
x64dbg Documentation

Release 0.1

x64dbg

Jun 13, 2019

Contents

1	Suggested reads	1
1.1	What is x64dbg?	1
1.2	Introduction	1
1.3	GUI manual	14
1.4	Commands	29
1.5	Developers	120
1.6	Licenses	255
2	Indices and tables	271

Suggested reads

If you came here because someone told you to read the manual, start by reading **all sections** of the *introduction*.

Contents:

1.1 What is x64dbg?

This is a x64/x32 debugger that is currently in active development.

The debugger (currently) has three parts:

- DBG
- GUI
- Bridge

DBG is the debugging part of the debugger. It handles debugging (using TitanEngine) and will provide data for the GUI.

GUI is the graphical part of the debugger. It is built on top of Qt and it provides the user interaction.

Bridge is the communication library for the DBG and GUI part (and maybe in the future more parts). The bridge can be used to work on new features, without having to update the code of the other parts.

1.2 Introduction

This section explains the basics of x64dbg. **Make sure to fully read this!**

Contents:

1.2.1 Features

This program is currently under active development. It supports many basic and advanced features to ease debugging on Windows.

Basic features

- Full-featured debugging of DLL and EXE files ([TitanEngine Community Edition](#))
- 32-bit and 64-bit Windows support from Windows XP to Windows 10
- Built-in assembler ([XEDParse/Keystone/asmjit](#))
- Fast disassembler ([Capstone](#))
- C-like expression parser
- Logging
- Notes
- Memory map view
- Modules and symbols view
- Source code view
- Thread view
- Content-sensitive register view
- Call stack view
- SEH view
- Handles, privileges and TCP connections enumeration.
- Multi-datatype memory dump
- Dynamic stack view
- Executable patching
- Dynamically recognize modules and strings
- User database ([JSON](#)) for comments, labels, bookmarks, etc.
- Basic debug symbol (PDB) support
- Extendable, debuggable scripting language for automation
- Plugin support with growing API
- Basic anti-debugging

Advanced features

- Decompiler ([snowman](#))
- Import reconstructor integrated ([Scylla](#))
- Analysis
- Conditional breakpoints and tracing with great flexibility
- Collect data while tracing

GUI features

- Intuitive and familiar, yet new user interface
- IDA-like sidebar with jump arrows
- IDA-like instruction token highlighter (highlight registers, commands, etc.)
- Fully customizable color scheme and short-cut keys
- Control flow graph
- Integrated mnemonic and registers help
- Code folding
- Easy integration with custom tools (favourites menu)

1.2.2 Input

When using x64dbg you can often use various things as input.

Commands

Commands have the following format:

```
command arg1, arg2, argN
```

Variables

Variables optionally start with a \$ and can only store one DWORD (QWORD on x64).

Registers

All registers (of all sizes) can be used as variables.

Remarks

- The variable names for most registers are the same as the names for them, except for the following registers:
- **x87 Control Word Flag:** The flags for this register is named like this: `_x87CW_UM`
- In addition to the registers in the architecture, x64dbg provides the following registers: CAX , CBX , CCX , CDX , CSP , CBP , CSI , CDI , CIP. These registers are mapped to 32-bit registers on 32-bit platform, and to 64-bit registers on 64-bit platform. For example, CIP is EIP on 32-bit platform, and is RIP on 64-bit platform. This feature is intended to support architecture-independent code.

Memory locations

You can read/write from/to a memory location by using one of the following expressions:

- `[addr]` read a DWORD/QWORD from `addr`.
- `n:[addr]` read `n` bytes from `addr`.

- `seg: [addr]` read a DWORD/QWORD from a segment at `addr`.
- `byte: [addr]` read a BYTE from `addr`.
- `word: [addr]` read a WORD from `addr`.
- `dword: [addr]` read a DWORD from `addr`.
- `qword: [addr]` read a QWORD from `addr` (x64 only).

Remarks

- `n` is the amount of bytes to read, this can be anything smaller than 4 on x32 and smaller than 8 on x64 when specified, otherwise there will be an error.
- `seg` can be `gs`, `es`, `cs`, `fs`, `ds`, `ss`. Only `fs` and `gs` have an effect.

Flags

Debug flags (interpreted as integer) can be used as input. Flags are prefixed with an `_` followed by the flag name. Valid flags are: `_cf`, `_pf`, `_af`, `_zf`, `_sf`, `_tf`, `_if`, `_df`, `_of`, `_rf`, `_vm`, `_ac`, `_vif`, `_vip` and `_id`.

Numbers

All numbers are interpreted as hex by default! If you want to be sure, you can `x` or `0x` as a prefix. Decimal numbers can be used by prefixing the number with a dot: `.123=7B`.

Expressions

See the *expressions* section for more information.

Labels/Symbols

User-defined labels and symbols are a valid expressions (they resolve to the address of said label/symbol).

Module Data

DLL exports

Type `GetProcAddress` and it will automatically be resolved to the actual address of the function. To explicitly define from which module to load the API, use: `[module].dll:[api]` or `[module]:[api]`. In a similar way you can resolve ordinals, try `[module]:[ordinal]`. Another macro allows you to get the loaded base of a module. When `[module]` is an empty string `:GetProcAddress` for example, the module that is currently selected in the CPU will be used.

Loaded module bases

If you want to access the loaded module base, you can write: `[module]:0`, `[module]:base`, `[module]:imagebase` or `[module]:header`.

RVA/File offset

If you want to access a module RVA you can either write `[module]:0+[rva]` or you can write `[module]:${rva}`. If you want to convert a file offset to a VA you can use `[module]:#[offset]`. When `[module]` is an empty string `:0` for example, the module that is currently selected in the CPU will be used.

Module entry points

To access a module entry point you can write `[module]:entry`, `[module]:oep` or `[module]:ep`. Notice that when there are exports with the names `entry`, `oep` or `ep` the address of these will be returned instead.

Remarks

Instead of the `:` delimiter you can also use a `.` If you need to query module information such as `[module]:imagebase` or `[module]:entry` you are advised to use a `?` as delimiter instead: `[module]?entry`. The `?` delimiter does checking for named exports later, so it will still work when there is an export called `entry` in the module.

Last words

Input for arguments can always be done in any of the above forms, except if stated otherwise.

1.2.3 Expressions

The debugger allows usage of basic expressions. Just type an expression in the command window and the result will be displayed in the console. Apart from calculations, it allows quick variable changes using a C-like syntax.

Input

The basic input (numbers/variables) can be used as constants in expressions, see *Input* for more information.

Operators

You can use the following operators in your expression. They are processed in the following order:

1. *parentheses/brackets*: `(1+2)`, `[1+6]` have priority over other operations.
2. *unary minus/binary not/logical not*: `-1` (negative 1), `~1` (binary not of 1), `!0` (logical not of 0).
3. *multiplication/division*: `2*3` (regular multiplication), `2`3` (gets high part of the multiplication), `6/3` (regular division), `5%3` (modulo/remainder of the division).
4. *addition/subtraction*: `1+3` (addition), `5-2` (subtraction).
5. *left/right shift/rotate*: `1<<2` (shift left, `shl` for unsigned, `sal` for signed), `10>>1` (shift right, `shr` for unsigned, `sar` for signed), `1<<<2` (rotate left), `1>>>2` (rotate right).
6. *smaller (equal)/bigger (equal)*: `4<10`, `3>6`, `1<=2`, `6>=7` (resolves to 1 if true, 0 if false).
7. *equal/not equal*: `1==1`, `2!=6` (resolves to 1 if true, 0 if false).
8. *binary and*: `12&2` (regular binary and).

9. *binary xor*: $2 \wedge 1$ (regular binary xor).
10. *binary or*: $2 | 8$ (regular binary or).
11. *logical and*: $0 \& \& 3$ (resolves to 1 if true, 0 if false).
12. *logical or*: $0 | | 3$ (resolves to 1 if true, 0 if false).
13. *logical implication*: $0 \rightarrow 1$ (resolved to 1 if true, 0 if false).

Quick-Assigning

Changing memory, a variable, register or flag can be easily done using a C-like syntax:

- $a ? = b$ where $?$ can be any non-logical operator. a can be any register, flag, variable or memory location. b can be anything that is recognized as an expression.
- $a ++ / a --$ where a can be any register, flag, variable or memory location.

Functions

You can use functions in expressions. See [Expression Functions](#) for the documentation of these functions.

1.2.4 Expression Functions

You may use functions in an expression. The following functions are defined by the debugger:

GUI Interaction

- `disasm.sel()`/`dis.sel()` : Get the selected address in the disassembly view.
- `dump.sel()` : Get the selected address in the dump view.
- `stack.sel()` : Get the selected address in the stack view.

Source

- `src.disp(addr)` : Get displacement of `addr` relative to last source line.
- `src.line(addr)` : Get the source line number of `addr`.

Modules

- `mod.party(addr)` : Get the party of the module `addr`. 0 is user module, 1 is system module.
- `mod.base(addr)` : Get the base address of the module `addr`.
- `mod.size(addr)` : Get the size of the module `addr`.
- `mod.hash(addr)` : Get the hash of the module `addr`.
- `mod.entry(addr)` : Get the entry address of the module `addr`.
- `mod.system(addr)` : True if the module at `addr` is a system module. No module is a user module.
- `mod.user(addr)` : True if the module at `addr` is a user module. No module is a user module.
- `mod.main()` : Returns the base of the main module (debuggee). If this is a DLL it will return 0 until loaded.

- `mod.rva(addr)` : Get the RVA of `addr`. If `addr` is not inside a module it will return 0.
- `mod.offset(addr)` : Get the file offset of `addr`. If `addr` is not inside a module it will return 0.
- `mod.isexport(addr)` : True if `addr` is an exported function from a module.

Process Information

- `peb()` : Get PEB address.
- `teb()` : Get TEB address.
- `tid()` : Get the current thread ID.

General Purpose

- `bswap(value)` : Byte-swap value.
- `ternary(condition, val1, val2)` : If `condition` is nonzero, return `val1`, otherwise return `val2`.
- `GetTickCount()` : Tick count of x64dbg.

Memory

- `mem.valid(addr)` : True if `addr` is a valid memory address.
- `mem.base(addr)` : Returns the base of the memory page of `addr` (can change depending on your memory map mode).
- `mem.size(addr)` : Returns the size of the memory page of `addr` (can change depending on your memory map mode).
- `mem.iscode(addr)` : True if `addr` is a page that is executable.
- `mem.decodepointer(ptr)` : Equivalent to calling the `DecodePointer` API on `ptr`, only works on Vista+.

Disassembly

- `dis.len(addr)` : Get the length of the instruction at `addr`.
- `dis.iscond(addr)` : True if the instruction at `addr` is a conditional branch.
- `dis.isbranch(addr)` : True if the instruction at `addr` is a branch (`jmp/call`).
- `dis.isret(addr)` : True if the instruction at `addr` is a `ret`.
- `dis.iscall(addr)` : True if the instruction at `addr` is a `call`.
- `dis.ismem(addr)` : True if the instruction at `addr` has a memory operand.
- `dis.isnop(addr)` : True if the instruction at `addr` is equivalent to a `NOP`.
- `dis.isunusual(addr)` : True if the instruction at `addr` is unusual.
- `dis.branchdest(addr)` : Branch destination of the instruction at `addr` (what it follows if you press enter on it).
- `dis.branchexec(addr)` : True if the branch at `addr` is going to execute.
- `dis.imm(addr)` : Immediate value of the instruction at `addr`.

- `dis.brtrue(addr)` : Branch destination of the instruction at `addr`.
- `dis.brfalse(addr)` : Address of the next instruction if the instruction at `addr` is a conditional branch.
- `dis.next(addr)` : Address of the next instruction from `addr`.
- `dis.prev(addr)` : Address of the previous instruction from `addr`.
- `dis.iscallsystem(addr)` : True if the instruction at `addr` goes to a system module.

Trace record

- `tr.enabled(addr)` : True if the trace record is enabled at `addr`.
- `tr.hitcount(addr)` : Number of hits on the trace record at `addr`.
- `tr.runtraceenabled()` : True if run trace is enabled.

Byte/Word/Dword/Qword/Ptr

- `ReadByte, Byte, byte(addr)` : Read a byte from `addr` and return the value.
- `ReadWord, Word, word(addr)` : Read a word (2 bytes) from `addr` and return the value.
- `ReadDword, Dword, dword(addr)` : Read a dword (4 bytes) from `addr` and return the value.
- `ReadQword, Qword, qword(addr)` : Read a qword (8 bytes) from `addr` and return the value (only available on x64).
- `ReadPtr, ReadPointer, ptr, Pointer, pointer(addr)` : Read a pointer (4/8 bytes) from `addr` and return the value.

Functions

- `func.start()` : Start of the function `addr` is part of, zero otherwise.
- `func.end()` : End of the function `addr` is part of, zero otherwise.

References

- `ref.count()` : Number of entries in the current reference view.
- `ref.addr(index)` : Get the address of the reference at `index`. Zero on failure.

Arguments

This assumes the return address is on the stack (eg you are inside the function).

- `arg.get(index)` : Gets the argument at `index` (zero-based).
- `arg.set(index, value)` : Sets the argument at `index` (zero-based) to `value`.

Exceptions

This is a set of functions to get information about the last exception. They can be used for exceptions breakpoints to construct more advanced conditions.

- `ex.firstchance()` : Whether the last exception was a first chance exception.
- `ex.addr()` : Last exception address.
- `ex.code()` : Last exception code.
- `ex.flags()` : Last exception flags.
- `ex.infocount()` : Last exception information count (number of parameters).
- `ex.info(index)` : Last exception information, zero if index is out of bounds.

Plugins

Plugins can register their own expression functions. See the plugin documentation for more details.

1.2.5 Variables

This program supports variables. There are three types of variables:

- **USER:** Variables created by the user using the `var` command. These variables have no access restrictions.
- **SYSTEM:** Variables created by the system, that can be read and written, but cannot be deleted.
- **READONLY:** Variables created by the system, that can be read, but not written or deleted.

Reserved Variables

There are a few reserved variables:

- `$res/$result`: General result variable.
- `$resN/$resultN`: Optional other result variables (N= 1-4).
- `$pid`: Process ID of the debugged executable.
- `$hp/$hProcess`: Debugged executable handle.
- `$lastalloc`: Last result of the `alloc` command.
- `$breakpointcondition` : Controls the pause behaviour in the conditional breakpoint command.
- `$breakpointcounter` : The hit counter of the breakpoint, set before the condition of the conditional breakpoint is evaluated.
- `$breakpointlogcondition` : The log condition of the conditional breakpoint. It cannot be used to control the logging behaviour.

1.2.6 Conditional Breakpoints

This section describes the conditional breakpoint capability in x64dbg.

Operations overview

When a breakpoint is hit, x64dbg will do the following things:

- If the breakpoint is an exception breakpoint, set the system variable `$breakpointexceptionaddress` to the exception address;
- Increment the *hit counter*;
- Set the system variable `$breakpointcounter` to the value of *hit counter*;
- If *break condition* is set, evaluate the *expression* (defaults to 1);
- If *fast resume* is set and *break condition* evaluated to 0:
 - Resume execution of the debuggee (skip the next steps). This will also skip executing plugin callbacks and GUI updates.
- If *log condition* is set, evaluate the *expression* (defaults to 1);
- If *command condition* is set, evaluate the *expression* (defaults to *break condition*);
- If *break condition* evaluated to 1 (or any value other than '0'):
 - Print the standard log message; (if *the breakpoint is set to be silent*, standard log message is suppressed.)
 - Execute plugin callbacks.
- If *log text* is set and *log condition* evaluated to 1 (or any value other than '0'):
 - Format and print the *log text* (see *String Formatting*).
- If *command text* is set and *command condition* evaluated to 1:
 - Set the system variable `$breakpointcondition` to the *break condition*;
 - Set the system variable `$breakpointlogcondition` to the *log condition*;
 - Execute the command in *command text*;
 - The *break condition* will be set to the value of `$breakpointcondition`. So if you modify this system variable in the script, you will be able to control whether the debuggee would break.
- If *break condition* evaluated to 1 (or any value other than '0'):
 - Break the debuggee and wait for the user to resume.

