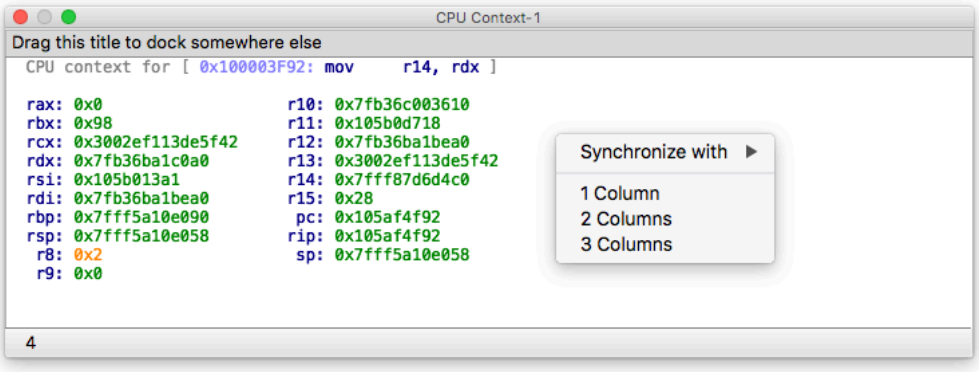
CPU context

Recent CPU context for the hook can be displayed by clicking **Show Recent CPU context** option in popup menu for the hook.

CPU context view displays current CPU state at the time when corresponding hook was triggered. It is currently possible to display registers for the following architectures:
- **ARM**
- **ARM64**
- **x64**

Changed register values will be highlighted if window is opened while hook is triggered several times.

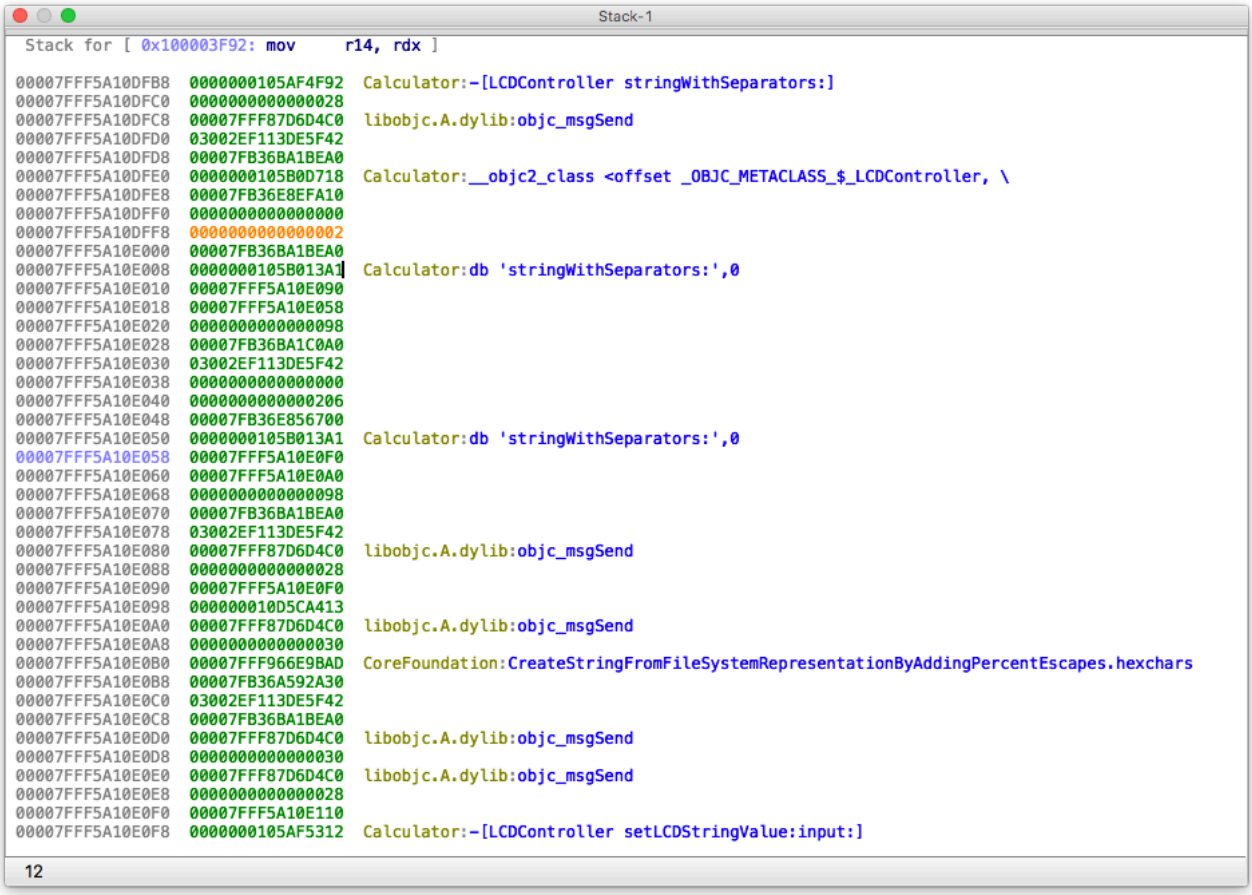
To change amount of columns use popup menu:



NOTE: default number of columns can be set in FridaLink Project Settings

Stack

Show Recent Stack option from hook popup menu brings **Stack View**.

