FIRST IDA Pro Integration

Release Closed BETA

Angel M. Villegas

May 03, 2023

IDA Pro Integration

CHAPTER 1

Installing Plugin

Since FIRST is an IDA Python plugin it only works with a license version of Hex Ray's IDA Pro. Due to the integrations with IDA Pro there is a minimum version number. The FIRST plugin only works with IDA 6.9 (service pack 1), relased May 2016, and higher.

Important: It is easier to install Python from Python.org with the latest 2.7 build instead of using the outdated version of Python bundled in with IDA Pro.

Attention: There are many ways to install FIRST, the quickest way is to use **`pip`** and run the setup script. The setup's location differs depending on the OS being used and other possible configurations. The defaults for Mac and Windows are below.

Mac

```
$ pip install first-plugin-ida
$ /usr/local/bin/first-plugin-ida
```

Windows

```
> pip install first-plugin-ida
> C:\Python27\Scripts\first-plugin-ida
```

To use FIRST, you will need to download the plugin and save it to the Hex Rays IDA Pro plugin folder. Directions for this differ depending on the operating system and a basic guide can be found below.

FIRST is available on PyPI, so to use it you can use **`pip`**:

```
$ pip install first-plugin-ida
```

Alternatively, if you don't have setuptools installed, download it from PyPi and run

```
$ python setup.py install
```

To use the bleeding-edge version of FIRST's IDA Pro Integration, you can get the source from GitHub and install it as above:

```
$ git clone git://github.com/vrtadmin/FIRST-plugin-ida
$ python setup.py install
```

Once first-plugin-ida is installed with pip, the post installation script needs to be executed. The script simply copies over the plugin and its files to the IDA Pro installation of your choosing. Depending on your system setup, configuration, and user privileges you may need to be admin or root to successfully use the script.

OS	Default Path
Mac	/usr/local/bin/first-plugin-ida
Windows	C:\Python27\Scripts\first-plugin-ida

The script will ask you for the full path to the IDA Pro installation. Providing it will copy the plugin to IDA Pro and its dependencies. The default location for IDA Pro installations are outline below.

OS	Default Path
Mac	Applications/IDA\ Pro\ 6.9/idaq.app/Contents/MacOS/plugins
Windows	C:\Program Files (x86)\IDA 6.9\plugins

Once the script completes without any errors you will be able to use FIRST in IDA Pro.

1.1 Manual Installation

If you do not wish to use pip or the post installation script then FIRST can be installed manually. To install the plugin you will need to get the plugin's source from GitHub. The source for the plugin includes every file in the FIRST-plugin-ida/first_plugin_ida folder except FIRST-plugin-ida/first_plugin_ida/__init__.py file. All other files need to be copied over to IDA Pro's plugins directory. Depending on the OS IDA is running on you may need to copy over other dependencies to IDA Pro's folders.

1.2 Requirements

Additionally, FIRST requires one third party module to work and an optional module if Kerberos Authentication is used

- [Required] Requests (https://pypi.python.org/pypi/requests)
- [Optional] Requests-kerberos (https://pypi.python.org/pypi/requests-kerberos)

1.3 Windows

Once you have a copy of the plug-in, installing the plug-in is as simple as copying the Python file into the plugins folder. For IDA 6.9, the default installation path can be found at:

Default IDA Pro Path	C:\Program Files (x86)\IDA 6.9\plugins
Dependency Instructions	pip install requests

1.4 Mac OS X

Installing on Mac OS X requires a little more work, once you've installed IDA Pro, copy the FIRST plugin to the <installed_path>/Contents/MacOS/plugins/ folder and the required dependencies to <installed_path>/Contents/MacOS/python/

Default IDA Pro Path	/Applications/IDA\ Pro\ 6.9/idaq.app/Contents/MacOS/plugins/
Dependency Instructions	pip install requests cp /usr/local/lib/python2.7/site-packages/requests* /Applications/IDAPro6.9/idaq.app/Contents/MacOS/python

1.5 Linux

During the setup, IDA asks whether to install with the bundled Python interpreter or use the local Python interpreter from the local system. Bundled Python is nice, everything just works by default. However the downside is that you can't really add and upgrade Python libraries, and the FIRST plugin requires the requests plugin which is not by default in the bundle.

If you installed with bundled Python, you can switch to use the local Python simply by renaming `libpython2.7.so.1.0` and `python/lib/python27.zip`. For instance:

```
$ cd $IDA_DIR
$ mv libpython2.7.so.1.0{,.orig}
$ mv python/lib/python27.zip{,.orig}
```

You can revert back to bundle Python by reverting the renames.

Unfortunately the downside of using local Python is that if you are running under x86_64, IDA being a 32-bit binary, it won't work out of the box. Fortunately, under recent Debian (e.g. stretch) and Ubuntu, you can install libs from other architectures. For instance, what worked for us:

```
$ dpkg --add-architecture i386
$ apt update
$ apt install --no-install-recommends gtk2-engines-murrine:i386 gtk2-engines-
↪pixbuf:i386 libc6-i686:i386 libcurl3:i386 libdbus-1-3:i386 libexpat1:i386␣
↪libffi6:i386 libfontconfig1:i386 libfreetype6:i386 libgcc1:i386 libglib2.0-0:i386␣
↪libgtk2.0-0:i386 libice6:i386 libpcre3:i386 libpng16-16:i386 libpython2.7:i386␣
↪libsm6:i386 libstdc++6:i386 libuuid1:i386 libx11-6:i386 libx11-xcb1:i386␣
↪libxau6:i386 libxcb1:i386 libxdmcp6:i386 libxext6:i386 libxi6:i386 libxrender1:i386␣
↪python-pyqt5:i386 xdg-utils zlib1g:i386 python-requests
```

Tip: if your distro is different or too old, try a chroot (e.g. debootstrap & schroot), works pretty well.