Hit counter

A hit counter records how many times a breakpoint has been reached. It will be incremented unconditionally, even if fast resume is enabled on this breakpoint. It may be viewed at breakpoint view and reset with *ResetBreakpointHit-Count*.

Logging

The log can be formatted by x64dbg to log the current state of the program. See *formatting* on how to format the log string.

Notes

You can set a conditional breakpoint with GUI by setting a software breakpoint(key F2) first, then right-click on the instruction and select “Edit breakpoint” command from the context menu. Fill in the conditional expression and/or other information as necessary, then confirm and close the dialog.

You should not use commands that can change the running state of the debuggee (such as `run`) inside the breakpoint command, because these commands are unstable when used here. You can use *break condition*, *command condition* or `$breakpointcondition` instead.

If you don’t know where the condition will become true, try *conditional tracing* instead!

Examples

A conditional breakpoint which never breaks

break condition: 0

A conditional breakpoint which breaks only if EAX and ECX both equal to 1

break condition: EAX==1 && ECX==1

A conditional breakpoint which breaks only if EAX is a valid address

break condition: mem.valid(EAX)

A conditional breakpoint which breaks on the third hit

break condition: \$breakpointcounter==3 or (\$breakpointcounter%3)==0

A conditional breakpoint which breaks only if executed by the thread 1C0

break condition: tid()==1C0

See also

- [Conditional Breakpoint Control](#)
- [Expressions](#)
- [Expression Functions](#)
- [String Formatting](#)

1.2.7 Conditional Tracing

This section describes the *conditional tracing* capability in x64dbg.

Operations overview

When a trace step is hit, x64dbg will do the following things:

- Increment the *trace counter*;
- Set the system variable `$tracecounter` to the value of *trace counter*;
- If *break condition* is set, evaluate the *expression* (defaults to 0);
- Execute plugin callbacks (allowing plugins to change the *break condition*);

- If *log condition* is set, evaluate the *expression* (defaults to 1);
- If *command condition* is set, evaluate the *expression* (defaults to *break condition*);
- If *switch condition* is set, evaluate the *expression* (defaults to 0)
- If *log text* is set and *log condition* evaluated to 1:
 - Format and print the *log text* (see *String Formatting*). To redirect the log to a file use *TraceSetLogFile*.
- If *command text* is set and *command condition* evaluated to 1:
 - Set the system variable `$tracecondition` to the *break condition*;
 - Set the system variable `$tracelogcondition` to the *log condition*;
 - Set the system variable `$traceswitchcondition` to the *switch condition*;
 - Execute the command in *command text*;
 - The *break condition* will be set to the value of `$tracecondition`. So if you modify this system variable in the script, you will be able to control whether the debuggee would break.
 - The *switch condition* will be set to the value of `$traceswitchcondition`. So if you modify this system variable in the script, you will be able to control whether the step type is switched.
- If *break condition* evaluated to 1:
 - Print the standard log message;
 - Break the debuggee and wait for the user to resume.
- If *switch condition* evaluated to 1:
 - Switch (invert) the step type. If you are tracing *in* it will switch to *out* (and the other way around). This allows you to for example not trace into system module calls with the condition `mod.party(dis.branchdest(cip)) == 1` or not trace into certain calls.

Logging

The log can be formatted by x64dbg to log the current state of the program. See *formatting* on how to format the log string. If you are looking for logging the address and disassembly of all instructions traced you can use `{p:cip}` `{i:cip}`. To redirect the log to a file use *TraceSetLogFile*.

Trace record

If you use one of the trace record-based tracing options, the initial evaluation of *break condition* includes the type of trace record tracing that you specified. The normal *break condition* can be used to break before the trace record condition is satisfied. If you want to include trace record in your condition for full control, you can use the *expression functions*.

Notes

You can start a conditional tracing by “Trace over until condition”/“Trace into until condition” commands in the *Debug menu*.

You should not use commands that can change the running state of the debuggee (such as `run`) inside the breakpoint command, because these commands are unstable when used here. You can use *break condition*, *command condition* or `$tracecondition` instead.

See also

- *Tracing*
- *Expressions*
- *Expression Functions*
- *String Formatting*

1.2.8 String Formatting

This section explains the simple string formatter built into x64dbg.

The basic syntax is `{?:expression}` where `?` is the optional type of the expression. The default type is `x`. To output `{` or `}` in the result, escape them as `{{` or `}}`.

Types

- `d` signed **d**ecimal: `-3`
- `u` **u**nsigned decimal: `57329171`
- `p` zero prefixed **p**ointer: `0000000410007683`
- `s` **s**tring pointer: `this is a string`
- `x` **x**hex: `3C28A`
- `a` **a**ddress info: `00401010 <module.EntryPoint>`
- `i` **i**nstruction text: `jmp 0x77ac3c87`

Complex Type

`{mem;size@address}` will print the `size` bytes starting at `address` in hex.

`{winerror@code}` will print the name of windows error code (returned with `GetLastError()`) and the description of it (with `FormatMessage`). It is similar to `ErrLookup` utility.

`{ntstatus@code}` will print the name of `NTSTATUS` error code and the description of it (with `FormatMessage`).

`{ascii[;length}@address}` will print the ASCII string at `address` with an optional `length` (in bytes).

`{ansi[;length}@address}` will print the ANSI string at `address` with an optional `length` (in bytes).

`{utf8[;length}@address}` will print the UTF-8 string at `address` with an optional `length` (in bytes).

`{utf16[;length}@address}` will print the UTF-16 string at `address` with an optional `length` (in words).

`{disasm@address}` will print the disassembly at `address` (equivalent to `{i:address}`).

`{modname@address}` will print the name of the module at `address`.

Examples

- `rax: {rax}` formats to `rax: 4C76`
- `password: {s:4*ecx+0x402000}` formats to `password: L"s3cret"`
- `log {x:bswap(rax)}` if `rax=0000000078D333E0` the result will be `E033D37800000000` because of `bswap` fun which reverse the hex value

Plugins

Plugins can use `_plugin_registerformatfunction` to register custom string formatting functions. The syntax is `{type;arg1;arg2;argN@expression}` where `type` is the name of the registered function, `argN` is any string (these are passed to the formatting function as arguments) and `expression` is any valid expression.

1.2.9 Inability

This section gives a list of features currently not supported in x64dbg. You are always welcome to contribute to x64dbg to help fixing them.

- Fine-grained memory breakpoint. Unlike other debuggers, memory breakpoint is supported only on a whole memory page, but not on a subrange of the memory page.
- Search for non-English strings. Searching for non-English strings via the built-in strings search may not be able to find all the non-English strings.
- Other pending issues at issues.x64dbg.com.

1.3 GUI manual

This section describes the usage of the program's graphical interface.

Contents:

1.3.1 Menus

This section describes menus in the GUI.

Content:

File

The file menu contains the following entries:

Open

The **Open** action lets you open an executable to debug it. The file can be an EXE file or a DLL file.

The command for this action is `InitDebug/initdbg/init`.

Recent Files

The **Recent Files** submenu contains several entries that you previously debugged. It does not include any file that cannot be debugged by the program.

The entries for this submenu can be found in the `Recent Files` section of the config INI file. You can edit that file to remove entries.

Attach

Attach lets you attach to a running process. It will show a dialog listing the running processes, and allow you to choose one to attach. Currently you can only attach to an executable that is of the same architecture as the program. (eg, you cannot attach to a 64-bit process with x32dbg)

If you are debugging an executable, attaching to another process will terminate the previous debuggee.

The command for this action is *AttachDebugger/attach*.

Detach

This action will detach the debugger from the debuggee, allowing the debuggee to run without being controlled by the debugger. You cannot execute this action when you are not debugging.

The command for this action is *DetachDebugger/detach*.

Import database

This allows you to import a database.

The relevant command for this action is *dbload/loaddb*.

Export database

This allows you to export an uncompressed database.

The relevant command for this action is *dbsave/savedb*.

Patch file

Opens the patch dialog. You can view your patches and apply the patch to a file in the dialog.

Restart as Admin

It will restart x64dbg and the current debuggee with administrator privilege.

Exit

Terminate the debugger. If any process is being debugged by this program, they are going to be terminated as well.

Debug

This menu contains the following actions. You cannot use any of these menu items except “**R**estart” and “**C**ommand” when you are not debugging.

Run

Execute the command *run/go/r/g*.

Run (Pass exceptions)

Execute the command *erun/ego/er/eg*.

Run (swallow exception)

Execute the command *serun/sego*.

Run until selection

Place a single-shoot software breakpoint at the selected instruction, and then execute the command *run/go/r/g* to run the debuggee.

Run until expression

Enter an address. The debugger will then place a software breakpoint at that address, and then execute the command *run/go/r/g* to run the debuggee.

Pause

Try to pause the debuggee when it is running, or try to stop animation. The command for this action is *pause*.

Restart

Execute the command *InitDebug/initdbg/init* with the most recent used file.

Close

Execute the command *StopDebug/stop/dbgstop*.

Change Command Line

Display the current command line arguments of the debuggee in a dialog, and allow you to change it. The command line arguments will be saved in the database for later use.

Step Into

Execute the command *StepInto/sti*.

Step Into (pass exceptions)

Execute the command *eStepInto/esti*.

Step Into (swallow exception)

Execute the command *seStepInto/sesti*.

Step Into (source)

Step into, until another source line is reached. The command for this menu entry is `TraceIntoConditional src.line(cip) && !src.disp(cip)`.

Trace into until condition

Enter an expression. The debugger will execute the command *TraceIntoConditional/ticnd*. Also see *Expressions* for the legal expression format.

Animate into

Execute *StepInto/sti* command at a steady frequency automatically.

Step Over

Execute the command *StepOver/step/sto/st*.

Step Over (pass exceptions)

Execute the command *eStepOver/estep/esto/est*.

Step Over (swallow exception)

Execute the command *seStepOver/sestep/sesto/sest*.

Step Over (source)

Step over, until another source line is reached. The command for this menu entry is `TraceOverConditional src.line(cip) && !src.disp(cip)`.

Run to User Code

Execute the command *RunToUserCode/rtu*.

Trace over until condition

Enter an expression. The debugger will execute the command *TraceOverConditional/tocnd*. Also see *Expressions* for the legal expression format.

Animate over

Execute *StepOver/step/sto/st* command at a steady frequency automatically.

Execute till return

Step over the instructions, until the current instruction pointed to by EIP or RIP is `ret` instruction.

The command for this action is *StepOut/rtr*.

Execute till return (pass exceptions)

Step over the instructions, until the current instruction pointed to by EIP or RIP is `ret` instruction. This instruction passes first-chance exceptions to the debuggee but swallows second-chance exceptions.

The command for this action is *eStepOut/ertr*.

Skip next instruction

Execute the command *skip*.

Animate command

Pop up a dialog to enter a command, and execute that command at a steady frequency.

Trace Record

Undo last instruction

Execute the command *InstrUndo*.

Command

Set focus to the command box at the bottom of the window, so that you can enter a command to execute.

Hide debugger (PEB)

Execute the command *HideDebugger/dbh/hide*.

Plugins

This menu includes all the available plugin menus. When you install a plugin, it may register a menu here. You can refer to the documentation of the plugin for more information.

Scylla

Launch scylla.

Favourites

This menu is customizable. When you click “Manage Favourite Tools” menu entry, a dialog will appear. You can add your custom tools to the menu, and also assign hotkeys to them. By default the path of the tool or script will appear in the menu, but if you set description of it, the description will appear in the menu instead.

- If you add %PID% in the command line of a tool, it will be replaced with the (decimal) PID of the debuggee (or 0 if not debugging).
- If you add %DEBUGGEE% it will add the (unquoted) full path of the debuggee.
- If you add %MODULE% it will add the (unquoted) full path of the module currently in the disassembly.
- If you add %-????-% it will perform *String Formatting* on whatever you put in place of ?????. Example: %-{cip}-% will be replaced with the hex value of cip.

Currently, three types of entries may be inserted into this menu: Tool, Script and Command.

See also:

You can also add entries to this menu via the following commands:

AddFavouriteCommand

AddFavouriteTool

AddFavouriteToolShortcut/SetFavouriteToolShortcut

Options

The options menu contains the following entries:

Preferences

Show the *Settings* dialog. You can modify various settings in the dialog.

Appearance

Show the **Appearance** dialog. You can customize the color scheme or font in the dialog.

Shortcuts

Show the **Shortcuts** dialog. You can customize the shortcut keys for most of the operations.

Customize Menus

Show the “Customize menus” dialog. You can click on the nodes to expand the corresponding menu and check or uncheck the menu items. Checked item will appear in “More commands” section of the menu, to shorten the menu displayed. You can check all menu entries that you don’t use.

Topmost

Keep the main window above other windows(or stop staying topmost).

Reload style.css

Reload `style.css` file. If this file is present, new color scheme specified in this file will be applied.

Set Initialization Script

Set a initialization script globally or for the debuggee. If a global initialization script is specified, it will be executed when the program is at the system breakpoint or the attach breakpoint for every debuggee. If a per-debuggee initialization script is specified, it will be executed after the global initialization script finishes. You can clear the script setting by clearing the file path and click “OK” in the browse dialog.

Import settings

Import settings from another configuration file. The corresponding entries in the configuration file will override the current configuration, but the missing entries will stay unmodified.

Languages

Allow the user to choose a language for the program. “*American English - United States*” is the native language for the program.

Help

This section describes the help menu.

Calculator

Show a calculator that can perform expression evaluation, hex to decimal conversion and more.

Check for updates

Connect to the server to query if an updated version of this software is available.

Donate

Donate to the developer team.

Blog

Visit the official blog.

Report Bug

Report a bug about this software.

Manual

The article you are reading.

FAQ

View the solution of some frequently asked question.

About

Information about this software.

Generate crash dump

Generate an exception to generate a crash dump. This may help if the software encounters a deadlock. You can submit this crash dump to the developer team to help them fix the bug.

1.3.2 Views

This section describes the usage of the views in the user interface.

Content:

CPU

This view is the main view. It includes the registers view, the disassembly view, the dump view and the watch view, the stack view, and the info box.

Graph

Graph view contains the control flow graph. When you use *graph* command or context menu in the disassembly view, it will show the control flow graph here.

There are two modes to show the control flow graph: Normal mode and overview mode.

In overview mode, the program will draw all the control flow graph within the window area, but not output the disassembly. When the first instruction is traced when trace record is enabled on this memory page, the whole basic block will be shown in a different color (Default is green).

Log

This view includes all the log messages. When an address is output in the log message, it will be shown as a hyperlink. You can click on it to follow it in disassembly or in dump, depending on its memory access rights.

The log view has the following context menu:

Clear

Clear the log view.

Select All

Select all the text of the log.

Copy

Copy the selected text.

Save

Save the log to a text file.

Disable Logging/Enable Logging

Disables or enables log output. When the logging is disabled, no more messages will output to the log.

Auto scroll

Enables or disables auto-scrolling. When enabled, the log view will scroll to the bottom as new log messages are coming in.

Redirect Log

Redirect log message to a file. If you enable this feature, all the messages will be saved to a UTF-16 encoded text file. The message will be saved to the text file no matter whether logging is enabled.

Disable Log Redirection

This menu entry is valid only if the log redirection is active. It stops log redirection.

Notes

Notes view have two text fields to edit, one globally and one for the debuggee. Any text entered here will be saved, and will be restored in future debugging sessions, so the user can make notes conveniently. Global notes will be stored in “notes.txt” under the working directory. Debuggee notes will be stored in the debug database. You cannot edit per-debuggee notes while not debugging.

Call Stack

Call stack view displays the call stack of the current thread. It has 6 columns.