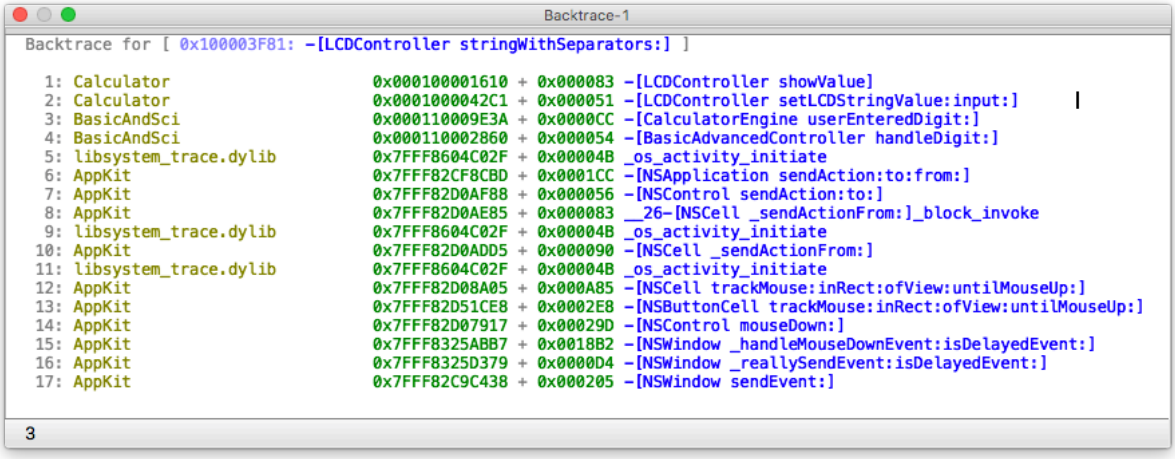


Changed values values will be highlighted if window is opened while hook is triggered several times.

Backtrace

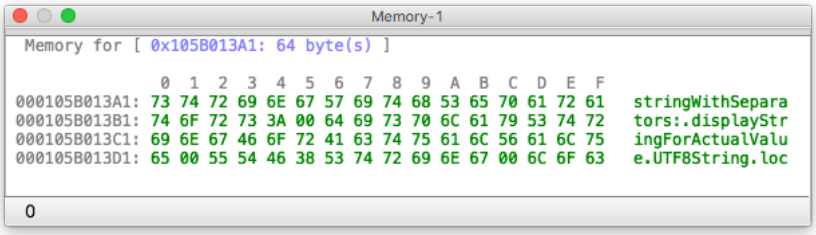
Backtrace engine is responsible for processing backtraces for FRAPL requests as well as for hook's. When this engine runs into a module which is not yet loaded into IDA database, it offers user to do so on a fly, otherwise callstack entries for these modules will not be parsed.

Recent backtrace for the hook can be displayed by clicking "Show Recent Backtrace" option in popup menu.



Memory View

When memory range is linked to some hook, it is possible to observe its recent content by choosing "Show" option from popup menu in Memory Manager.