Copy the FIRST plugin (`first.py`) to your plugins directory (`~/.idapro/plugins`) and start IDA (32 or 64), it should all work!

CHAPTER 2

Configuring

FIRST plugin can be configured in three ways; the Welcome dialog box when initially installed, the plugin main window and manually modifying the config file (no recommend for most).

2.1 Initial Installation

When FIRST is initially installed and IDA is loaded a *Welcome* dialog box will appear. This only appears when FIRST is not configured.

Notice: public server moved to first.talosintelligence.com:443.

Fig. 1: FIRST welcome dialog box (with invalid API key)

The configuration requires a valid API key that can be obtained from the server (go to first.talosintelligence.com to register with public server). Make sure to test the connection and select **Save**

2.2 Plugin Main Window

To access the plugin's main window, users can either press **1** from a IDA Pro View window or by the IDA Pro Edit menu (Edit > Plugins > FIRST). Once the main window is open select the **Configuration** tab from the left panel. From here the connection details can be updated. Selecting the **Test** button will test if a connection to FIRST can be made. However, it does not save and use that server connection until the **Save** button is selected.

Fig. 2: FIRST Server Configuration (with invalid API key)

CHAPTER 3

Adding Annotations

To add your own annotations for a function or several functions to FIRST you can use the right click menu in the IDA View window. Simply right click anywhere in the IDA View window and select **Add this function to FIRST** or **Add multiple functions to FIRST**.

Fig. 1: FIRST's IDA Integration Right Click Menu

3.1 Adding a Single Function

To add a single function, right click any where within the IDA View window of a valid function.

Note: A valid function is a function that IDA Pro was able to interpret as a function. If you are viewing disassembly in Graph mode them anywhere in the window will is valid. However, if you are viewing disassembly in Text mode then the address corresponding to the instruction selected should be associated with a function (this is indicated by the black color instruction address' text).

Once the popup menu is shown, select **Add this function to FIRST**. The Add function metadata dialog box will pop up with the annotations associated with the function. The annotations cannot be modified in this dialog box, it is only for review purposes. If the annotations need to be changed then close the dialog box and update the function in IDA and add the function once the function annotations are to your liking. After reviewing the annotations select the **Upload** button. If no errors occured your function annotations will be added to FIRST and the Output window will state how many functions were added to FIRST.

Fig. 2: Add a Single Function to FIRST

3.2 Adding Multiple Function at Once

To add multiple functions, right click any where within the IDA View window. Once the popup menu is shown, select **Add multiple functions to FIRST**. The Mass Function Upload dialog box will pop up with a list of a majority, if not all, functions defined in IDA Pro.

Note: FIRST skips any function that is a wrapper for a jmp instruction.

```
.text:100023C5 sub_100023C5    proc near
.text:100023C5                 jmp     short loc_100023D6
.text:100023C7 sub_100023C5    endp
```

The dialog box uses a couple of different colors to signify various properties associated with a given function, its state or changes in its metadata.

Color Property	Meaning
Changed	The function's metadata has changed. This will apply in a couple of situations; [1] if the function previous had metadata from FIRST applied to it but you changed it in some way; [2] You've uploaded your annotations to FIRST and then altered them in IDA without adding again, [3] not metadata is associated with the function and you added your annotations. These functions, most likely, should be added to FIRST.
Unchanged	The function's metadata has not changed from the last metadata applied to it. This is reserved for functions that have metadata applied to it. If no metadata is applied to the function then it will likely show up as Default.
Default	The function's metadata has not been altered from IDA Pro's auto analysis. This will apply to all functions with the **sub_** prefix and Library functions detected by IDA's FLIRT signatures.
Selected	The function has been selected to be added to FIRST. To select a function, just click on it in the list of function. To deselect a function, clikc on it once more.

The dialog box also provides a way to quickly filter out any function starting with **sub_** and to select all functions displayed in the list. Simply select the checkbox associated with the operation you wish and the function list will be updated to reflect it.

Note: Selecting **Filter Out "sub_" functions** and **Select All** in either order will cause only the function not starting with **sub_** to be selected.

After reviewing your selection, select the **Upload** button. If no errors occured your function(s) annotations will be added to FIRST and the Output window will state how many functions were added to FIRST.

Fig. 3: Add Multiple Functions to FIRST

3.3 Data Collected and Sent to FIRST

Through IDA Pro's IDA Python APIs FIRST will retrieve several bits of information for the function to send back to the server. The information will be used by the system to uniquely identify the function and supply Engines with enough information to execute their algorithms. The data sent sent can be found below

Table 1: Data sent to FIRST server

Data	Description
Function Name	Reverser's created annotation.
Function Prototype	Reverser's created annotation.
Function Repeatable Comment	Reverser's created annotation. This is the repeatable function comment (key press: ;), not a function comment (key press: Shift + ;)
Opcodes	All the opcodes assocaited with the function. The opcodes are retrieved in order of lowest address to highest address. This is based off of IDA's ability to determine function boundries by its basic blocks
Architecture	The architecture the function is executed on (i.e. intel32, intel64, arm32, etc.)
IAT Function Calls	The functions called from this function that are exports from other samples and in the samples IAT.
Sample's MD5	The MD5 of the sample, IDA calculates this when the sample is initially loaded. The original sample is not required and the MD5 is retrieved through IDA's API.
Sample's CRC32	The CRC32 of the sample, IDA calculates this when the sample is initially loaded. The original smaple is not required and the CRC32 is retrieved through IDA's API.

CHAPTER 4

Deleting Your Annotations

Important: It is important to note you can only delete annotations you've created. Applying someone else's annotations to a function and modifying it will make a new annoations tied to you and not the original creator.