Address is the base address of the stack frame.

To is the address of the code that is going to return to.

From is the probable address of the routine that is going to return.

Size is the size of the call stack frame, in bytes.

Comment is a brief description of the call stack frame.

Party describes whether the procedure that is going to return to, is a user module or a system module.

When **Show suspected call stack frame** option in the context menu in call stack view is active, it will search through the entire stack for possible return addresses. When it is inactive, it will use standard stack walking algorithm to get the call stack. It will typically get more results when **Show suspected call stack frame** option is active, but some of which may not be actual call stack frames.

Trace

Trace view is a view in which you can see history of stepped instructions. This lets you examine the details of each instruction stepped when you are stepping manually or *tracing* automatically, or view a trace history previously saved. This functionality must be enabled explicitly from trace view or *CPU view*. It features saving all the instructions, registers and memory accesses during a trace.

Start Run Trace

To enable trace logging into trace view, you first enable it via “**Start Run Trace**” menu. It will pop up a dialog allowing you to save the recorded instructions into a file. The default location of this file is in the database directory.

Once started, every instruction you stepped or traced will appear immediately in Trace view. Instructions executed while the debuggee is running or stepping over will not appear here.

Stop Run Trace

This menu can stop recording instructions.

Close

Close current trace file and clear the trace view.

Open

Open a trace file to view the content of it. It can be used when not debugging, but it is recommended that you debug the corresponding debuggee when viewing a trace, as it will be able to render the instructions with labels from the database of the debuggee.

Recent files

Open a recent file to view the content of it.

1.3.3 Settings

This section describes the settings dialog and each setting in the dialog. All the settings with “*” mark do not take effect until next start.

Contents:

Events

This page contains a list of debug events. You can specify whether the program should pause when the debug events happen.

System Breakpoint

This event happens when the process is being initialized but have not begun to execute user code yet.

TLS Callbacks

Set a single-shoot breakpoint on the TLS callbacks when a module is loaded to pause at the TLS callback.

Entry Breakpoint

Set a single-shoot breakpoint on the entry of the EXE module to pause at the entry point.

DLL Entry

Set a single-shoot breakpoint on the entry of the DLL module to pause at the entry point.

Attach Breakpoint

This event happens when the process is successfully attached.

Thread Entry

Set a single-shoot breakpoint on the entry of the thread when a thread is about to run.

DLL Load

Pause when a DLL is mapped to the address space.

DLL Unload

Pause when a DLL is unmapped from the address space.

Thread Start

Pause when a new thread is about to run.

Thread End

Pause when a thread has exited.

Debug Strings

Pause when a debug string is emitted by the debuggee.

Exceptions

This page contains a list of ignored exceptions. When a listed first-chance exception occurs, x64dbg will pass that exception to the debuggee without pausing.

Add Range

You can specify a range of exception codes to ignore. The input is hexadecimal.

Delete Range

Delete an ignored exception range, so that it will not be ignored.

Add Last

Add the last exception to the list.

GUI

This section describes various settings in the GUI tab.

Show FPU registers as little endian

Some FPU registers, especially SSE and AVX registers, are usually used to perform parallel computation. Using little endian helps to correspond floating point numbers to their index in memory arrays. However, big endian representation are more familiar to most users. This option can set whether FPU registers are shown as little endian or as big endian. You also edit the FPU registers in the endianness set here.

Save GUI layout and column orders

Allow column order, width and layout of some views, to be saved in the config file. Note that not all views support this option. Currently, this option has not been implemented in the CPU view.

Don't show close dialog

Do not show the close dialog when the debugger exits.

Show PID in HEX

Show PID in hexadecimal in the attach dialog. If not set, it will use decimal, just like in the Task Manager.

Enable Load/Save Tab Order

Allow x64dbg to load and save tab order. If not set, x64dbg will always use the default tab order.

Show Watch Labels in Side Bar

When you add a watched variable in the watch view, a label with the name of the watched variable can appear in the side bar of the disassembly view if the address is in the sight. They just look like labels for registers. This label might help you understand the operation and progress of a self modifying routine. If disabled, no labels will be added in the side bar for watched variables.

Do not call SetForegroundWindow

When a debug event occurs, x64dbg will focus itself so you can view the state of the debuggee. In some circumstances this might not be desired. This option can be used to tell x64dbg not to focus itself when a debug event occurs.

Other settings

These settings do not appear in settings dialog, nor can they be changed in x64dbg GUI elsewhere, but can be modified by editing the INI configuration file.

Engine

AnimateInterval

If set to a value of milliseconds, animation will proceed every specified milliseconds. Minimum value is 20ms.

MaxSkipExceptionCount

If set (default is 10000), during a run that ignores first-chance exceptions(example, `erun`), it will only ignore that specified number of first-chance exceptions. After that the debuggee will pause when one more first-chance exception happens. If set to 0 first-chance exceptions will always be ignored during such runs.

Gui

NonprintReplaceCharacter

If set to a Unicode value, dump view will use this character to represent nonprintable characters, instead of the default ”.”

NullReplaceCharacter

If set to a Unicode value, dump view will use this character to represent null characters, instead of the default ”.”

Misc

AnimatelgnoreError

Set to 1 to ignore errors while animating, so animation will continue when an error in the animated command occurs.

NoSeasons

Set to 1 to disable easter eggs and Christmas icons.

1.3.4 Dialogs

This sections describes the dialogs in x64dbg graphical user interface.

Contents:

Entropy

This dialog contains a graph that displays the entropy changing trend of selected data.

The height of each point represents the entropy of a continuous 128-byte data block. The data blocks are sampled evenly over the selected buffer. The base address differences between the neighbouring sampled data blocks are the same. If the selected buffer is over 38400 bytes (300*128), there will be gaps between sampled data blocks. If the selected buffer is less than 38400 bytes, the data blocks will overlap. If the selected buffer is less than 128 bytes (size of a data block), then the data block size will be set to half the buffer size.

1.3.5 Translate the x64dbg

The x64dbg GUI is currently available in multiple languages. The launcher is available in both English and Chinese. You can choose the UI language in the *Options* menu.

You can contribute your translations at <http://translate.x64dbg.com>

1.3.6 Tips

This section contains some useful tips about the user interface of this program.

Modules view

The modules view is inside the symbols view.

Relative Addressing

If you double-click the address column, then relative addressing will be used. The address column will show the relative address relative to the double-clicked address.

Tables

You can reorder and hide any column by right-clicking, middle-clicking or double-clicking on the header. Alternatively, you can drag one column header to another one to exchange their order.

Highlight mode

Don't know how to highlight a register? Press Ctrl+H (or click "Highlight mode" menu on the disassembly view). When the red border is shown, click on the register(or command, immediate or any token), then that token will be highlighted with an underline.

Middle mouse button

In disassembly view, pressing middle mouse button will copy the selected address to the clipboard.

Select the entire function or block

You can select the entire function by double-clicking on the checkbox next to the disassembly. This checkbox can also be used to fold the block into a single line.

Code page

You can use the codepage dialog(in the context menu of the dump view) to select a code page. UTF-16LE is the codepage that matches windows unicode encoding. You can use UTF-16LE code page to view strings in a unicode application.

Change Window Title

You can rename the windows of x64dbg by renaming “x64dbg.exe” or “x32dbg.exe” to another name. You should also rename the “x64dbg.ini” or “x32dbg.ini” to keep it the same name as the debugger.

1.3.7 Unusual instructions

Unusual instructions are the instruction which is either privileged, invalid, have no use in ordinary applications, or make attempts to access sensitive information.

To notify the user of their existence, unusual instructions are usually special-colored in the disassembly.

The following instructions are considered unusual:

- All privileged instructions (including I/O instructions and RDMSR/WRMSR)
- RDTSC,RDTSCP,RDRAND,RDSEED
- CPUID
- SYSENTER and SYSCALL
- UD2 and UD2B

1.4 Commands

This is the documentation of x64dbg commands.

Contents:

1.4.1 General Purpose

This section contains various commands that are used for calculations etc.

Content:

inc

Increase a value.

arguments

arg1 Destination.

result

This command does not set any result variables.

dec

Decrease a value.

arguments

arg1 Destination.

result

This command does not set any result variables.

add

Add two values.

arguments

arg1 Destination.

arg2 Source.

result

This command does not set any result variables.

sub

Subtract two values.

arguments

arg1 Destination.

arg2 Source.

result

This command does not set any result variables.

mul

Multiply two values.

arguments

arg1 Destination.

arg2 Source.

result

This command does not set any result variables.

div

Devide two values.

arguments

arg1 Destination.

arg2 Source.

result

This command does not set any result variables.

and

Binary AND two values.

arguments

arg1 Destination.

arg2 Source.

result

This command does not set any result variables.

or

Binary OR two values.

arguments

arg1 Destination.

arg2 Source.

result

This command does not set any result variables.

xor

Binary XOR two values.

arguments

arg1 Destination.

arg2 Source.

result

This command does not set any result variables.

neg

Negate a value.

arguments

arg1 Destination.

result

This command does not set any result variables.

not

Binary NOT a value.

arguments

arg1 Destination.

result

This command does not set any result variables.

bswap

Perform a bswap operation.

arguments

arg1 Destination.

result

This command does not set any result variables.

rol

Binary ROL a value.

arguments

arg1 Destination.

arg2 Source.

result

This command does not set any result variables.

ror

Binary ROR a value.

arguments

arg1 Destination.

arg2 Source.

result

This command does not set any result variables.

shl/sal

Binary SHL/SAL (signed/unsigned shift left) a value.

arguments

arg1 Destination.

arg2 Source.

result

This command does not set any result variables.

shr

Binary SHR (unsigned shift right) a value.

arguments

arg1 Destination.

arg2 Source.

result

This command does not set any result variables.

sar

Binary SAR (signed shift right) a value.

arguments

arg1 Destination.

arg2 Source.

result

This command does not set any result variables.

push

Push a value on the stack.

arguments

arg1 The value to push on the stack.

result

This command does not set any result variables.

pop

Pop a value from the stack.

arguments

[arg1] The destination. When not specified it will just increase CSP.

result

This command does not set any result variables.

test

Binary TEST a value.

arguments

arg1 Value to test.

arg2 Tester.

result

This command sets the internal variables `$_EZ_FLAG` and `$_BS_FLAG`. `$_EZ_FLAG` is set to 1 when `arg1 & arg2 == 0`. `$_BS_FLAG` is always set to 0.

cmp

This command compares two expressions. Notice that when you want to check for values being bigger or smaller, the comparison `arg1>arg2` is made. If this evaluates to true, the `$_BS_FLAG` is set to 1, meaning the value is bigger. So you test if `arg1` is bigger/smaller than `arg2`.

arguments

arg1 First expression to compare.

arg2 Second expression to compare.

result

This command sets the internal variables `$_EZ_FLAG` and `$_BS_FLAG`. They are checked when a branch is performed.

mov/set

Set a variable.

arguments

arg1 Variable name (optionally prefixed with a \$) to set. When the variable does not exist, it will be created.

arg2 Value to store in the variable. If you use #11 22 33# it will write the bytes 11 22 33 in the process memory at arg1.

result

This command does not set any result variables.

1.4.2 Debug Control

Contents:

InitDebug/initdbg/init

Initializes the debugger. This command will load the executable (do some basic checks), set breakpoints on TLS callbacks (if present), set a breakpoint at the process entry point and break at the system breakpoint before giving back control to the user.

arguments

arg1 Path to the executable file to debug. If no full path is given, the `GetCurrentDirectory` API will be called to retrieve a full path. Use quotation marks to include spaces in your path.

[arg2] Commandline to create the process with.

[arg3] Current folder (passed to the `CreateProcess` API).

result

This command will give control back to the user after the system breakpoint is reached. It will set \$pid and \$hp/\$hProcess variables.

StopDebug/stop/dbgstop

Terminate the current debuggee and stop debugging it.

arguments

This command has no arguments.

result

This command does not set any result variables.

AttachDebugger/attach

Attach the debugger to a running process.

arguments

arg1 Process Identifier (PID) of the running process.

[arg2] Handle to an Event Object to signal (this is for internal use only).

[arg3] Thread Identifier (TID) of the thread to resume after attaching (this is for internal use only).

result

This command will give control back to the user after the system breakpoint is reached. It will set \$pid and \$hp/\$hProcess variables.

DetachDebugger/detach

Detach the debugger from the currently-debugged process.

arguments

This command has no arguments.

results

This command does not set any result variables.

run/go/r/g

Free the lock and allow the program to run.

arguments

[arg1] When specified, place a single-shot breakpoint at this location before running.

results

This command does not set any result variables.

erun/ego/er/eg

Free the lock and allow the program to run, **passing all first-chance exceptions to the debuggee.**

arguments

[arg1] When specified, place a single-shot breakpoint at this location before running.

results

This command does not set any result variables.

serun/sego

Free the lock and allow the program to run, **swallowing the current exception, skipping exception dispatching in the debuggee.**

arguments

[arg1] When specified, place a single-shot breakpoint at this location before running.

results

This command does not set any result variables.

pause

Pause the debuggee or stop animation if animation is in progress.

arguments

This command has no arguments.

result

This command does not set any result variables.

DebugContinue/con

Set debugger continue status.

arguments

[arg1] When set (to anything), the exception will be handled by the program. Otherwise the exception will be swallowed.

result

This command does not set any result variables.

StepInto/sti

Single Step (using Trap-Flag).

arguments

[arg1] The number of steps to take. If not specified 1 is used.

result

This command does not set any result variables.

eStepInto/esti

Single Step (using Trap-Flag), **passing all first-chance exceptions to the debuggee.**

arguments

[arg1] The number of steps to take. If not specified 1 is used.

result

This command does not set any result variables.

seStepInto/sesti

Single Step (using Trap-Flag), **swallowing the current exception, skipping exception dispatching in the debuggee.**

arguments

[arg1] The number of steps to take. If not specified 1 is used.

result

This command does not set any result variables.

StepOver/step/sto/st

Step over calls. When the instruction at EIP/RIP isn't a call, a StepInto is performed.

arguments

This command has no arguments.

results

This command does not set any result variables.

eStepOver/estep/esto/est

Step over calls, **passing all first-chance exceptions to the debuggee**. When the instruction at EIP/RIP isn't a call, a `eStepInto` is performed.

arguments

[arg1] The number of steps to take. If not specified 1 is used.

result

This command does not set any result variables.

seStepOver/sestep/sesto/sest

Step over calls, **swallowing the current exception, skipping exception dispatching in the debuggee**. When the instruction at EIP/RIP isn't a call, a `eStepInto` is performed.

arguments

[arg1] The number of steps to take. If not specified 1 is used.

result

This command does not set any result variables.

StepOut/rtr

Return from function by calling `StepOver` until the current instruction is a `RET`.

arguments

[arg1] The number of times to step out. If not specified 1 is used.

result

This command does not set any result variables.

eStepOut/ertr

Return from function by calling eStepOver until the current instruction is a RET. This command **passes all first-chance exceptions to the debuggee**.

arguments

[arg1] The number of times to step out. If not specified 1 is used.

result

This command does not set any result variables.

skip

Skip the next instruction. This command swallows the current exception (if present). Useful if you want to continue after an INT3 command.

arguments

[arg1] The number of instructions to skip. If not specified 1 is used.

result

This command does not set any result variables.

InstrUndo

Undo last instruction stepped. This command is only valid if some instructions are stepped in. Stepping over, running or tracing will clear the history context.

arguments

This command has no arguments.

results

This command does not set any result variables.

1.4.3 Breakpoint Control

This section contains breakpoint control (set/delete/enable/disable) commands.

Context:

SetBPX/bp/bpx

Set an INT3 (SHORT/LONG) or UD2 breakpoint and optionally assign a name to it.

arguments

`arg1` Address to put a breakpoint on. This can be an API name.