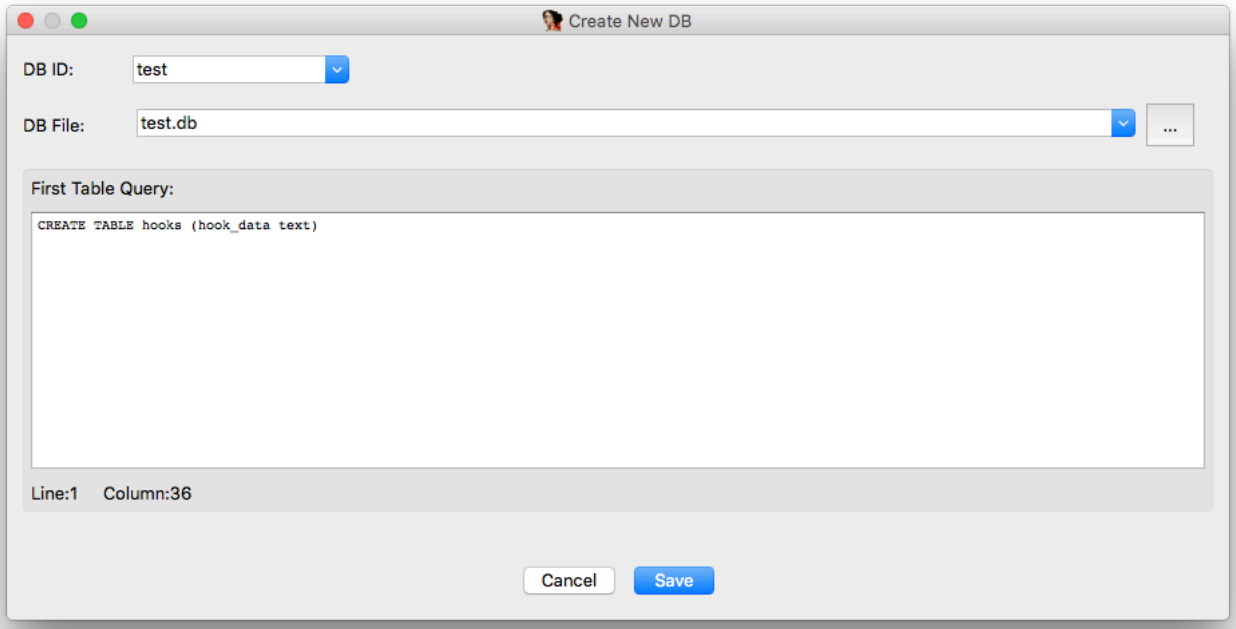


This view will be updated every time when hook is triggered. To update content manually, use 'Fetch' popup menu.

Changed bytes will be highlighted if window is opened while hook is triggered several times.

Database

To export data outside FridaLink, use DB functions from main plugin menu and FRAPL API:
Edit/Plugins/Frida Link/Create DB - create new SQLite database file



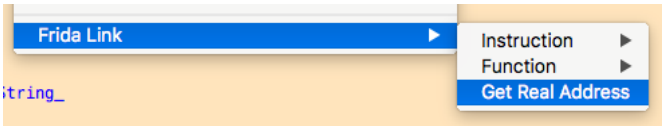
Edit/Plugins/Frida Link/Open DB - open existing SQLite database file
Edit/Plugins/Frida Link/Execute DB Query - execute query on one of opened DBs

In order to query exising DBs use **FrLExecQuery([*DB ID*] , [*QUERY*])** API from any Frida script.
This is an example code for custom hook script:

```
javascript FrLExecQuery("test", "INSERT INTO hooks VALUES (test data)");
```

Miscellaneous

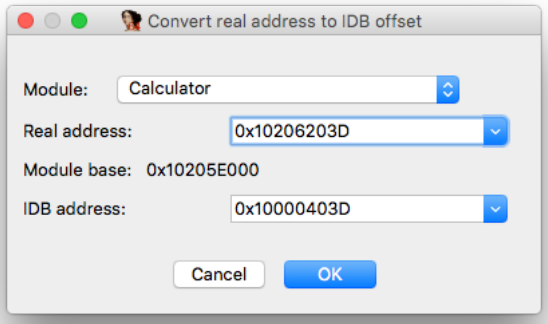
To get real address in target's memory for current IDA EA, use "Get Real Address" popup menu in IDA View:



The address conversion information will appear in FRAPL Log.

```
FRAPL: [ Calculator ] 0x100004043 => 0x1004B8043 cmp     rbx, rax
FRAPL: [ BasicAndSci ] 0x1100056DC => 0x107E1F6DC mov     rsi, cs:selRef_substringWithRange__0
```

To get IDB EA from real target's address use **Show Address Converter** option from Main menu.



Helper API

FRAPL and Frida API are available inside FridaLink custom code. The most popuplar functions are listed below:

`FraplLog([string])` - adds **regular** log entry to this window

`FraplError([string])` - adds **error** log entry to this window

`FrLExecQuery([db_id], [query])` - execute **[QUERY]** on database with id = **[DB_ID]**

`FrLBreakPoint()` - stop thread execution until **Release Breakpoint** is called.

`ResolveImportSymbol([module], [symbol], [return], [args])` - resolve symbol, for example:

```
javascript const CFDataGetBytePtr = ResolveImportSymbol("CoreFoundation", "CFDataGetBytePtr", "pointer", ["pointer"]); var data = CFData
```

`schedule_sync([queue], [work])` - scheduler function synchronously on **frapl queue**

```
javascript schedule_sync(fraMainQueue, function () { // your code here });
```

`GetWordSize()` - get size of word for current architecture

`GetTargetOS()` - get platform version (for example `OSX 10.11.4 (15E65)`)

`UintArrayToHex([ua], [prefix])` - dump uint array **[ua]** into hex string with prefix **[prefix]**