To manange the annotations you've created and have added to FIRST open the FIRST plugin main window. To access the plugin's main window, users can either press **1** from a IDA Pro View window or by the IDA Pro Edit menu (Edit > Plugins > FIRST). Once the main window is open select the **Management** tab from the left panel. The plugin will query the server for all metadata you have added. This data will be populated in a tree view to allow you to view an organized list of your metadata. To delete an annotation, either right click and select **Delete** or press the Delete key.

Fig. 1: Query FIRST for All Functions

CHAPTER 5

Checking for Annotations

To check for a single or all functions in your IDB, you can use the right click menu in the IDA View window. Simply right click anywhere in the IDA View window and select **Check FIRST for this function** or **Query FIRST for all function matches**.

Fig. 1: FIRST's IDA Integration Right Click Menu

The dialog boxes associated with querying FIRST for one or all functions are similar in a lot of ways. Both use two colors to signify if the match is currently applied and what match is selected.

Color Property	Meaning
Applied	The match is currently applied to the function it is a match for. This does not mean your local IDB is synced up with the latest version of the metadata associated with the match. Selecting it again will ensure the latest annoations are applied to your IDB
Selected	The function has been selected to be added to FIRST. To select a function, just click on it in the list of function. To deselect a function, clikc on it once more.

5.1 Query for Single Function

To check FIRST for a single function, right click any where within the IDA View window of a valid function.

Note: A valid function is a function that IDA Pro was able to interpret as a function. If you are viewing disassembly in Graph mode them anywhere in the window will is valid. However, if you are viewing disassembly in Text mode then the address corresponding to the instruction selected should be associated with a function (this is indicated by the black color instruction address' text).

Once the popup menu is shown, select **Check FIRST for this function**. The Check Function dialog box will pop up and any matches (exact or similar) will be displayed in a tree structure.

Fig. 2: Query FIRST for Single Function

5.2 Query for All Functions

To query FIRST for all functions in the IDB, right click any where within the IDA View window. Once the popup menu is shown, select **Query FIRST for all function matches**. The Check all Functions dialog box will pop up with a list of matches for functions within the IDB. While results are being populated, you can go through an look at each match.

The tree view is used to display the data in a meaniful way. The top most layer shows the address and current function name the matches correspond to. The next column displays the number of matches to that function. Expanding that node will show the matches and their details. Selecting to expand a match node will display the comment associated with that match.

Note: Selecting **Show only "sub_" functions** and **Select Highest Ranked** in either order will cause only the function starting with **sub_** to be selected.

FIRST will not query for all functions at once. Instead all functions are grouping in sets of 20 and each set is set to the server. This will allow results to be returned and the researcher to start looking over the results. As new matches are returned the data will be added.

To help understand if the match makes sense, right click anywhere within the function tree node or its matches, and click **Go to Function**. This will focus the last used IDA View window on the function in question.

After reviewing your selection, select the **Apply** button. If no errors occured the selected matches will be applied to your function(s) and the Output window will state how many functions were applied with FIRST annotations.

Important: If multiple functions attempt to apply the same match, then only the function first applied with the annotations will change. IDA Pro will prompt you with each additional function that the metadata cannot be applied to. The Output window will display the addresses of the functions that metadata couldn't be applied to.

Danger: Selecting to apply annotations to your functions will overwrite any annotations currently applied. There is no UNDO for this operation. Also, when applying a function metadata to your IDB's function, this could result in incorrect function prototypes (for similar and possibly exact matches).

Fig. 3: Query FIRST for All Functions

CHAPTER 6

View Metadata History

The history for metadata can be viewed in several different places. - A match in a Query FIRST operation (Check Function or Check All Functions dialog box) - The management and currently applied view of the main FIRST plugin window - A function with FIRST metadata applied from the IDA View window via the right click menu

The Check Function/Check All Functions dialog boxes and Currently Applied/Management views use a tree structure to display function annotations. Right click anywhere within the record for an annotation to open the popup menu. In this menu select the **View History** operation and a new dialog box will open up displaying the revisions made to the annotations by the original author.

Fig. 1: View History from Check Function Dialog Box

Fig. 2: View History from Check All Functions Dialog Box

Fig. 3: View History from Currently Applied View

When annotations from FIRST are applied to a function, a new right click menu option becomes available. The **View metadata history** operation will open a dialog box with the revision history.

Fig. 4: View History from Management View

Fig. 5: View History from Function with FIRST Annotations

CHAPTER 7

Currently Applied

Some times during analysis it can be very useful to know what annotations from FIRST where applied to the IDB. Additionaly, it could help quickly track down some functionality of interest. To aid in this, a user can go to the main FIRST plugin in window (press **1** from the IDA View window or through IDA Pro's menus: **Edit** > **Plugins** > **FIRST**). Once at the main FIRST plugin window, select the **Currently Applied** tab from the left panel.

Fig. 1: Go to Function from Currently Applied

CHAPTER 8

Scripting FIRST in IDA

CHAPTER 9

IDA Integration FIRST API

The below shows autodocs?? Yay, It works!!

class first_plugin_ida.first.**FIRST**

> **static initialize**()
>
> Initializes FIRST by installing hooks and populating required data strucutres.