[`arg2`] Name of the breakpoint, use quotation marks to include spaces. This name can be used by the EnableBPX, DisableBPX and DeleteBPX functions as alias, but is mainly intended to provide a single line of information about the currently-hit breakpoint. When `arg2` equals to a valid type (`arg3`) the type is used and `arg2` is ignored.

[`arg3`] Breakpoint type. Can be one of the following options in random order: “ss” (single shot breakpoint), “long” (CD03), “ud2” (0F0B) and “short” (CC). You can combine the “ss” option with one of the type options in one string. Example: “SetBPX 00401000,”entrypoint”,ssud2” will set a single shot UD2 breakpoint at 00401000 with the name “entrypoint”. When specifying no type or just the type “ss” the default type will be used. Per default this equals to the “short” type. You can change the default type using the “SetBPXOptions” command.

result

This command does not any result variables.

DeleteBPX/bpc/bc

Delete a breakpoint set using the SetBPX command.

arguments

[`arg1`] Name or address of the breakpoint to delete. If this argument is not specified, all breakpoints will be deleted.

result

This command does not set any result variables.

EnableBPX/bpe/be

Enable a breakpoint set using the SetBPX command.

arguments

[`arg1`] Name or address of the breakpoint to enable. If this argument is not specified, all breakpoints will be enabled.

result

This command does not set any result variables.

DisableBPX/bpd/bd

Disable a breakpoint set using the SetBPX command.

arguments

[arg1] Name or address of the breakpoint to disable. If this argument is not specified, all breakpoints will be disabled.

result

This command does not set any result variables.

SetHardwareBreakpoint/bph/bphws

Set a hardware breakpoint (using debug registers).

arguments

arg1 Address of the hardware breakpoint.

[arg2] Hardware breakpoint type. Can be either 'r' (readwrite), 'w' (write) or 'x' (execute). When not specified, 'x' is assumed.

[arg3] Hardware breakpoint size. Can be either '1', '2', '4' or '8' (x64 only). Per default, '1' is assumed. The address you're putting the hardware breakpoint on must be aligned to the specified size.

result

This command does not set any result variables.

DeleteHardwareBreakpoint/bphc/bphwc

Delete a hardware breakpoint set using the SetHardwareBreakpoint command.

arguments

[arg1] Name or address of the hardware breakpoint to delete. If this argument is not specified, all hardware breakpoints will be deleted.

result

This command does not set any result variables.

EnableHardwareBreakpoint/bphe/bphwe

Enable a previously disabled hardware breakpoint.

arguments

[arg1] Address of the hardware breakpoint to enable. If this argument is not specified, as many as possible hardware breakpoints will be enabled.

result

This command does not set any result variables.

DisableHardwareBreakpoint/bphd/bphwd

Disable a hardware breakpoint.

arguments

[arg1] Address of the hardware breakpoint to disable. If this argument is not specified, all hardware breakpoints will be disabled.

result

This command does not set any result variables.

SetMemoryBPX/membp/bpm

Set a memory breakpoint (GUARD_PAGE) on the whole memory region the provided address is in.

arguments

arg1 Address of or inside a memory region that will be watched.

[arg2] 1/0 restore the memory breakpoint once it's hit? When this value is not equal to '1' or '3', it's assumed to be arg3. This means "bpm eax,r" would be the same command as: "bpm eax,0,r".

[arg3] Breakpoint type, it can be 'a' (read+write+execute) 'r' (read), 'w' (write) or 'x' (execute). Per default, it's 'a' (read+write+execute)

result

This command does not set any result variables.

DeleteMemoryBPX/membpc/bpmc

Delete a memory breakpoint set using the SetMemoryBPX command.

arguments

[arg1] Name or (base) address of the memory breakpoint to delete. If this argument is not specified, all memory breakpoints will be deleted.

result

This command does not set any result variables.

EnableMemoryBreakpoint/membpe/bpme

Enable a previously disabled memory breakpoint.

arguments

[arg1] Address of the memory breakpoint to enable. If this argument is not specified, all memory breakpoints will be enabled.

result

This command does not set any result variables.

DisableMemoryBreakpoint/membpd/bpmd

Disable a memory breakpoint.

arguments

[arg1] Address of the memory breakpoint to disable. If this argument is not specified, all memory breakpoints will be disabled.

result

This command does not set any result variables.

LibrarianSetBreakpoint/bpdll

Set a singleshoot breakpoint on DLL load/unload.

arguments

arg1 DLL Name to break on.

[arg2] 'l' means on load, 'u' means on unload. When not specified, x64dbg will break on both load and unload.

[arg3] When specified, the breakpoint will not be singleshoot. When not specified the breakpoint will be removed after it has been hit.

result

This command does not set any result variables.

LibrarianRemoveBreakpoint/bcdll

Remove a DLL breakpoint.

arguments

arg1 DLL Name to remove the breakpoint from.

result

This command does not set any result variables.

LibrarianEnableBreakpoint/bpedll

Enable a DLL breakpoint set using the LibrarianSetBreakpoint command.

arguments

[arg1] DLL Name of the DLL breakpoint to enable. If this argument is not specified, all DLL breakpoints will be enabled.

result

This command does not set any result variables.

LibrarianDisableBreakpoint/bpdll

Enable a DLL breakpoint set using the LibrarianSetBreakpoint command.

arguments

[arg1] DLL Name of the DLL breakpoint to disable. If this argument is not specified, all DLL breakpoints will be disabled.

result

This command does not set any result variables.

SetExceptionBPX

Set an exception breakpoint. If an exception breakpoint is active, all the exceptions with the same chance and code will be captured as a breakpoint event and will not be handled by the default exception handling policy.

arguments

`arg1` Exception name or code of the new exception breakpoint

`[arg2]` Chance. Set to `first/1` to capture first-chance exceptions, `second/2` to capture second-chance exceptions, `all/3` to capture all exceptions. Default value is `first`.

result

This command does not any result variables.

DeleteExceptionBPX

Delete an exception breakpoint set using the SetExceptionBPX command.

arguments

`[arg1]` Name, exception name or code of the exception breakpoint to delete. If this argument is not specified, all exception breakpoints will be deleted.

result

This command does not set any result variables.

EnableExceptionBPX

Enable an exception breakpoint set using the SetExceptionBPX command.

arguments

`[arg1]` Name, exception name or code of the exception breakpoint to enable. If this argument is not specified, all exception breakpoints will be enabled.

result

This command does not set any result variables.

DisableExceptionBPX

Disable an exception breakpoint set using the SetExceptionBPX command.

arguments

[arg1] Name, exception name or code of the exception breakpoint to enable. If this argument is not specified, all exception breakpoints will be disabled.

result

This command does not set any result variables.

bpgoto

Configure the breakpoint so that when the program reaches it, the program will be directed to a new location. It is equivalent to the following commands:

```
SetBreakpointCondition arg1, 0
SetBreakpointCommand arg1, "CIP=arg2"
SetBreakpointCommandCondition arg1, 1
SetBreakpointFastResume arg1, 0
```

arguments

arg1 The address of the breakpoint.

arg2 The new address to execute if the breakpoint is reached.

results

This command does not set any result variables.

bplist

Get a list of breakpoints. This list includes their state (enabled/disabled), their type, their address and (optionally) their names.

arguments

This command has no arguments.

result

This command does not set any result variables. A list entry has the following format:

```
STATE:TYPE:ADDRESS[:NAME]
```

STATE can be 0 or 1. 0 means disabled, 1 means enabled. Only singleshoot and 'normal' breakpoints can be disabled.

TYPE can be one of the following values: BP, SS, HW and GP. BP stands for a normal breakpoint (set using the SetBPX command), SS stands for SINGLESHOT, HW stands for HARDWARE and GP stand for Guard Page, the way of setting memory breakpoints.

ADDRESS is the breakpoint address, given in 32 and 64 bits for the x32 and x64 debugger respectively.

NAME is the name assigned to the breakpoint.

SetBPXOptions/bptype

Set the default type for the “SetBPX” command.

arguments

arg1 Default type. This can be “short” (CC), “long” (CD03) or “ud2” (0F0B). Type default type affects both NORMAL and SINGLESHOT breakpoints.

result

This command does not set any result variables.

1.4.4 Conditional Breakpoint Control

This section describes commands that can be used to set various advanced properties of breakpoints.

Contents:

SetBreakpointName/bpname

Sets the name of a software breakpoint. It will be displayed in the breakpoints view and in the log when the breakpoint is hit.

arguments

arg1 The address of an existing software breakpoint.

[arg2] The name of the breakpoint (empty when not specified).

result

This command does not set any result variables.

SetBreakpointCondition/bpcond/bpcnd

Sets the software breakpoint condition. When this condition is set, it is evaluated every time the breakpoint hits and the debugger would stop only if condition is not 0.

arguments

arg1 The address of the breakpoint.

[arg2] The condition expression.

result

This command does not set any result variables.

SetBreakpointLog/bplog/bpl

Sets log text when a software breakpoint is hit. When log condition is not specified, it will always be logged regardless of the break condition, otherwise it will be logged when the logging condition is satisfied.

arguments

arg1 The address of the breakpoint.

[arg2] The log format string (see introduction/formatting).

result

This command does not set any result variables.

SetBreakpointLogCondition/bplogcondition

Sets the logging condition of a software breakpoint. When log condition is not specified, log text always be logged regardless of the break condition, otherwise it will be logged when the logging condition is satisfied.

arguments

arg1 The address of the breakpoint.

[arg2] The logging condition (default condition when not specified).

result

This command does not set any result variables.

SetBreakpointCommand

Sets the command to execute when a software breakpoint is hit. If the command condition is not specified, it will be executed when the debugger breaks, otherwise it will be executed when the condition is satisfied.

arguments

arg1 The address of the breakpoint.

[arg2] The command (empty when not specified).

result

This command does not set any result variables.

SetBreakpointCommandCondition

Sets the command condition of a software breakpoint. When command condition is not specified, the command will be executed when the debugger would break, otherwise it will be executed when the condition is satisfied.

arguments

arg1 The address of the breakpoint.

[arg2] The command condition (default condition when not specified).

result

This command does not set any result variables.

SetBreakpointFastResume

Sets the fast resume flag of a software breakpoint. If this flag is set and the break condition doesn't evaluate to break, no GUI, plugin, logging or any other action will be performed, except for incrementing the hit counter.

arguments

arg1 The address of the breakpoint.

[arg2] The fast resume flag. If it is 0 (default), fast resume is disabled, otherwise it is enabled

result

This command does not set any result variables.

SetBreakpointSingleshoot

Sets the singleshoot flag of a software breakpoint. If this flag is set the breakpoint will be removed on the first hit.

arguments

arg1 The address of the breakpoint.

[arg2] The singleshoot flag. If it is 0 (default), singleshoot is disabled, otherwise it is enabled

result

This command does not set any result variables.

SetBreakpointSilent

Sets the silent flag of a software breakpoint. If this flag is set, the default log message will not appear. User-defined log is not affected.

arguments

arg1 The address of the breakpoint.

[arg2] The silent flag. If it is 0 (default), silent is disabled, otherwise it is enabled

result

This command does not set any result variables.

GetBreakpointHitCount

Gets the hit counter of a software breakpoint.

arguments

arg1 The address of the breakpoint.

result

\$result will be set to the current value of the hit counter.

ResetBreakpointHitCount

Resets the hit counter of a software breakpoint.

arguments

arg1 The address of the breakpoint.

[arg2] The new hit count (zero when not specified).

result

This command does not set any result variables.

SetHardwareBreakpointName/bphwname

Sets the name of a hardware breakpoint. It will be displayed in the breakpoints view and in the log when the breakpoint is hit.

arguments

arg1 The address of an existing hardware breakpoint.

[arg2] The name of the breakpoint (empty when not specified).

result

This command does not set any result variables.

SetHardwareBreakpointCondition/bphwcond

Sets the hardware breakpoint condition. When this condition is set, it is evaluated every time the breakpoint hits and the debugger would stop only if condition is not 0.

arguments

arg1 The address of the breakpoint.

[arg2] The condition expression.

result

This command does not set any result variables.

SetHardwareBreakpointLog/bphwlog

Sets log text when a hardware breakpoint is hit. When log condition is not specified, it will always be logged regardless of the break condition, otherwise it will be logged when the logging condition is satisfied.

arguments

arg1 The address of the breakpoint.

[arg2] The log format string (see introduction/formatting).

result

This command does not set any result variables.

SetHardwareBreakpointLogCondition/bphwlogcondition

Sets the logging condition of a hardware breakpoint. When log condition is not specified, log text always be logged regardless of the break condition, otherwise it will be logged when the logging condition is satisfied.

arguments

arg1 The address of the breakpoint.

[arg2] The logging condition (default condition when not specified).

result

This command does not set any result variables.

SetHardwareBreakpointCommand

Sets the command to execute when a hardware breakpoint is hit. If the command condition is not specified, it will be executed when the debugger breaks, otherwise it will be executed when the condition is satisfied.

arguments

arg1 The address of the breakpoint.

[arg2] The command (empty when not specified).

result

This command does not set any result variables.

SetHardwareBreakpointCommandCondition

Sets the command condition of a hardware breakpoint. When command condition is not specified, the command will be executed when the debugger would break, otherwise it will be executed when the condition is satisfied.

arguments

arg1 The address of the breakpoint.

[arg2] The command condition (default condition when not specified).

result

This command does not set any result variables.

SetHardwareBreakpointFastResume

Sets the fast resume flag of a hardware breakpoint. If this flag is set and the break condition doesn't evaluate to break, no GUI, plugin, logging or any other action will be performed, except for incrementing the hit counter.

arguments

arg1 The address of the breakpoint.

[arg2] The fast resume flag. If it is 0 (default), fast resume is disabled, otherwise it is enabled

result

This command does not set any result variables.

SetHardwareBreakpointSingleshoot

Sets the singleshoot flag of a hardware breakpoint. If this flag is set the breakpoint will be removed on the first hit.

arguments

arg1 The address of the breakpoint.

[arg2] The singleshoot flag. If it is 0 (default), singleshoot is disabled, otherwise it is enabled

result

This command does not set any result variables.

SetHardwareBreakpointSilent

Sets the silent flag of a hardware breakpoint. If this flag is set, the default log message will not appear. User-defined log is not affected.

arguments

arg1 The address of the breakpoint.

[arg2] The silent flag. If it is 0 (default), silent is disabled, otherwise it is enabled

result

This command does not set any result variables.

GetHardwareBreakpointHitCount

Gets the hit counter of a hardware breakpoint.

arguments

arg1 The address of the breakpoint.

result

\$result will be set to the current value of the hit counter.

ResetHardwareBreakpointHitCount

Resets the hit counter of a hardware breakpoint.

arguments

arg1 The address of the breakpoint.

[arg2] The new hit count (zero when not specified).

result

This command does not set any result variables.

SetMemoryBreakpointName/bpmname

Sets the name of a memory breakpoint. It will be displayed in the breakpoints view and in the log when the breakpoint is hit.

arguments

arg1 The address of an existing memory breakpoint.

[arg2] The name of the breakpoint (empty when not specified).

result

This command does not set any result variables.

SetMemoryBreakpointCondition/bpmcond

Sets the memory breakpoint condition. When this condition is set, it is evaluated every time the breakpoint hits and the debugger would stop only if condition is not 0.

arguments

arg1 The address of the breakpoint.

[arg2] The condition expression.

result

This command does not set any result variables.

SetMemoryBreakpointLog/bpmlog

Sets log text when a memory breakpoint is hit. When log condition is not specified, it will always be logged regardless of the break condition, otherwise it will be logged when the logging condition is satisfied.

arguments

arg1 The address of the breakpoint.

[arg2] The log format string (see introduction/formatting).

result

This command does not set any result variables.

SetMemoryBreakpointLogCondition/bpmlogcondition

Sets the logging condition of a memory breakpoint. When log condition is not specified, log text always be logged regardless of the break condition, otherwise it will be logged when the logging condition is satisfied.

arguments

arg1 The address of the breakpoint.

[arg2] The logging condition (default condition when not specified).

result

This command does not set any result variables.

SetMemoryBreakpointCommand

Sets the command to execute when a memory breakpoint is hit. If the command condition is not specified, it will be executed when the debugger breaks, otherwise it will be executed when the condition is satisfied.

arguments

arg1 The address of the breakpoint.

[arg2] The command (empty when not specified).

result

This command does not set any result variables.

SetMemoryBreakpointCommandCondition

Sets the command condition of a memory breakpoint. When command condition is not specified, the command will be executed when the debugger would break, otherwise it will be executed when the condition is satisfied.

arguments

arg1 The address of the breakpoint.

[arg2] The command condition (default condition when not specified).

result

This command does not set any result variables.

SetMemoryBreakpointFastResume

Sets the fast resume flag of a memory breakpoint. If this flag is set and the break condition doesn't evaluate to break, no GUI, plugin, logging or any other action will be performed, except for incrementing the hit counter.

arguments

arg1 The address of the breakpoint.

[arg2] The fast resume flag. If it is 0 (default), fast resume is disabled, otherwise it is enabled

result

This command does not set any result variables.

SetMemoryBreakpointSingleshoot

Sets the singleshoot flag of a memory breakpoint. If this flag is set the breakpoint will be removed on the first hit.

arguments

arg1 The address of the breakpoint.

[arg2] The singleshoot flag. If it is 0 (default), singleshoot is disabled, otherwise it is enabled

result

This command does not set any result variables.

SetMemoryBreakpointSilent

Sets the silent flag of a memory breakpoint. If this flag is set, the default log message will not appear. User-defined log is not affected.

arguments

arg1 The address of the breakpoint.

[arg2] The silent flag. If it is 0 (default), silent is disabled, otherwise it is enabled

result

This command does not set any result variables.

GetMemoryBreakpointHitCount

Gets the hit counter of a memory breakpoint.

arguments

arg1 The address of the breakpoint.

result

\$result will be set to the current value of the hit counter.

ResetMemoryBreakpointHitCount

Resets the hit counter of a memory breakpoint.

arguments

arg1 The address of the breakpoint.

[arg2] The new hit count (zero when not specified).

result

This command does not set any result variables.

SetLibrarianBreakpointName

Sets the name of a librarian breakpoint. It will be displayed in the breakpoints view and in the log when the breakpoint is hit.

arguments

arg1 The DLL name.

[arg2] The name of the breakpoint (empty when not specified).

result

This command does not set any result variables.

SetLibrarianBreakpointCondition

Sets the librarian breakpoint condition. When this condition is set, it is evaluated every time the breakpoint occurs and the debugger would stop only if condition is not 0.

arguments

arg1 The DLL name.

[arg2] The condition expression.

result

This command does not set any result variables.

SetLibrarianBreakpointLog

Sets log text when a librarian breakpoint is hit. When log condition is not specified, it will always be logged regardless of the break condition, otherwise it will be logged when the logging condition is satisfied.

arguments

arg1 The DLL name.

[arg2] The log format string (see introduction/formatting).

result

This command does not set any result variables.

SetLibrarianBreakpointLogCondition

Sets the logging condition of a librarian breakpoint. When log condition is not specified, log text always be logged regardless of the break condition, otherwise it will be logged when the logging condition is satisfied.

arguments

arg1 The DLL name.

[arg2] The logging condition (default condition when not specified).

result

This command does not set any result variables.

SetLibrarianBreakpointCommand

Sets the command to execute when a librarian breakpoint is hit. If the command condition is not specified, it will be executed when the debugger breaks, otherwise it will be executed when the condition is satisfied.

arguments

arg1 The DLL name.

[arg2] The command (empty when not specified).

result

This command does not set any result variables.

SetLibrarianBreakpointCommandCondition

Sets the command condition of a librarian breakpoint. When command condition is not specified, the command will be executed when the debugger would break, otherwise it will be executed when the condition is satisfied.

arguments

arg1 The DLL name.

[arg2] The command condition (default condition when not specified).

result

This command does not set any result variables.

SetLibrarianBreakpointFastResume

Sets the fast resume flag of a librarian breakpoint. If this flag is set and the break condition doesn't evaluate to break, no GUI, plugin, logging or any other action will be performed, except for incrementing the hit counter.

arguments

arg1 The DLL name.

[arg2] The fast resume flag. If it is 0 (default), fast resume is disabled, otherwise it is enabled

result

This command does not set any result variables.

SetLibrarianBreakpointSingleshoot

Sets the singleshoot flag of a librarian breakpoint. If this flag is set the librarian breakpoint will be removed on the first hit.

arguments

arg1 The DLL name.

[arg2] The singleshoot flag. If it is 0 (default), singleshoot is disabled, otherwise it is enabled

result

This command does not set any result variables.

SetLibrarianBreakpointSilent

Sets the silent flag of a librarian breakpoint. If this flag is set, the default log message will not appear. User-defined log is not affected.

arguments

arg1 The DLL name.

[arg2] The silent flag. If it is 0 (default), silent is disabled, otherwise it is enabled

result

This command does not set any result variables.

GetLibrarianBreakpointHitCount

Gets the hit counter of a librarian breakpoint.

arguments

arg1 The DLL name.

result

\$result will be set to the current value of the hit counter.

ResetLibrarianBreakpointHitCount

Resets the hit counter of a librarian breakpoint.

arguments

arg1 The DLL name.

[arg2] The new hit count (zero when not specified).

result

This command does not set any result variables.

SetExceptionBreakpointName

Sets the name of an exception breakpoint. It will be displayed in the breakpoints view and in the log when the breakpoint is hit.

arguments

arg1 The name, exception name or code of the exception breakpoint.

[arg2] The name of the breakpoint (empty when not specified).

result

This command does not set any result variables.

SetExceptionBreakpointCondition

Sets the exception breakpoint condition. When this condition is set, it is evaluated every time the exception occurs (chance must match) and the debugger would stop only if condition is not 0.

arguments

arg1 The name, exception name or code of the exception breakpoint.

[arg2] The condition expression.

result

This command does not set any result variables.

SetExceptionBreakpointLog

Sets log text when an exception breakpoint is hit. When log condition is not specified, it will always be logged regardless of the break condition, otherwise it will be logged when the logging condition is satisfied.

arguments

arg1 The name, exception name or code of the exception breakpoint.

[arg2] The log format string (see introduction/formatting).

result

This command does not set any result variables.

SetExceptionBreakpointLogCondition

Sets the logging condition of an exception breakpoint. When log condition is not specified, log text always be logged regardless of the break condition, otherwise it will be logged when the logging condition is satisfied.

arguments

arg1 The name, exception name or code of the exception breakpoint.

[arg2] The logging condition (default condition when not specified).

result

This command does not set any result variables.

SetExceptionBreakpointCommand

Sets the command to execute when an exception breakpoint is hit. If the command condition is not specified, it will be executed when the debugger breaks, otherwise it will be executed when the condition is satisfied.

arguments

arg1 The name, exception name or code of the exception breakpoint.

[arg2] The command (empty when not specified).

result

This command does not set any result variables.

SetExceptionBreakpointCommandCondition

Sets the command condition of an exception breakpoint. When command condition is not specified, the command will be executed when the debugger would break, otherwise it will be executed when the condition is satisfied.

arguments

arg1 The name, exception name or code of the exception breakpoint.

[arg2] The command condition (default condition when not specified).

result

This command does not set any result variables.

SetExceptionBreakpointFastResume

Sets the fast resume flag of an exception breakpoint. If this flag is set and the break condition doesn't evaluate to break, no GUI, plugin, logging or any other action will be performed, except for incrementing the hit counter.

arguments

arg1 The name, exception name or code of the exception breakpoint.

[arg2] The fast resume flag. If it is 0 (default), fast resume is disabled, otherwise it is enabled

result

This command does not set any result variables.

SetExceptionBreakpointSingleshoot

Sets the singleshoot flag of an exception breakpoint. If this flag is set the exception breakpoint will be removed on the first hit.

arguments

arg1 The name, exception name or code of the exception breakpoint.

[arg2] The singleshoot flag. If it is 0 (default), singleshoot is disabled, otherwise it is enabled

result

This command does not set any result variables.

SetExceptionBreakpointSilent

Sets the silent flag of an exception breakpoint. If this flag is set, the default log message will not appear. User-defined log is not affected.

arguments

arg1 The name, exception name or code of the exception breakpoint.

[arg2] The silent flag. If it is 0 (default), silent is disabled, otherwise it is enabled

result

This command does not set any result variables.

GetExceptionBreakpointHitCount

Gets the hit counter of an exception breakpoint.

arguments

arg1 The name, exception name or code of the exception breakpoint.

result

\$result will be set to the current value of the hit counter.

ResetExceptionBreakpointHitCount

Resets the hit counter of an exception breakpoint.

arguments

arg1 The name, exception code or name of the exception breakpoint.

[arg2] The new hit count (zero when not specified).

result

This command does not set any result variables.

See also:

Conditional Breakpoints

1.4.5 Tracing

Contents:

TraceIntoConditional/ticnd

Trace the program by `StepInto`, until the specified condition is satisfied, or maximum number of steps reached.

arguments

`arg1` The condition used. When this is evaluated to be a value other than 0, tracing will stop.

`[arg2]` The maximum step count to trace before the debugger gives up.

results

This command does not set any result variables.

TraceOverConditional/tocnd

Trace the program by `StepOver`, until the specified condition is satisfied, or maximum number of steps reached.

arguments

`arg1` The condition used. When this is evaluated to be a value other than 0, tracing will stop.

`[arg2]` The maximum step count to trace before the debugger gives up.

results

This command does not set any result variables.

TraceIntoBeyondTraceRecord/tibt

Perform `StepInto` until the program reaches somewhere outside the trace record.

arguments

`[arg1]` The break condition of tracing. When this condition is satisfied, tracing will stop regardless of EIP/RIP location. If this argument is not specified then tracing will be unconditional.

`[arg2]` The maximum steps before the debugger gives up. If this argument is not specified, the default value will be 50000.

result

This command does not set any result variables.

TraceOverBeyondTraceRecord/tobt

Perform `StepOver` until the program reaches somewhere outside the trace record.

arguments

[arg1] The break condition of tracing. When this condition is satisfied, tracing will stop regardless of EIP/RIP location. If this argument is not specified then tracing will be unconditional.

[arg2] The maximum steps before the debugger gives up. If this argument is not specified, the default value will be 50000.

result

This command does not set any result variables.

TraceIntoTraceRecord/tiit

Perform StepInto until the program reaches somewhere inside the trace record.

arguments

[arg1] The break condition of tracing. When this condition is satisfied, tracing will stop regardless of EIP/RIP location. If this argument is not specified then tracing will be unconditional.

[arg2] The maximum steps before the debugger gives up. If this argument is not specified, the default value will be 50000.

result

This command does not set any result variables.

TraceOverTraceRecord/toit

Perform StepOver until the program reaches somewhere inside the trace record.

arguments

[arg1] The break condition of tracing. When this condition is satisfied, tracing will stop regardless of EIP/RIP location. If this argument is not specified then tracing will be unconditional.

[arg2] The maximum steps before the debugger gives up. If this argument is not specified, the default value will be 50000.

result

This command does not set any result variables.

RunToParty

Run the program until the program reaches somewhere belonging to the party number. This works by putting temporary memory breakpoints on all memory pages with matching party number.

arguments

arg1 The party number. This value cannot be an expression.

results

This command does not set any result variables.

RunToUserCode/rtu

Run until user code is reached. It is equivalent to `RunToParty 0`.

arguments

This command has no arguments.

results

This command does not set any result variables.

TraceSetLog/SetTraceLog

Change the trace log text and condition during tracing. See *Conditional Tracing* for more information.

arguments

[arg1] Log text. If not specified the current text/condition is cleared.

[arg2] Log condition. If not specified the default condition is used.

results

This command does not set any result variables.

TraceSetCommand/SetTraceCommand

Change the trace command text and condition during tracing. See *Conditional Tracing* for more information.

arguments

[arg1] Command text. If not specified the current text/condition is cleared.

[arg2] Command condition. If not specified the default condition is used.

results

This command does not set any result variables.

TraceSetLogFile/SetTraceLogFile

Redirect the trace log to a file.

arguments

arg1 File name to redirect the trace log to. This file will be cleared and overwritten when the trace starts. **This does nothing if you don't set the log text!**

results

This command does not set any result variables.

StartRunTrace/opentrace

Start recording a run trace with a specified file. The file will also be opened in the trace view. Note you need to use *TraceIntoConditional* or other command to actually trace the program.

arguments

arg1 The file name. Default file extension “trace32” or “trace64” is not added automatically.

result

This command does not set any result variables.

StopRunTrace/tc

Stops recording a run trace and closes file.

arguments

This command has no arguments.

result

This command does not set any result variables.

1.4.6 Thread Control

Contents:

createthread[,threadcreate,newthread,threadnew]

Create a new thread at the specified entry.

arguments

arg1 The entry of the new thread.

[arg2] The argument of the new thread. If the argument is not specified, the default argument is 0.

results

\$result will be set to the thread id of the new thread.

switchthread/threadswitch

Switch the internal current thread to another thread (resulting in different callstack + different registers displayed).

arguments

[arg1] ThreadId of the thread to switch to (see the Threads tab). When not specified, the main thread is used.

result

This command does not set any result variables.

suspendthread/threadsuspend

Suspend a thread in the debuggee.

arguments

[arg1] ThreadId of the thread to suspend (see the Threads tab). When not specified, the main thread is used.

result

This command does not set any result variables.

resumethread/threadresume

Resume a thread in the debuggee.

arguments

[arg1] ThreadId of the thread to resume (see the Threads tab). When not specified, the main thread is used.

result

This command does not set any result variables.

killthread/threadkill

Kill a thread in the debuggee.

arguments

[arg1] ThreadId of the thread to kill (see the Threads tab). When not specified, the main thread is used.

[arg2] Thread exit code. When not specified, 0 will be used.

result

This command does not set any result variables.

suspendallthreads/threadsuspendall

Suspend all threads in the debuggee.

arguments

This command has no arguments.

result

This command does not set any result variables.

resumeallthreads/threadresumeall

Resume all threads in the debuggee.

arguments

This command has no arguments.

result

This command does not set any result variables.

setthreadpriority/setprioritythread/threadsetpriority

Set thread priority in the debuggee.

arguments

arg1 ThreadId of the thread to change the priority of (see the Threads tab).

arg2 Priority value, this can be the integer of a valid thread priority (see MSDN) or one of the following values: "Normal", "AboveNormal", "TimeCritical", "Idle", "BelowNormal", "Highest", "Lowest".

result

This command does not set any result variables.

setthreadname/threadsetname

Set thread name (only for the debugger, nothing changes in the debuggee).

arguments

arg1 ThreadId of the thread to change the priority of (see the Threads tab).

arg2 New thread name. Leave empty to remove the current name.

result

This command does not set any result variables.

1.4.7 Memory Operations

This section contains commands to manipulate memory inside the debuggee.

Contents:**alloc**

Allocate memory in the debuggee (using VirtualAllocEx). The memory is allocated with PAGE_EXECUTE_READWRITE protection.

arguments

[arg1] Size of the memory to allocate. When not specified, a default size of 0x1000 is used.

[arg2] Address to allocate the memory at. Unspecified or zero means a random address.

result

This command sets \$result to the allocated memory address. It also sets the \$lalloc variable to the allocated memory address when VirtualAllocEx succeeded.

Fill/memset

Set memory of the debuggee to a specified byte.

arguments

arg1 Memory address to start setting bytes.

arg2 Value (byte) to set memory with.