9.1 FIRSTUI :: Generic

class first_plugin_ida.first.**FIRSTUI**

> **class Generic**
>
> > **get_server_thread**()
> >
> > **init_bottom_layout**()
> >
> > **init_data_layout**()
> >
> > **init_middle_layout**()
> >
> > **init_signals**()
> >
> > **init_top_layout**()
> >
> > **init_window**()
> >
> > **setupUi**(*dialog*)

9.2 FIRSTUI :: Welcome

class first_plugin_ida.first.**FIRSTUI**

class Welcome

init_data_layout()

init_middle_layout()

init_signals()

init_top_layout()

init_window()

show()

9.3 FIRSTUI :: Check

class first_plugin_ida.first.**FIRSTUI**

class Check

init_data_layout()

init_middle_layout()

init_signals()

init_top_layout()

init_window()

tree_clicked(*index*)

9.4 FIRSTUI :: CheckAll

class first_plugin_ida.first.**FIRSTUI**

class CheckAll

Check all docs

custom_menu(*point*)

filter_only_subs()

get_groups()

get_server_thread()

init_bottom_layout()

init_data_layout()

init_middle_layout()

init_signals()

init_top_layout()

init_window()

metadata_history(*metadata_id*)

select_highest()

tree_clicked(*index*)

9.5 FIRSTUI :: Upload

class first_plugin_ida.first.**FIRSTUI**

class Upload

init_bottom_layout()

init_data_layout()

init_middle_layout()

init_signals()

init_top_layout()

9.6 FIRSTUI :: UploadAll

class first_plugin_ida.first.**FIRSTUI**

class UploadAll

filter_sub_callback(*value*)

get_selected_data()

init_bottom_layout()

init_data_layout()

init_middle_layout()

init_signals()

init_top_layout()

init_window()

select_all_callback(*value*)

table_clicked(*index*, *table_view*, *data_model*)

9.7 FIRSTUI :: Requests

class first_plugin_ida.first.**FIRSTUI**

> **class Requests**
>
> > **class Callback**(*func*, ***kwargs*)
> >
> > **class MsgBox**(*title*, *msg*, *icon=<sphinx.ext.autodoc.importer._MockObject object>*)
> >
> > **class Print**(*msg*)

9.8 FIRSTUI :: History

class first_plugin_ida.first.**FIRSTUI**

> **class History**(*metadata_id*)
>
> > **init_bottom_layout**()
> >
> > **init_data_layout**()
> >
> > **init_middle_layout**()
> >
> > **init_signals**()
> >
> > **init_top_layout**()
> >
> > **init_window**()
> >
> > **utc_to_local**(*utc_str*)

9.9 FIRSTUI :: SharedObjects

class first_plugin_ida.first.**FIRSTUI**

> **class SharedObjects**
>
> > **static get_config**(*obj*)
> >
> > **static make_match_info**(*match*, *full=True*, *check_all=True*)
> > Build a tree item for a function_ea node (level-1) This is the function match information (name, prototype, rank) @param function_context: a dbFunction_Context object @return: QStandradItemModel item for the function context
> >
> > **static make_model_headers**(*model*, *full=True*, *check_all=True*)
> > Set the model horizontal header data @param model: the QStandardItemModel which headers should be set
> >
> > When full is set to False this mean the headers are for the user to review metadata they've created.
> >
> > **static server_config_layout**(*obj*, *outer_layout*, *config=None*)
> > Server Configuration GUI components

static test_connection(*obj*)

9.10 FIRST :: Callbacks

class `first_plugin_ida.first.`**FIRST**

class Callbacks

Callbacks for FIRST's Dialog UI components.

This class contains only static methods and should be accessed as such.

class Update

Updating callback class.

This class is basic and sets up data and complete callbacks for the add operation.

Parameters dialog (`FIRSTUI.Update`) – Dialog box the accepted button was selected.

class Upload(*dialog*)

Uploading/Adding callback class.

This class is basic and sets up data and complete callbacks for the add operation.

Parameters dialog (`FIRSTUI.Upload` or `FIRSTUI.UploadAll`) – Dialog box the accepted button was selected.

message

Format string for the wait box message.

Type `str`

message = 'Uploading metadata for {0} function(s)\n {1}% complete'

static accepted(*fclass*, *dialog*)

Registered callback for accept dialog action.

Parameters

- **fclass** (`idaapi.PluginForm`) – The plugin form part of
- **dialog** (`FIRSTUI.*`) – A dialog box object.

static check(*dialog*)

Check and CheckAll dialog box handler.

Parameters dialog (`FIRSTUI.Check` or `FIRSTUI.CheckAll`) – Check or CheckAll dialog box.

static welcome(*dialog*)

Welcome dialog box handler.

Parameters dialog (`FIRSTUI.Welcome`) – Welcome dialog box.

9.11 FIRST :: Configuration

class `first_plugin_ida.first.`**FIRST**

class Configuration(*config=None*)

Class containing configuration details for FIRST.

Parameters config (`RawConfigParser`) – Configuration details for plugin.

api_key

The user's API key.

Type `str`

`auth`
Authenication used with FIRST (default: None).
Type `HTTPKeberosAuth`

`authentication`
Flag set if authentication is used in connection.
Type `bool`

`load_config`(*config*)
Loads configuration details into this instance.
Parameters **`config`** (`RawConfigParser`) – The configuration details to load.

`port`
80)

Type `int`
Type The FIRST server port (Default

`protocol`
The TCP protocol used to communicate with FIRST.
Type `str`

`save_config`(*config_path*)
Saves the configuration set in this instance to disk.
Parameters **`config_path`** (`str`) – File path to save configuration.

`server`
The FIRST server.
Type `str`

`set_api_key`(*key*)

`set_authentication`(*_authentication*)

`set_data`(*key*, *value*)
Sets a specific configuration setting.
Parameters
- **`key`** (`str`) – The configuration setting.
- **`value`** (`str`) – The configuration setting value.