[arg3] Size to set. When not specified the rest of the page is used.

result

This command does not set any result variables.

free

Free memory in the debuggee (using VirtualFreeEx).

arguments

[arg1] Address of the memory to free. When not specified, the value at \$lastalloc is used.

result

This command sets \$result to 1 if VirtualFreeEx succeeded, otherwise it's set to 0. \$lastalloc is set to zero when the address specified is equal to \$lastalloc.

getpagerights/getpagerights/getrightspage

Get the rights of a memory page.

arguments

arg1 Memory Address of page (it fix the address if this arg is not the top address of a page).

result

This command does not set any result variables.

setpagerights/setpagerights/setrightspage

Change the rights of a memory page.

arguments

arg1 Memory Address of page (it fix the address if this arg is not the top address of a page).

arg2 New Rights, this can be one of the following values: "Execute", "ExecuteRead", "ExecuteReadWrite", "ExecuteWriteCopy", "NoAccess", "ReadOnly", "ReadWrite", "WriteCopy". You can add a G at first for add PAGE GUARD. example: "GReadOnly". Read the MSDN for more info.

result

This command does not set any result variables.

savedata

Save a memory region to disk.

arguments

arg1 The filename. If you use `:memdump:` as name it will save a file as `memdump_pid_addr_size.bin` in the x64dbg directory. You can use *String Formatting* here.

arg2 The address of the memory region.

arg3 The size of the memory region.

results

This command does not set any result variables.

1.4.8 Operating System Control

This section contains the commands that can be used to control certain properties managed by the operating system.

Content:

DisablePrivilege

Revoke the privilege.

arguments

arg1 The name of the privilege. Example: `SeDebugPrivilege`

results

This command does not set any result variables.

EnablePrivilege

Permit the privilege.

arguments

arg1 The name of the privilege. Example: SeDebugPrivilege

results

This command does not set any result variables.

GetPrivilegeState

Query whether the privilege is enabled on the debuggee.

arguments

arg1 The name of the privilege. Example: SeDebugPrivilege.

results

This command sets `$result` to 1 if the privilege is disabled on the debuggee, 2 or 3 if the privilege is enabled on the debuggee, 0 if the privilege is not found in the privilege collection of the token of the debuggee or something is wrong with the API.

handleclose/closehandle

Close a remote handle so that its system resources can be released.

arguments

arg1 The handle value of the handle, in the context of the debuggee.

results

This command does not set any result variables.

1.4.9 Watch Control

This section describes the commands that control the watch view.

Contents:

AddWatch

Add a watch item.

arguments

`arg1` The expression to watch.

`[arg2]` The data type of the watch item. `uint` displays hexadecimal value, `int` displays signed decimal value, `ascii` displays the ASCII string pointed by the value. `unicode` displays the Unicode string pointed by the value. `uint` is the default type.

results

This command sets `$result` value to the id of the watch item.

DelWatch

Delete a watch item.

arguments

`arg1` The id of the watch item to delete.

result

This command does not set any result variables.

SetWatchdog

Set the watchdog mode of a watch item.

arguments

`arg1` The id of the watch item.

`[arg2]` The watchdog mode. Possible values:

- `disabled` : Watchdog is disabled.
- `changed` : Watchdog is triggered when the value is changed.
- `unchanged` : Watchdog is triggered when the value is not changed.
- `istrue` : Watchdog is triggered when the value is not 0.
- `isfalse` : Watchdog is triggered when the value is 0.

When this argument is not specified, the mode will be set to “changed” if the current watchdog mode is “disabled”, otherwise watchdog will be disabled.

results

This command does not set any result variables.

SetWatchExpression

Change the expression of an existing watch item.

arguments

arg1 The id of the watch item.

arg2 The new expression to watch.

arg3 The new data type of the watch item.

results

This command does not set any result variables.

SetWatchName

Rename a watch item.

arguments

arg1 The id of the watch item to rename.

arg2 The new name.

results

This command does not set any result variables.

CheckWatchdog

Evaluate all the watch items, trigger or reset watchdog when appropriate.

arguments

This command has no arguments.

results

This command set `$result` to 1 if any watchdog is triggered, 0 otherwise.

1.4.10 Variables

This section contains commands that can manipulate variables.

Contents:

var/varnew

Declare a new variable.

arguments

arg1 Variable name (will be prefixed with '\$' if not done).

[arg2] Initial variable value (see console input for details).

result

This command does not set any result variables.

vardel

Delete a user-defined variable.

arguments

arg1 Name of the variable to delete (\$ will be prepended when not present).

result

This command does not set any result variables.

varlist

Get a list of all variables and their values.

arguments

[arg1] Filter (USER, SYSTEM, READONLY).

result

This command does not set any result variables.

1.4.11 Searching

This section contains commands that are used to search data.

Contents:

find

Find a pattern.

arguments

`arg1` The address to start searching from. Notice that the searching will stop when the end of the memory page this address resides in has been reached. This means you cannot search the complete process memory without enumerating the memory pages first.

`arg2` The byte pattern to search for. This byte pattern can contain wildcards (?) for example: `EB0?90??8D`.

[`arg3`] The size of the data to search in. Default is the size of the memory region.

result

The `$result` variable is set to the virtual address of the address that matches the byte pattern. `$result` will be 0 when the pattern could not be matched.

findall

Find all occurrences of a pattern.

arguments

`arg1` The address to start searching from. Notice that the searching will stop when the end of the memory page this address resides in has been reached. This means you cannot search the complete process memory without enumerating the memory pages first.

`arg2` The byte pattern to search for. This byte pattern can contain wildcards (?) for example: `EB0?90??8D`.

[`arg3`] The size of the data to search in. Default is the size of the memory region.

result

`$result` is set to the number of occurrences.

findallmem/findmemall

Find all occurrences of a pattern in the entire memory map.

arguments

arg1 The address to start searching from.

arg2 The byte pattern to search for. This byte pattern can contain wildcards (?) for example: EB0?90??8D.

[arg3] The size of the data to search in. Default is the entire memory map.

result

\$result is set to the number of occurrences.

findasm/asmfind

Find assembled instruction.

arguments

arg1 Instruction to look for (make sure to use quoted “mov eax, ebx” to ensure you actually search for that instruction). You can use *String Formatting* here.

[arg2] Address of/inside a memory page to look in. When not specified CIP will be used.

[arg3] The size of the data to search in. Default is the size of the memory region.

result

The \$result variable is set to the number of references found.

findguid/guidfind

Find references to GUID. The referenced GUID must be registered in the system, otherwise it will not be found.

arguments

[arg1] The base of the memory range. If not specified, RIP or EIP will be used.

[arg2] The size of the memory range.

[arg3] The region to search. 0 is current region (specified with arg1 and arg2). 1 is current module (the module specified with arg1). 2 is all modules.

results

Set \$result to 1 if any GUID is found, 0 otherwise.

reffind/findref/ref

Find references to a certain value.

arguments

arg1 The value to look for.

[arg2] Address of/inside a memory page to look in. When not specified CIP will be used.

[arg3] The size of the data to search in.

result

The \$result variable is set to the number of references found.

reffindrange/findrefrange/refrange

Find references to a certain range of values.

arguments

arg1 Start of the range (will be included in the results when found).

[arg2] End of range (will be included in the results when found). When not specified the first argument will be used.

[arg3] Address of/inside a memory page to look in. When not specified CIP will be used.

[arg4] The size of the data to search in.

result

The \$result variable is set to the number of references found.

refstr/strref

Find referenced text strings.

arguments

[arg1] Address of/inside a memory page to find referenced text strings in. When not specified CIP will be used.

[arg2] The size of the data to search in.

result

The \$result variable is set to the number of string references found.

modcallfind

Find all inter-modular calls.

arguments

[arg1] Address of/inside a memory page to find inter-modular calls in. When not specified EIP/RIP will be used.

[arg2] The size of the data to search in.

result

The `$result` variable is set to the number of inter-modular calls found.

setmaxfindresult/findsetmaxresult

Set the maximum number of occurrences found.

arguments

arg1 The maximum number of occurrences. The default value is 5000.

results

This command does not set any result variables.

1.4.12 User Database

This section contains commands that manipulate the user database (comments, labels and bookmarks).

dbsave/savedb

Save a program database from memory to disk.

arguments

[arg1] Path to save the database to. If not specified your current program database is used.

result

This command does not set any result variables.

dbload/loaddb

Load a program database from disk in memory.

arguments

[arg1] Path to load the database from. If specified your current data will not be automatically cleared (import). If not specified all your data will be cleared and the current program database is reloaded from disk.

result

This command does not set any result variables.

dbclear/cleardb

Clear the program database from memory (not from disk).

arguments

This command has no arguments.

result

This command does not set any result variables.

commentset/cmt/cmtset

Set a comment.

arguments

arg1 Address to set the comment at (preferably inside a module).

arg2 Comment text.

result

This command does not set any result variables.

commentdel/cmtc/cmtdel

Delete a comment.

arguments

arg1 Address of the comment to delete.

result

This command does not set any result variables.

commentlist

List user-defined comments in reference view.

arguments

This command has no arguments.

result

\$result will be set to the number of user-defined comments.

commentclear

Delete all comments in all modules.

arguments

This command has no arguments.

result

This command does not set any result variables.

labelset/lbl/lblset

Set a label.

arguments

arg1 Address to set the label at (preferably inside a module).

arg2 Label text. You can use *String Formatting* here.

result

This command does not set any result variables.

labeldel/lblc/lbldel

Delete a label.

arguments

arg1 Address of the label to delete.

result

This command does not set any result variables.

labellist

List user-defined labels in reference view.

arguments

This command has no arguments.

result

\$result will be set to the number of user-defined labels.

labelclear

Delete all labels in all modules.

arguments

This command has no arguments.

result

This command does not set any result variables.

bookmarkset/bookmark

Set a bookmark.

arguments

arg1 Address to set the bookmark at (preferably inside a module).

result

This command does not set any result variables.

bookmarkdel/bookmarkc

Delete a bookmark.

arguments

arg1 Address of the bookmark to delete.

result

This command does not set any result variables.

bookmarklist

List user-defined bookmarks in reference view.

arguments

This command has no arguments.

result

\$result will be set to the number of user-defined bookmarks.

bookmarkclear

Delete all bookmarks in all modules.

arguments

This command has no arguments.

result

This command does not set any result variables.

functionadd/func

Add a function.

arguments

arg1 Function range start.

arg2 Function range end.

result

This command does not set any result variables.

functiondel/func

Delete a function.

arguments

arg1 Address inside the function range to delete.

result

This command does not set any result variables.

functionlist

List user-defined functions in reference view.

arguments

This command has no arguments.

result

\$result will be set to the number of user-defined functions.

functionclear

Delete all functions in all modules.

arguments

This command has no arguments.

result

This command does not set any result variables.

argumentadd

Add a argument.

arguments

arg1 argument range start.

arg2 argument range end.

result

This command does not set any result variables.

argumentdel

Delete a argument.

arguments

arg1 Address inside the argument range to delete.

result

This command does not set any result variables.

argumentlist

List user-defined arguments in reference view.

arguments

This command has no arguments.

result

\$resultwill be set to the number of user-defined arguments.

argumentclear

Delete all arguments in all modules.

arguments

This command has no arguments.

result

This command does not set any result variables.

1.4.13 Analysis

This section contains commands that are used for analysis.

Contents:

analyse/analyze/anal

Do function analysis.

arguments

This command has no arguments.

result

This command does not set any result variables.

exanalyse/exanalyze/exanal

Do exception directory analysis. This kind of analysis doesn't work on 32-bit executables.

arguments

This command has no arguments.

results

This command does not set any result variables.

cfanalyze/cfanalyse/cfanal

Do control flow analysis in the module selected in the disassembly view.

arguments

This command has no arguments.

results

This command does not set any result variables.

analyse_nukem/analyze_nukem/anal_nukem

Do function analysis using nukem's algorithm.

arguments

This command has no arguments.

result

This command does not set any result variables.

analxrefs/analx

Do xrefs analysis in the module selected in the disassembly view.

arguments

This command has no arguments.

results

This command does not set any result variables.

analrecur/analr

Do single function analysis.

arguments

arg1 The base address of the function to analyze.

results

This command does not set any result variables.

analadv

Do function analysis, embedded data analysis and xref analysis.

arguments

This command has no arguments.

results

This command does not set any result variables.

virtualmod

Tell the debugger to treat a memory range as a virtual module.

arguments

arg1 the user-supplied module name.

arg2 the base of the memory range.

[arg3] the size of the memory range.

result

This command does not set any result variables.

symdownload/downloadsym

Attempt to download a symbol from a Symbol Store.

arguments

[arg1] Module name (with or without extension) to attempt to download symbols for. When not specified, an attempt will be done to download symbols for all loaded modules.

[arg2] Symbol Store URL. When not specified, the default store will be used.

result

This command does not set any result variables.

imageinfo

Output the image information for a module. The information describes the Characteristics and DLL Characteristics fields in the PE header structure.

arguments

[arg1] The base of the module. If not specified the module at CIP will be used.

results

This command does not set any result variables.

GetRelocSize/grs

Get the correct size of a relocation table. This is useful while unpacking and restoring the original relocation table.

arguments

arg1 The address of the relocation table to analyze.

results

The found size of the relocation table is stored in `$result`.

exhandlers

Print all exception handlers, including SEH(StructuredExceptionHandler), VEH(VectoredExceptionHandler), VCH(VectoredContinueHandler) and UnhandledExceptionFilter, into the log.

arguments

This command has no arguments

results

This command does not set any result variables.

exinfo

Print the `EXCEPTION_DEBUG_INFO` structure from the last exception.

Sample output:

```

EXCEPTION_DEBUG_INFO:
    dwFirstChance: 1
    ExceptionCode: 80000001 (EXCEPTION_GUARD_PAGE)
    ExceptionFlags: 00000000
    ExceptionAddress: 00007FFE16FB1B91 ntdll.00007FFE16FB1B91
    NumberParameters: 2
ExceptionInformation[00]: 0000000000000008
ExceptionInformation[01]: 00007FFE16FB1B91 ntdll.00007FFE16FB1B91

```

arguments

This command has no arguments

results

This command does not set any result variables.

traceexecute

Tell the debugger that an address has been traced.

arguments

`arg1` The address.

result

This command does not set any result variables.

1.4.14 Types

This section contains commands that are used to manipulate data types.

Contents:

DataUnknown

Mark data at address as Unknown.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataByte/db

Mark data at address as Byte.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataWord/dw

Mark data at address as Word.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataDword/dw

Mark data at address as Dword.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataFword

Mark data at address as Fword.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataQword/dq

Mark data at address as Qword.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataTbyte

Mark data at address as Tbyte.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataOword

Mark data at address as Oword.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataMmword

Mark data at address as Mmword.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataXmmword

Mark data at address as Xmmword.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataYmmword

Mark data at address as Ymmword.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataFloat/DataReal4/df

Mark data at address as Float.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataDouble/DataReal8

Mark data at address as Double.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataLongdouble/DataReal10

Mark data at address as Long double.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataAscii/da

Mark data at address as Ascii.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataUnicode/du

Mark data at address as Unicode.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataCode/dc

Mark data at address as Code.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataJunk

Mark data at address as Junk.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

DataMiddle

Mark data at address as Middle.

arguments

arg1 The address you want to mark.

[arg2] Size (in bytes) to mark, when not set this defaults to 1.

result

This command does not set any result variables.

AddType

Add a type alias.

arguments

arg1 An existing type.

arg2 The new type alias.

result

This command does not set any result variables.

AddStruct

Add a new struct.

arguments

arg1 The type name of the struct.

result

This command does not set any result variables.

AddUnion

Add a new union.

arguments

arg1 The type name of the union.

result

This command does not set any result variables.

AddMember

Add a new member to the end of a struct/union.

arguments

arg1 The type name of the struct/union (parent).

arg2 The type of the new member.

arg3 The name of the new member.

[arg4] The array size. A value greater than zero will make this member an array.

[arg5] Offset from the start of the structure, only use this for implicitly padded structures. Overlapping with other members is **not** allowed.

result

This command does not set any result variables.

AppendMember

Add a new member to the end of the last manipulated struct/union.

arguments

arg1 The type of the new member.

arg2 The name of the new member.

[arg3] The array size. A value greater than zero will make this member an array.

[arg4] Offset from the start of the structure, only use this for implicitly padded structures. Overlapping with other members is **not** allowed.

result

This command does not set any result variables.

AddFunction

Add a new function.

arguments

arg1 The type name of the function.

arg2 The return type.

[arg3] Calling convention, choose between `cdecl`, `stdcall`, `thiscall` and `delphi`.

[arg4] Set to `nonzero` to mark this function as `noreturn`.

result

This command does not set any result variables.

AddArg

Add a new argument to the end of a function.

arguments

`arg1` The type name of the function (parent).

`arg2` The type of the new argument.

`arg3` The name of the new argument.

result

This command does not set any result variables.

AppendArg

Add a new argument to the end of the last manipulated function.

arguments

`arg1` The type of the new argument.

`arg2` The name of the new argument.

result

This command does not set any result variables.

SizeofType

Get the size of a type.

arguments

`arg1` Name of the type.

result

This command will set `$result` to the size of the type.

VisitType

Visit a type and print its members.

arguments

arg1 The type to visit.

[arg2] Address to print from. If not specified (or zero) the type will be printed without values.

[arg3] Maximum pointer resolution depth (default is 0). This can be used to also display structures (and values) pointed to by members of the type you are visiting.

result

This command does not set any result variables.

ClearTypes

Clear all types.

arguments

[arg1] The owner to clear. Leave this empty unless you know what you're doing.

result

This command does not set any result variables.

RemoveType

Remove a type.

arguments

arg1 The type to remove.

result

This command does not set any result variables.

EnumTypes

Enumerate all types.

arguments

This command has no arguments.

result

This command does not set any result variables.

LoadTypes

Load types from a JSON file.

arguments

`arg1` The path to the JSON file. The owner of the loaded types will be the filename of the JSON file. Any types previously defined with this owner will be removed.

result

This command does not set any result variables.

ParseTypes

Parse and load types from a header file.

arguments

`arg1` The path to the header file. The owner of the loaded types will be the filename of the header file. Any types previously defined with this owner will be removed.

result

This command does not set any result variables.

1.4.15 Plugins

This section contains debugger-embedded plugin commands.

Contents:

StartScylla/scylla/imprec

Start the Scylla plugin auto-selecting the currently debugged DLL/EXE and EIP/RIP as entry point.

arguments

This command has no arguments.

result

This command does not set any result variables.

plugload/pluginload/loadplugin

Load a plugin.

arguments

arg1 Name of the plugin.

result

This command does not set any result variables.

plugunload/pluginunload/unloadplugin

Unload a plugin.

arguments

arg1 Name of the plugin.

result

This command does not set any result variables.

1.4.16 Script Commands

This section contains various commands that are only used or available in a scripting context. Commands that also exist in a non-scripting context have priority.

Contents:

call

A call works exactly the same as an unconditional branch, but it places it's address on the script stack.

arguments

arg1 The label to jump to.

result

This command does not set any result variables.

invalid

Invalid command to throw an error message. This command will halt the script execution.

arguments

This command has no arguments.

result

This command does not set any result variables.

error

Show an error message and terminate the script.

arguments

arg1 The error message to show.

result

This command does not set any result variables.

Jxx/IFxx

There are various branches that can react on the flags set by the `cmp` (and maybe other) command(s):

- unconditional branch - `jmp/goto*` branch if not equal - `jne/ifne (q)/jnz/ifnz`
- branch if equal - `je/ife (q)/jz/ifz`
- branch if smaller - `jb/ibf/jl/ifl`
- branch if bigger - `ja/ifa/jg/ifg`
- branch if bigger/equal - `jbe/ifbe (q)/jle/ifle (q)`
- branch if smaller/equal - `jae/ifae (q)/jge/ifge (q)`

arguments

arg1 The label to jump to.

result

This command does not set any result variables.

log

Put information in the log.

arguments

[arg1] Format string (see *String Formatting*). When not specified, a newline will be logged.

result

This command does not set any result variables.

msg

Display a message box.

arguments

arg1 Message box text. You can use *String Formatting* here.

result

This command does not set any result variables.

msgyn

Display a message box, asking the user to answer yes or no.

arguments

arg1 Message box text. You can use *String Formatting* here.

result

The `$result` variable will be set to 1 when the user answered yes. Otherwise it's set to 0.

pause

Halt the script execution. The user can resume the script after this command.

arguments

This command has no arguments.

result

This command does not set any result variables.

printstack[,logstack]

Print the stack trace in the log.

arguments

This command has no arguments.

result

This command does not set any result variables.

ret

When called without an entry on the stack, this command will end the script and set the script IP to the first line. When 'call' was executed before, ret will return from that call.

arguments

This command has no arguments.

result

This command does not set any result variables.

scriptload

Load a script file.

arguments

arg1 Script file to load.

result

This command does not set any result variables.

scriptdll/dllscript

Execute a script DLL.

arguments

arg1 The filename and path of the script DLL. If a full path is not provided x64dbg will look in the scripts directory for the DLL.

results

This command does not set any result variables. However, the script DLL may set any variable.

remarks

A script DLL is a DLL that exports either `AsyncStart ()` or `Start ()` function.

If the DLL exports `AsyncStart ()` function, then x64dbg will call this function on a separate thread. If the DLL exports `Start ()` function, then x64dbg will call this function on the current thread, blocking any further command execution until the script DLL finishes execution. If both `AsyncStart ()` and `Start ()` are exported, only `AsyncStart ()` will be executed. Any return value of `AsyncStart ()` and `Start ()` will not be used by x64dbg.

After `AsyncStart ()` or `Start ()` finishes, the script DLL will be unloaded from the process.

1.4.17 GUI

This section describes the commands that control various portions of the GUI.

Contents:

disasm/dis/d

Disassemble at a certain position.

arguments

[arg1] The address to disassemble at. When not specified, there will be disassembled at CIP.

result

This command does not set any result variables.

dump

Dump at a certain position.

arguments

arg1 The address to dump at.

result

This command does not set any result variables.

sdump

Dump the stack at a certain position.

arguments

[arg1] The address to dump at (must be inside the thread stack range). If not specified, `csp` will be used.

result

This command does not set any result variables.

memmapdump

Follow an address in the memory map.

arguments

arg1 The address to follow.

result

This command does not set any result variables.

graph

Graph the control flow of function in the graph view.

arguments

[arg1] The address of the function. The default value is EIP or RIP. [arg2] Options. If it contains “force” the graph will be reanalyzed, if it contains “silent” no messages will be printed on the console.

results

This command does not set any result variables.

guiupdateenable

Enables GUI update after `guiupdatedisable` is executed.

arguments

[arg0] If not 0, tells the debugger to update its GUI immediately.

result

This command does not set any result variables.

guiupdatedisable

Disable GUI update. This can speed up script execution.

arguments

This command has no arguments.

result

This command does not set any result variables.

setfreezestack

Set if the stack should be frozen.

arguments

arg1 '0' for unfrozen, '1' for frozen.

result

This command does not set any result variables.

refinit

Initialize reference view for command usage.

arguments

[arg1] The title of the new reference view. You can use *String Formatting* here.

result

This command does not set any result variables.

refadd

Add an entry to the reference view. You need to call 'refinit' before using refadd.

arguments

arg1 Address to put in the reference view.

arg2 Text to put after the address. You can use *String Formatting* here.

result

This command does not set any result variables.

reget

Retrieve the address of a reference.

arguments

arg1 Zero-based index of the reference address to retrieve.

result

The `$result` variable will be set to the address of the requested reference (zero on failure).

EnableLog/LogEnable

Enable the log output.

arguments

This command has no arguments.

results

This command does not set any result variables.

DisableLog/LogDisable

Disable the log output. New log messages will not be appended to the log view, but they will still be saved in the log file if log redirection is enabled in the log view.

arguments

This command has no arguments.

results

This command does not set any result variables.

ClearLog/cls/lc/lclr

Clear the log window.

arguments

This command has no arguments.

result

This command does not set any result variables.

AddFavouriteTool

Add a tool in the favourites menu.

Note: You can use %PID% in the path. This special placeholder will be replaced by the PID of the debuggee before the tool is launched.

arguments

arg1 The path of the tool.

[arg2] The optional description of the tool. When this is set, it is displayed in the menu instead of the full path.

results

This command does not set any result variables.

AddFavouriteCommand

Add a command in the favourites menu.

arguments

arg1 The command to add.

[arg2] The optional shortcut key for the command.

results

This command does not set any result variables.

AddFavouriteToolShortcut/SetFavouriteToolShortcut

Set the shortcut key for an existing favourite tool.

arguments

arg1 The full path of an existing favourite tool.

arg2 The shortcut key for it.

results

This command does not set any result variables.

FoldDisassembly

Fold the disassembly within the specified range.

arguments

arg1 The start address of the range.

arg2 The length of the range.

results

This command does not set any result variables.

1.4.18 Miscellaneous

This section contains all commands that do not directly fit in another section.

Contents:

chd

Change current directory (SetCurrentDirectory).

arguments

arg1 Path of a directory to change to.

result

This command does not set any result variables.

zzz/doSleep

Halt execution for some time (equivalent of calling kernel32.Sleep).

arguments

[arg1] Time (in milliseconds) to sleep. If not specified this is set to 100ms (0.1 second). Keep in mind that input is in hex per default so `Sleep 100` will actually sleep 256 milliseconds (use `Sleep .100` instead).

result

This command does not set any result variables.

HideDebugger/dbh/hide

Hide the debugger from (very) simple detection methods.

arguments

This command has no arguments.

result

This command does not set any result variables.

loadlib

Load a DLL into debugged program memory.

arguments

arg1 The name/path of the module to load.

result

The `$result` variable will be set to the address of the loaded library.

asm

Assemble an instruction.

arguments

`arg1` Address to place the assembled instruction at.

`arg2` Instruction text. You can use *String Formatting* here.

[`arg3`] When specified the remainder of the previous instruction will be filled with NOPs.

result

\$result will be set to the assembled instruction size. 0 on failure.

gpa

Get the address of an export inside a DLL.

arguments

`arg1` Export name.

[`arg2`] DLL name.

result

The \$resultvariable is set to the export address. When the export is not found, \$resultwill be set to 0.

setjit/jitset

Set the Just-In-Time Debugger in Windows. In WIN64 systems there are two JIT entries: one for a x32 debugger and other for a x64 debugger. In a WIN64 system when a x32 process crash: Windows attach the x32 debugger stored in the x32-JIT entry.

Important notes:

- Its possible change the x32-JIT entry from the x64 debugger (using the x32 arg).
- Its possible change the x64-JIT entry from the x32 debugger ONLY if the x32 debugger its running in a WIN64 System (using the x64 arg).

arguments

Without arguments: Set the current debugger as JIT.

`arg1`

1. *oldsave*: Set the current debugger as JIT and save the last JIT entry.
2. *restore*: Set the old JIT entry stored as JIT and remove it from debugger db.
3. *old* (without `arg2`): Set the old JIT entry stored as new JIT.
4. *old* (with `arg2`): Set the `arg2` as old JIT entry stored.
5. *x32*: Set the `arg2` as new x32-JIT entry.

6. *x64*: Set the *arg2* as new x64-JIT entry.

result

This command does not set any result variables.

getjit/jitget

Get the Just-In-Time Debugger in Windows. In WIN64 systems there are two JIT entries: one for a x32 debugger and other for a x64 debugger. In a WIN64 system when a x32 process crash: Windows attach the x32 debugger stored in the x32-JIT entry.

Important notes:

- Its possible get the x32-JIT entry from the x64 debugger (using the x32 arg).
- Its possible get the x64-JIT entry from the x32 debugger ONLY if the x32 debugger its running in a WIN64 System (using the x64 arg).

arguments

Without arguments: Get the current JIT debugger.

arg2

1. *old*: Get the old JIT entry stored.
2. *x32*: Get the x32-JIT entry.*x64*: Get the x64-JIT entry.

result

This command does not set any result variables.

getjitauto/jitgetauto

Get the Auto Just-In-Time Debugger FLAG in Windows. if this flag value its TRUE Windows runs the debugger without user confirmation when a process crash. In WIN64 systems there are two JIT AUTO FLAG entries: one for a x32 debugger and other for a x64 debugger. In a WIN64 system when a x32 process crash with AUTO FLAG = FALSE: Windows confirm before attach the x32 debugger stored in the x32-JIT entry.

Important notes:

- Its possible get the x32-JIT AUTO FLAG entry from the x64 debugger (using the x32 arg).
- Its possible get the x64-JIT AUTO FLAG entry from the x32 debugger ONLY if the x32 debugger its running in a WIN64 System (using the x64 arg).

arguments

without args: Get current JIT entry FLAG.

arg1

1. *x32*: Get the x32-JIT AUTO FLAG.

2. x64: Get the x64-JIT AUTO FLAG.

result

This command does not set any result variables.

setjitauto/jitsetauto

Set the Auto Just-In-Time Debugger FLAG in Windows. if this flag value its TRUE Windows runs the debugger without user confirmation when a process crash. In WIN64 systems there are two JIT AUTO FLAG entries: one for a x32 debugger and other for a x64 debugger. In a WIN64 system when a x32 process crash with AUTO FLAG = FALSE: Windows confirm before attach the x32 debugger stored in the x32-JIT entry.

Important notes:

- Its possible set the x32-JIT AUTO FLAG entry from the x64 debugger (using the x32 arg).
- Its possible set the x64-JIT AUTO FLAG entry from the x32 debugger ONLY if the x32 debugger its running in a WIN64 System (using the x64 arg).

arguments

arg1

1. 1/ON: Set current JIT entry FLAG as TRUE.
2. 0/FALSE: Set current JIT entry FLAG as FALSE.
3. x32: Set the x32-JIT AUTO FLAG TRUE or FALSE. It needs an arg2: can be ON/1 or OFF/0.
4. x64: Set the x64-JIT AUTO FLAG TRUE or FALSE. It needs an arg2: can be ON/1 or OFF/0.

result

This command does not set any result variables.

getcommandline/getcmdline

It gets the actual command line.

arguments

This command has no arguments.

result

This command does not set any result variables.

setcommandline/setcmdline

It changes the command line data.

arguments

arg1 New command line.

result

This command does not set any result variables.

mnemonichelp

Output the detailed help information about an assembly mnemonic to the log.

arguments

arg1 the mnemonic name

result

This command does not set any result variables.

mnemonicbrief

Output the brief help information about an assembly mnemonic to the log.

arguments

arg1 the mnemonic name

result

This command does not set any result variables.

config

Get or set the configuration of x64dbg. It can also be used to load and store configuration specific to the script in the configuration file of x64dbg.

arguments

arg1 Section name of the INI file.

arg2 Key name of the INI file.

[arg3] Optional new value of the configuration. If this argument is set to a number, it will be stored in the configuration file and `$result` is not updated. If this argument is not set, the current configuration will be read into `$result`.

results

This command sets `$result` to the current configuration number if `arg3` is not set.

1.5 Developers

This section contains documentation intended to help developers to write code related to this program.

Contents:

1.5.1 Plugins

This section describes various plugin functions for x64dbg.

You can install plugins by copying the `*.dp32` (x32 plugins) or `*.dp64` (x64 plugins) to the 'plugins' directory.

Contents:

The basics

This page covers the basic principles of plugin development for x64dbg. See the [plugin page](#) for example plugins and templates.

Exports

A plugin has at least one export. This export must be called `pluginit`. See the `PLUG_INITSTRUCT` and the plugin headers for more information. The other valid exports are:

`plugstop` called when the plugin is about to be unloaded. Remove all registered commands and callbacks here. Also clean up plugin data.

`plugsetup` Called when the plugin initialization was successful, here you can register menus and other GUI-related things.