`set_port`(*_port*)

`set_protocol`(*_protocol*)

`set_server`(*_server*)

`set_verify`(*_verify*)

`verify`
Whether the SSL cert will be verified.
Type `bool`

9.12 FIRST :: DB

class `first_plugin_ida.first.`**`FIRST`**

class DB
FIRST DB Class

Provides functions to save data to and retrieve data from IDA's IDB backend. Additionally, it contains functions for calculating the index functions should be saved to in the IDB to provide constant time lookups.

This class contains only static methods and should be accessed as such.

`record_size`

The number of bytes that can be saved into one index in the IDB's array. Once the number of bytes are hit the record is split and will continue in the next index.

Note: IDA enforces a hard limit of 1024, setting this value higher than that will result in information loss.

Type `int`

`max_records`

Determines how many array indices can be used to store data for a given function.

Note: If this number is increased and there is enough data to use all the indices, this could result in over writting other FIRST function data saved in the IDB.

Type `int`

`static get_function`(*address=None*, *function=None*)

Retrieves function and all its details from the IDB DB.

The data returned here may not match the current state of the IDB. Either the address or function argument should be provided. Providing neither will result in a return value of None.

Parameters

- **`address`** (*int*, optional) – The start address of the function.
- **`function`** (`MetadataShim`, optional) – The current MetadataShim object for the function.

Returns

If function exits and is saved it is returned.

None: On failure.

Return type FIRSTMetadata

`static get_index`(*function*)

Computes the base index for the function.

The index computed by thios function is index into an IDB array.

Parameters **`function`** (`MetadataShim`) – The function to get an index for.

Results: int: The index into the array.

None: On failure.

`static get_tag`(*function*)

Calculates and returns the tag for the given function.

Function that will return array id corresponding to the array with the function data in it if the array exists. If the array does not exist then it is created and the created array id is returned.

Parameters **`function`** (`MetadataShim`) – The function to get a tag for.

Results: int: The array ID on success.

None: On failure.

`max_records = 16`

`record_size = 1024`

`static save`(*functions*)

Saves one or more functions to the IDB DB.

This function can be used to save one or more FIRSTMetadata objects to the IDB's database.

Parameters **`functions`** (*FIRSTMetadata* or *list* of *FIRSTMetadata*) –

Returns None

9.13 FIRST :: Info

`class` `first_plugin_ida.first.`**`FIRST`**

`class Info`

Information gathering functions.

Will get different information required by FIRST to interact with server or other plug-in side operations.

This class contains only static methods and should be accessed as such.

`processor_map`

Dictionary mapping between IDA's naming convention to FIRST's.

Type `dict`

`include_bits`

List of processors that should include the number of bits.

Type `list`

`static get_apis`(*address*)

Returns a list of all APIs used by a function.

The address provided will be used to get a function and each instruction in the function is examined for APIs in the sample's IAT.

Parameters **`address`** (*int*) – An address associated with a function. The address can be any address within the function.

Returns Empty list or list of *MetadataShim* objects

Return type list

`static get_architecture`()

Returns the architecture the sample is built for.

The values are normalized for the FIRST server. It altered then FIRST will not match on other functions with the same architecture.

Returns

str. String representation of the architecture associated with the sample. Examples: intel32, intel64, arm32, mips, etc.

`static get_file_details`()

Returns details about the sample.

The MD5 and CRC32 fields will always be returned since IDA Pro provides that information. If the IDB is created with the original sample then the sample will be hashed to get the SHA1 and SHA256. All tthe data is stored in the IDB to prevent getting the information multiple times.

Returns dict. Dictionary of file hashes and CRC32.

`include_bits = ['intel', 'arm']`

`static is_32bit`()

Returns if the sample is 32bit or not.

Returns True is 32bit or False.

Return type bool

processor_map = {'metapc': 'intel'}

static set_file_details(*md5*, *crc32*, *sha1=None*, *sha256=None*)

Sets details about the sample.

This is a work around for situations where there is no original sample on disk that IDA analyzes. FIRST requires a MD5 and CRC32 to store functions, without it the function will not be saved.

Parameters **md5** (`str`) – Valid MD5 hash

static signature(*address*)

Returns opcodes for the function the address is associated with.

Given a virtual address, this function will return it in a series of bytes or None. The opcodes are ordered in address ascending order.

Parameters **address** (*int*) – An address associated with a function. The address can be any address within the function.

Returns

A string of binary data on success.

None: On failure.

Return type str

9.14 FIRST :: Metadata

class first_plugin_ida.first.**FIRST**

class Metadata

Class containing Misc Metadata functions.

Contains helper functions that will allow interaction with the memory list containing all functions within the IDB.

This class contains only static methods and should be accessed as such.

static get_function(*function_address*)

Get the MetadataShim object for a given function.

Parameters **function_address** (*int*) – A functions start address. The value should be the start address of the function or else the function will return None.

Returns

object on success.

None on failure.

Return type MetadataShim

static get_functions_with_applied_metadata()

Returns a list of functions with FIRST metadata applied to it.

Returns Empty list or list of *MetadataShim* objects

Return type list

static get_non_jmp_wrapped_functions()

Returns a list of functions addresses

Functions definited in the IDB, from auto analysis or manually definited, are part of the list returned. Functions that are just wrappers with a jmp instruction are not included.

Returns

Empty list or list of integer values

The list of integer values correspond to a function's start address

Return type list

static get_segment_functions(*segment*)

Returns functions for a given segment.

Parameters segment (*segment_t*) – The segment functions will be returned from. segment_t objects are returned from IDA's getseg API.

Returns

Empty list or list of MetadataShim objects on success.

None: None on failure.

Fails if argument is not a segment_t or there are no functions in that segment.

Return type list

static get_segments_with_functions()

Returns a list of segments with defined functions in it.

Returns Empty list or list of segment_t objects

Return type list

static populate_function_list()

Initializes FIRST's function list

This should be called to initialize the FIRST.function_list global variable, thus it should be called once IDA's auto analysis is complete to ensure it gets as many functions as possible.

Base case: User loads up sample in IDA for first time or IDB is opened in IDA with FIRST for the first time action: create new function list, save, monitor for changes

Complex case: User reopens an IDB that already has FIRST data in it action: extract function list from IDB, monitor for changes

9.15 FIRST :: MetadataServer

class first_plugin_ida.first.**FIRST**

class MetadataServer(*data*, *address=None*, *engine_info=None*)

Class to contain a FIRST match and its data.

FIRST Metadata container, it encapsulates the data received from the FIRST server.

Parameters

- **data** (`dict`) – Dictionary with the following key values set: name, prototype, creator, id, comment, rank
- **address** (`int`) – The VA associated with the function the instance refers to.
- **engine_info** (`dict`) – Dictionary with engine names mapping to the engine's description.

Raises FIRST.Error – If data is not a `dict` or does not have the required keys.

address

The virtual address associated with the function.

Type `int`

comment
The comment associated with the function.
Type `str`

creator
The handle of the annotation creator.
Type `str`

engine_info
The mapping from engine name to its description.
Type `dict`

id
The FIRST ID associated with this metadata.
Type `str`

name
The name of the function
Type `str`

prototype
The prototype of the function
Type `str`

rank
The number of unqiue applies of this metadata.
Type `int`

similarity
The percentage of similarity between this function and the original queried for function. This value can be very rough estimate depending on the engine.
Type `float`

9.16 FIRST :: MetadataShim

class `first_plugin_ida.first.`**FIRST**

class MetadataShim(*address=0*, *name=''*, *creator=None*)
Shim between interacting with various IDA components and FIRST.

FIRST Metadata Container provides thin shim for interacting with function and affecting IDA's UI. Changes made from FIRST are updated in the UI and IDA's IDB DB.

When creating a MetadataShim instance, at least the address should be provided to the constructor. However, it can be useful to create an empty object and populate if with data by calling its `from_db` method.

Parameters

- **address** (`int`, optional) – The VA of the function.
- **name** (`str`, optional) – The original name of the function. This should be used to set an original baseline for the function. The default name (sub_X, where X is the function start VA) can be overwritten if this is set.
- **creator** (`str`, optional) – The creator's handle

Examples

Creating MetadataShim instance from function.

```
>>> m1 = MetadataShim(address=0x401000)
```

Creating MetadataShim instance from function and setting original name.

```
>>> m2 = MetadataShim(address=0x401e40, name='memcpy')
```

Creating MetadataShim instace from a function with a creator.

```
>>> m3 = MetadataShim(address=0x401330, creator='demonduck#1337')
```

`address`
The virtual address associated with the function.
Type `int`

`apis`
The APIs called by the function.
Type `list`

`apply_metadata`(*data*)
Applies metadata to the function.

The metadata will be applied and become visable in IDA Pro. Updates sample's IDB DB with the new function annotations.
Parameters **`data`** (`MetadataServer`) – The metadata result from FIRST server.

`comment`
The repeatable comment associated with the function.

Returns only the first 1024 bytes of the comment. If a comment is longer than that, then it will be truncated to 1024. This mean data could be lost.
Returns The function's repeatable comment
Return type str
Type `str`

`created`
True if the annotations were created by user.
Type `bool`

`creator`
The handle of the annotation creator.
Type `str`

`from_db`(*data_str*)
Converts IDB DB data to MetadataShim object.
Parameters **`data_str`** (*str*) – JSON data in a string. JSON data keys required: author, changed, original_name, offset, id.

`has_changed`
True if function metadata has changed.
Type `bool`

`id`
The FIRST ID associated with the function.
Type `str`

is_lib
True if function is a library function.
Type `bool`

name
The name of the function
Type `str`

offset
The function offset from the start of the segment.
Type `int`

original_name
The orginal name of the function.

Unfortunately, this is a best guess. If the function has been detected as a library function by IDA then we use the current name since there is no way to get any of the previous names it might have had. If it is not a library function then the original name is sub_X, where X is the VA of the function.
Returns The original name of the function.
Return type str
Type `str`

prototype
The prototype of the function
Type `str`

segment
The start address of the function's segment.

Returns None if no segment can be retrieved
Type `int`

signature
The opcodes associated with the function.
Type `str`

snapshot()
Saves off current function annotations

Preserves the function name, comment, prototype and FIRST ID currently associated with the function. This will be used to compare with to detect future changes.

to_db()
Provides data structure for the IDB's DB.
Returns

FIRST information for the DB.
{ 'offset' : `int`,

'original_name' : `str`,

'author' : `str`,

'id' : `str`,

'changed' : `bool`

}
Return type dict

update_comment()
Updates IDB DB if comment has changed since last snapshot.

update_db(*changed*)
Updates the IDB DB with FIRST identifiers.

update_name()
Updates IDB DB if name has changed since last snapshot.

update_prototype()
Updates IDB DB if prototype has changed since last snapshot.

9.17 FIRST :: Model

class `first_plugin_ida.first.`**FIRST**

class Model

class Base(*header*, *data=None*, *parent=None*)
A QT QAbstractTableModel Implementation.

Parameters

- **header** (`list`) – The column values.
- **data** (`dict`) – Dictionary of values.
- **parent** (`QtCore.QObject`) – The parent object.

Overloads many class methods to provide the functionality FIRST required.

columnCount(*parent=<sphinx.ext.autodoc.importer._MockObject object>*)
The number of columns for the children of the given parent.
Parameters **parent** (`QtCore.QModelIndex`, optional) – Parent
Returns Number of columns
Return type int

data(*index*, *role=<sphinx.ext.autodoc.importer._MockObject object>*)
The data stored under the given role for the item referred to by the index.
Parameters
- **index** (`QtCore.QModelIndex`) – Index
- **role** (`Qt.ItemDataRole`) – Default `Qt.DisplayRole`

Returns data

headerData(*section*, *orientation*, *role=<sphinx.ext.autodoc.importer._MockObject object>*)
The data for the given role and section in the header with the specified orientation.
Parameters
- **section** (`int`) –
- **orientation** (`Qt.Orientation`) –
- **role** (`Qt.DisplayRole`) –

Returns data

raw_data(*i*)
Provides a way to get the raw data in the model.
Parameters **i** (`int`) – The data index to be retrieved.
Returns dict. The data held at the given index, otherwise None.

rowCount(*parent=<sphinx.ext.autodoc.importer._MockObject object>*)
The number of rows under the given parent.

When the parent is valid it means that rowCount is returning the number of children of parent.

Parameters **`parent`** (`QtCore.QModelIndex`, optional) – Parent

Returns Number of rows

Return type int

`class Check`(*data*, *parent=None*)

Expands on the Qt QStandardItemModel for Check operations.

`add_data`(*data*)

Provides a way to add more data to the model.

Parameters **`data`** (`dict`) – Data to be added to the model.

`data`(*index*, *role*)

The data stored under the given role for the item referred to by the index.

Parameters

- **`index`** (`QtCore.QModelIndex`) – Index
- **`role`** (`Qt.ItemDataRole`) – Default `Qt.DisplayRole`

Returns data

`get_selected_data`()

Returns a dictionary of data selected in the model.

`select_highest_ranked`(*flag*, *hidden=[]*)

Sets the highsest rank annotations as (de)selected.

Parameters

- **`flag`** (`bool`) – Where to select or deselect the highest. True to select highest, False to deselect highest.
- **`hidden`** (`list` or `int`) – Address that should be skipped.

`set_id_selected`(*data*)

Add or removes data associated with an ID to/from the selected ids array.

Parameters **`data`** (`list`) – The data to be (de)selected.

`unselect_group`(*data*)

Unselects a group of addresses at once.

Parameters **`data`** (`list` of `int`) – List of addresses.

`class TreeView`(*widget=None*, *depth=2*)

A QT QTreeView Implementation.

Parameters

- **`widget`** (`Qt.QObject`, optional) – The parent.
- **`depth`** (`int`, optional) – The depth of the tree.

`drawRow`(*painter*, *option*, *index*)

Draws the row in the tree view that contains the model item index, using the painter given. The option control how the item is displayed.

Parameters

- **`painter`** (`QtGui.QPainter`) – Painter
- **`option`** (`QtGui.QStyleOptionViewItem`) – Options
- **`index`** (`QtCore.QModelIndex`) – Index

`class Upload`(*header*, *data*, *parent=None*)

Expands on the Base QAbstractTableModel for Add operation.

Data held in this DataModel is sorted based on their offset within the IDB. A couple of additional functions are added to this model to provide more functionality to modify the selected underlying data.

data (*index*, *role*)
The data stored under the given role for the item referred to by the index.

Parameters

- **index** (`QtCore.QModelIndex`) – Index
- **role** (`Qt.ItemDataRole`) – Default `Qt.DisplayRole`

Returns data

filter_sub_functions (*flag*)
Filters out or restores any sub_* functions.

Parameters **flag** (`bool`) – Flag to filter out or restore sub_* functions

get_selected_data ()
Returns the list of data selected in the model.

select_all (*flag*)
Makes all visible functions selected or deselected.

Parameters **flag** (`bool`) – Flag to select or deselect all.

set_colors (*changed='66d9ef'*, *unchanged='d2d2d2'*, *default='ffffff'*, *select='a9c5ff'*)
Sets the colors associated with the various properties.

Parameters

- **changed** (`str`) – Change color, default: '66d9ef'
- **unchanged** (`str`) – Unchanged color, default: 'd2d2d2'
- **default** (`str`) – Default color, default: 'ffffff'
- **select** (`str`) – Selected color, default: 'a9c5ff'

set_row_selected (*row*)
Causes a row to be selected or deselected.

Parameters **row** (`int`) – The row index to be selected.

9.18 FIRST :: Server

data_callback_prototype (*thread*, *response*)
Function prototype for data_callback arguments.

Parameters

- **thread** (*threading.Thread*) – The thread associated with the server operation.
- **response** (*requests.models.response*) – The response from the server in JSON form.

complete_callback_prototype (*thread*, *data*)
Function prototype for complete_callback arguments. This function should call `FIRST.server.remove_operation` to ensure data is released once it is not needed.

Parameters

- **thread** (*threading.Thread*) – The thread associated with the server operation.
- **data** (*dict*) – All data received from the server.

class `first_plugin_ida.first.`**FIRST**

class Server (*config*, *h_md5*, *crc32*, *h_sha1=None*, *h_sha256=None*)
Encapsulate interacting with the FIRST server's REST API.

Note: Using functions `set_protocol`, `set_server`, and `set_port` do not update the configuration details, just the server instance represented with this class.

`urn`
URL format string.

Type `str`

`paths`
Mapping between operations and FIRST URI path format strings.

Type `dict`

`MAX_CHUNK`
The maximum number of entries sent to the server. Default: 20

Note: The FIRST server can set the max number of entries received. If this value is greater than the server's then the server will not perform the operation.

Type `int`

`Args`
config (`Configuration`): FIRST configuration information. h_md5 (`str`): The MD5 of the sample. crc32 (`int`): The CRC32 of the sample. h_sha1 (`str`, optional): The SHA1 of the sample. h_sha256 (`str`, optional): The SHA256 of the sample.

`MAX_CHUNK = 20`

`add`(*metadata*, *data_callback=None*, *complete_callback=None*)
Adds function metadata to FIRST.

This is a long operation, thus it has the option of providing a `data_callback` and `complete_callback` arguments. Those arguments are functions that will be called with the newly returned data and when the whole operation is complete, respectively. Both functions should follow the below their respective prototypes; `data_callback_prototype` and `complete_callback_prototype`.

:param metadata (`list` of `MetadataShim` or: `MetadataShim`): The metadata to be added to FIRST. :param data_callback: A function to call when data is receieved from the server. :type data_callback: *`data_callback_prototype`*, optional :param complete_callback: A function to call when the whole long operation completes. :type complete_callback: *`complete_callback_prototype`*, optional

Returns threading.Thread. The thread created for the operation.

`applied`(*metadata_id*)
Sets a FIRST annotation as applied to this sample.

This is a short operation and is a blocking call.

Parameters **`metadata_id`** (`str`) – The FIRST annotation ID.

Returns JSON data returned from the server. None on failure.

Return type dict

`checkin`(*action*)
Checks in with FIRST server to ensure annotations can be added.

This function must be called before any annotations are added to FIRST. This function allows the FIRST server to setup information about the sample, thereby allowing functions to be associated with the sample. This only needs to be called once and is attempted before the first user selected operation.

This operation is not done if the operation to be performed is to test the connection to the server.

Parameters **`action`** (`str`) – The FIRST operation to be performed

`created`(*data_callback=None*, *complete_callback=None*)

Retrieves FIRST annotations the user has created.

This is a long operation, thus it has the option of providing a `data_callback` and `complete_callback` arguments. Those arguments are functions that will be called with the newly returned data and when the whole operation is complete, respectively. Both functions should follow the below their respective prototypes; `data_callback_prototype` and `complete_callback_prototype`.

Parameters

- **`data_callback`** (*`data_callback_prototype`*, optional) – A function to call when data is receieved from the server.
- **`complete_callback`** (*`complete_callback_prototype`*, optional) – A function to call when the whole long operation completes.

Returns threading.Thread. The thread created for the operation.

`delete`(*metadata_id*)

Deletes a FIRST annotation created by the user.

This is a short operation and is a blocking call.

Parameters **`metadata_id`** (`str`) – The FIRST annotation ID.

Returns JSON data returned from the server. None on failure.

Return type dict

`get`(*metadata_ids*, *data_callback=None*, *complete_callback=None*)

Retrieves FIRST annotations the user has created.

This is a long operation, thus it has the option of providing a `data_callback` and `complete_callback` arguments. Those arguments are functions that will be called with the newly returned data and when the whole operation is complete, respectively. Both functions should follow the below their respective prototypes; `data_callback_prototype` and `complete_callback_prototype`.

Parameters

- **`metadata`** (`list` of `MetadataShim`) – The metadata to be retrieved from FIRST.
- **`data_callback`** (*`data_callback_prototype`*, optional) – A function to call when data is receieved from the server.
- **`complete_callback`** (*`complete_callback_prototype`*, optional) – A function to call when the whole long operation completes.

Returns threading.Thread. The thread created for the operation.

`history`(*metadata*)

Gets annotation history from FIRST.

This is a short operation and is a blocking call.

Parameters **metadata** (`MetadataShim` or `MetadataServer`) – The FIRST annotation the history is being requested.

Returns JSON data returned from server. None on failure.

Return type dict

paths = {'add': 'api/metadata/add/{0[api_key]}', 'applied': 'api/metadata/applie

remove_operation(*server_thread*)
Removes operation from server thread structure.

Parameters **server_thread** (`threading.Thread`) – The thread to remove.

scan(*metadata*, *data_callback=None*, *complete_callback=None*)
Queries FIRST for matches.

This is a long operation, thus it has the option of providing a `data_callback` and `complete_callback` arguments. Those arguments are functions that will be called with the newly returned data and when the whole operation is complete, respectively. Both functions should follow the below their respective prototypes; `data_callback_prototype` and `complete_callback_prototype`.

Parameters

- **metadata** (`list` of `MetadataShim`) – The metadata to be queried for matches in FIRST.
- **data_callback** (*`data_callback_prototype`*, optional) – A function to call when data is receieved from the server.
- **complete_callback** (*`complete_callback_prototype`*, optional) – A function to call when the whole long operation completes.

Returns threading.Thread. The thread created for the operation.

set_port(*port*)
Overrides the FIRST server port set in the configuration.

Parameters **port** (`int`) – The FIRST server port.

set_protocol(*protocol*)
Overrides the FIRST server protocol set in the configuration.

Parameters **protocol** (`int`) – The FIRST server protocol.

set_server(*server*)
Overrides the FIRST server set in the configuration.

Parameters **port** (`int`) – The FIRST server.

stop_operation(*server_thread*)
Signals a server thread to stop its work.

Parameters **server_thread** (`threading.Thread`) – The thread to stop.

test_connection()
Interacts with server to see if there is a valid connection.

This is a short operation and is a blocking call.

Returns

True if connection can be made and FIRST returns a success message. False otherwise.

Return type bool

`to_json`(*response*)
Converts Requests' response object to json.

Parameters **`response`** (`requests.models.Response`) – A request response.

Returns JSON data or empty dictionary.

Return type dict

`unapplied`(*metadata_id*)
Sets a FIRST annotation as unapplied to this sample.

This is a short operation and is a blocking call.

Parameters **`metadata_id`** (`str`) – The FIRST annotation ID.

Returns JSON data returned from the server. None on failure.

Return type dict

`urn = '{0.protocol}://{0.server}:{0.port}/{1}'`

Index