`CB*` Instead of calling `_plugin_registercallback`, you can create a `CDECL` export which has the name of the callback. For example when you create an export called `CBMENUENTRY`, this will be registered as your callback for the event `CB_MENUENTRY`. Notice that you should not use an underscore in the export name.

`CBALLEVENTS` An export with the name `CBALLEVENTS` will get every event registered to it. This is done prior to registering optional other export names.

Definitions

Initialization exports.

```
extern "C" __declspec(dllexport) bool pluginit(PLUG_INITSTRUCT* initStruct);
extern "C" __declspec(dllexport) bool plugstop();
extern "C" __declspec(dllexport) void plugsetup(PLUG_SETUPSTRUCT* setupStruct);
```

Callback exports. **Make sure to only export callbacks that you actually use!**

```
extern "C" __declspec(dllexport) void CBINITDEBUG(CBTYPE cbType, PLUG_CB_INITDEBUG*
↳info);
extern "C" __declspec(dllexport) void CBSTOPDEBUG(CBTYPE cbType, PLUG_CB_STOPDEBUG*
↳info);
extern "C" __declspec(dllexport) void CBEXCEPTION(CBTYPE cbType, PLUG_CB_EXCEPTION*
↳info);
extern "C" __declspec(dllexport) void CBDEBUGEVENT(CBTYPE cbType, PLUG_CB_DEBUGEVENT*
↳info);
extern "C" __declspec(dllexport) void CBMENUENTRY(CBTYPE cbType, PLUG_CB_MENUENTRY*
↳info);
```

Notes

The following coding guide are helpful to make your plugin more complaint.

Character encoding

x64dbg uses UTF-8 encoding everywhere it accepts a string. If you are passing a string to x64dbg, ensure that it is converted to UTF-8 encoding. This will help to reduce encoding errors.

Functions

This section contains information about the *_plugin_* prefixed functions exported by x64dbg.

Contents:

_plugin_debugpause

This function returns debugger control to the user. You would use this function when you write an unpacker that needs support from x64dbg (for example in development). Calling this function will set the debug state to 'paused' and it will not return until the user runs the debuggee using the *run* command .

```
void _plugin_debugpause();
```

Parameters

This function has no parameters.

Return Values

This function does not return a value.

_plugin_debugskipexceptions

This function returns sets if the debugger should skip first-chance exceptions. This is useful when creating unpackers or other plugins that run the debuggee.

```
void _plugin_debugskipexceptions(  
bool skip //skip flag  
);
```

Parameters

skip Flag if we need to skip first-chance exceptions or not.

Return Values

This function does not return a value.

_plugin_logprintf

This function prints a message to the log window.

```
void _plugin_logprintf(  
const char* format, //format string  
... //additional arguments  
);
```

Parameters

format Format string that has the same specifications as printf.

... Additional arguments (when needed by the format string).

Return Values

This function does not return a value.

_plugin_logputs

This function prints a single line to the log window.

```
void _plugin_logputs(  
const char* text //text to print  
);
```

Parameters

text Piece of text to put to the log window. This text can contain line breaks.

Return Values

This function does not return a value.

`_plugin_menuadd`

This function adds a new child menu to a menu.

```
int _plugin_menuadd(
int hMenu, //menu handle to add the new child menu to
const char* title //child menu title
);
```

Parameters

hMenu Menu handle from a previously-added child menu or from the main plugin menu.

title Menu title.

Return Values

Returns the child menu handle (unique), -1 on failure.

`_plugin_menuaddentry`

This function adds a menu entry to a menu.

```
bool _plugin_menuaddentry(
int hMenu, //menu handle to add the new child menu to
int hEntry, //plugin-wide identifier for the menu entry
const char* title //menu entry title
);
```

Parameters

hMenu Menu handle from a previously-added child menu or from the main plugin menu.

hEntry A plugin-wide identifier for the menu entry. This is the value you will get in the `PLUG_CB_MENUENTRY` callback structure.

title Menu entry title.

Return Values

Returns *true* when the entry was added without problems, *false* otherwise.

`_plugin_menuaddseparator`

This function adds a new child menu to a menu.

```
bool _plugin_menuaddseparator(
int hMenu //menu handle to add the separator to
);
```

Parameters

hMenu Menu handle from a previously-added child menu or from the main plugin menu.

Return Values

Returns *true* on success.

`_plugin_menuclear`

This function removes all entries and child menus from a menu. It will not remove the menu itself.

```
bool _plugin_menuclear (  
int hMenu //menu handle of the menu to clear  
);
```

Parameters

hMenu Menu handle from a previously-added child menu or from the main plugin menu.

Return Values

Returns *true* on success.

`_plugin_menuentryseticon`

This function sets an icon to a menu entry.

```
void _plugin_menuentryseticon (  
int pluginHandle, //plugin handle  
int hEntry, //handle of the menu entry  
const ICONDATA* icon //icon data  
);
```

Parameters

pluginHandle Handle of the calling plugin.

hEntry Menu handle from a previously-added child menu or from the main plugin menu.

icon Icon data. See *bridgemain.h* for a definition.

Return Values

This function does not return a value.

`_plugin_menuentrysetchecked`

This function sets the checked state of a menu entry. Notice that this function sets a menu item as checkable and thus it will toggle per default on click. If you want different behavior, make sure to call this function on every click with your desired state.

```
void _plugin_menuentrysetchecked (
int pluginHandle, //plugin handle
int hEntry, //handle of the menu entry
bool checked //new checked state
);
```

Parameters

pluginHandle Handle of the calling plugin.

hEntry Menu handle from a previously-added child menu or from the main plugin menu.

checked New checked state.

Return Values

This function does not return a value.

`_plugin_menuseticon`

This function sets an icon to a menu.

```
void _plugin_menuseticon (
int hMenu, //handle of the menu
const ICONDATA* icon //icon data
);
```

Parameters

hMenu Menu handle from a previously-added child menu or from the main plugin menu.

icon Icon data. See `bridgemain.h` for a definition.

Return Values

This function does not return a value.

`_plugin_registercallback`

This function registers an event callback for a plugin. Every plugin can have it's own callbacks for every event. It is not possible to have multiple callbacks on the same event.

```
void _plugin_registercallback(  
int pluginHandle, //plugin handle  
CBTYPE cbType, //event type  
CBPLUGIN cbPlugin //callback function  
);
```

Parameters

pluginHandle Handle of the calling plugin.

cbType The event type. This can be any of the following values:

- CB_INITDEBUG** , //callbackInfo: PLUG_CB_INITDEBUG*
- CB_STOPDEBUG** , //callbackInfo: PLUG_CB_STOPDEBUG*
- CB_CREATEPROCESS** , //callbackInfo: PLUG_CB_CREATEPROCESS*
- CB_EXITPROCESS** , //callbackInfo: PLUG_CB_EXITPROCESS*
- CB_CREATETHREAD** , //callbackInfo: PLUG_CB_CREATETHREAD*
- CB_EXITTHREAD** , //callbackInfo: PLUG_CB_EXITTHREAD*
- CB_SYSTEMBREAKPOINT** , //callbackInfo: PLUG_CB_SYSTEMBREAKPOINT*
- CB_LOADDLL** , //callbackInfo: PLUG_CB_LOADDLL*
- CB_UNLOADDLL** , //callbackInfo: PLUG_CB_UNLOADDLL*
- CB_OUTPUTDEBUGSTRING** , //callbackInfo: PLUG_CB_OUTPUTDEBUGSTRING*
- CB_EXCEPTION** , //callbackInfo: PLUG_CB_EXCEPTION*
- CB_BREAKPOINT** , //callbackInfo: PLUG_CB_BREAKPOINT*
- CB_PAUSEDDEBUG** , //callbackInfo: PLUG_CB_PAUSEDDEBUG*
- CB_RESUMEDDEBUG** , //callbackInfo: PLUG_CB_RESUMEDDEBUG*
- CB_STEPPED** , //callbackInfo: PLUG_CB_STEPPED*
- CB_ATTACH** , //callbackInfo: PLUG_CB_ATTACHED*
- CB_DETACH** , //callbackInfo: PLUG_CB_DETACHED*
- CB_DEBUGEVENT** , //callbackInfo: PLUG_CB_DEBUGEVENT*
- CB_MENUENTRY** , //callbackInfo: PLUG_CB_MENUENTRY*
- CB_WINEVENT** , //callbackInfo: PLUG_CB_WINEVENT*
- CB_WINEVENTGLOBAL** , //callbackInfo: PLUG_CB_WINEVENTGLOBAL*
- CB_LOADDB** , //callbackInfo: PLUG_CB_LOADSAVEDB*
- CB_SAVEDB** , //callbackInfo: PLUG_CB_LOADSAVEDB*
- CB_FILTERSYMBOL** , //callbackInfo: PLUG_CB_FILTERSYMBOL*
- CB_TRACEEXECUTE** , //callbackInfo: PLUG_CB_TRACEEXECUTE*

cbPlugin Callback with the following typedef:

```
void CBPLUGIN(
CBTYPE bType //event type (useful when you use the same function for multiple events
void* callbackInfo //pointer to a structure of information (see above)
);
```

Return Values

This function does not return a value.

`_plugin_registercommand`

This function registers a command for usage inside scripts or the command bar.

```
bool _plugin_registercommand(
int pluginHandle, //plugin handle
const char* command, //command name
CBPLUGINCOMMAND cbCommand, //function that is called when the command is executed
bool debugonly //restrict the command to debug-only
);
```

Parameters

pluginHandle Handle of the calling plugin.

command Command name.

cbCommand Callback with the following typedef:

```
bool CBPLUGINCOMMAND(
int argc //argument count (number of arguments + 1)
char* argv[] //array of arguments (argv[0] is the full command, arguments start at
↳argv[1])
);
```

debugonly When set, the command will never be executed when there is no target is being debugged.

Return Values

This function returns true when the command was successfully registered, make sure to check this, other plugins may have already registered the same command.

`_plugin_unregistercallback`

This plugin unregisters a previously set callback. It is only possible to remove callbacks that were previously set using `_plugin_registercallback`.

```
bool _plugin_unregistercallback(
int pluginHandle, //plugin handle
CBTYPE cbType //callback type to remove
);
```

Parameters

pluginHandle Handle of the calling plugin.

cbType The event type. This can be any of the following values:

- CB_INITDEBUG,
- CB_STOPDEBUG,
- CB_CREATEPROCESS,
- CB_EXITPROCESS,
- CB_CREATETHREAD,
- CB_EXITTHREAD,
- CB_SYSTEMBREAKPOINT,
- CB_LOADDLL,
- CB_UNLOADDLL,
- CB_OUTPUTDEBUGSTRING,
- CB_EXCEPTION,
- CB_BREAKPOINT,
- CB_PAUSEDEBUG,
- CB_RESUMEDEBUG,
- CB_STEPPED,
- CB_ATTACH,
- CB_DETACH,
- CB_DEBUGEVENT,
- CB_MENUENTRY,
- CB_WINEVENT,
- CB_WINEVENTGLOBAL

Return Values

This function returns true when the callback was removed without problems.

plugin_unregistercommand

This function removes a command set by a plugin. It is only possible to remove commands that you previously registered using *plugin_registercommand*.

```
bool plugin_unregistercommand(  
int pluginHandle, //plugin handle  
const char* command //command name  
);
```

Parameters

pluginHandle Handle of the calling plugin.

command Command name.

Return Values

This function returns true when the callback was removed without problems.

`_plugin_waituntilpaused`

Wait until the debugger paused.

```
bool _plugin_waituntilpaused();
```

Parameters

This function does not have any arguments.

Return Values

This command returns true if the debuggee is still active, returns false if the debuggee has stopped running.

`_plugin_hash`

This function allows you to hash some data. It is used by x64dbg in various places.

```
bool _plugin_hash(  
const void* data, //data to hash  
duint size //size (in bytes) of the data to hash  
);
```

Parameters

data Data to hash

size Size (in bytes) of the data to hash.

Return Values

Returns the hash.

Structures

This section describes the various plugin SDK structures.

Contents:

PLUG_INITSTRUCT

This structure is used by the only **needed** export in the plugin interface:

```
struct PLUG_INITSTRUCT
{
    //data provided by the debugger to the plugin.
    [IN] int pluginHandle; //handle of the plugin

    //data provided by the plugin to the debugger (required).
    [OUT] int sdkVersion; //plugin SDK version, use the PLUG_SDKVERSION define for
↪this
    [OUT] int pluginVersion; //plugin version, useful for crash reports
    [OUT] char pluginName[256]; //plugin name, also useful for crash reports
};
```

PLUG_SETUPSTRUCT

This structure is used by the function that allows the creation of plugin menu entries:

```
struct PLUG_SETUPSTRUCT
{
    //data provided by the debugger to the plugin.
    [IN] HWND hwndDlg; //GUI window handle
    [IN] int hMenu; //plugin menu handle
    [IN] int hMenuDisasm; //plugin disasm menu handle
    [IN] int hMenuDump; //plugin dump menu handle
    [IN] int hMenuStack; //plugin stack menu handle
};
```

Callback Structures

This section describes the various plugin SDK callback structures.

These structures are used inside event callbacks (registered using `_plugin_registercallback`).

Notice that the pointer 'void* callbackInfo' is never NULL, but the members of the various structures can be NULL.

Also remember that you cannot use any of the provided pointers out of the callback function scope.

In general AVOID time-consuming operations inside callbacks, do these in separate threads.

You can register a callback with `_plugin_registercallback` function. The type definition for the callbacks is:

```
void CBPLUGIN(
CBTYPE bType //event type (useful when you use the same function for multiple events
void* callbackInfo //pointer to a structure of information (see below)
);
```

Contents:

PLUG_CB_INITDEBUG

Called on debug initialization, useful to initialize some variables:

```
struct PLUG_CB_INITDEBUG
{
    const char* szFileName;
};
```

PLUG_CB_STOPDEBUG

Called when the debugging has been stopped, useful to reset some variables:

```
struct PLUG_CB_STOPDEBUG
{
    void* reserved;
};
```

PLUG_CB_CREATEPROCESS

Called after process creation (in the debug loop), after the initialization of the symbol handler, the database file and setting breakpoints on TLS callbacks / the entry breakpoint:

```
struct PLUG_CB_CREATEPROCESS
{
    CREATE_PROCESS_DEBUG_INFO* CreateProcessInfo;
    IMAGEHLP_MODULE64* modInfo;
    const char* DebugFileName;
    PROCESS_INFORMATION* fdProcessInfo;
};
```

PLUG_CB_EXITPROCESS

Called after the process exits (in the debug loop), before the symbol handler is cleaned up:

```
struct PLUG_CB_EXITPROCESS
{
    EXIT_PROCESS_DEBUG_INFO* ExitProcess;
};
```

PLUG_CB_CREATETHREAD

Called after thread creation (in the debug loop), after adding the thread to the internal thread list, before breaking the debugger on thread creation and after setting breakpoints on the thread entry:

```
struct PLUG_CB_CREATETHREAD
{
    CREATE_THREAD_DEBUG_INFO* CreateThread;
    DWORD dwThreadId;
};
```

PLUG_CB_EXITTHREAD

Called after thread termination (in the debug loop), before the thread is removed from the internal thread list, before breaking on thread termination:

```
struct PLUG_CB_EXITTHREAD
{
    EXIT_THREAD_DEBUG_INFO* ExitThread;
    DWORD dwThreadId;
};
```

PLUG_CB_SYSTEMBREAKPOINT

Called at the system breakpoint (in the debug loop), after setting the initial dump location, before breaking the debugger on the system breakpoint:

```
struct PLUG_CB_SYSTEMBREAKPOINT
{
    void* reserved;
};
```

PLUG_CB_LOADDLL

Called on DLL loading (in the debug loop), after the DLL has been added to the internal library list, after setting the DLL entry breakpoint:

```
struct PLUG_CB_LOADDLL
{
    LOAD_DLL_DEBUG_INFO* LoadDll;
    IMAGEHLP_MODULE64* modInfo;
    const char* modname;
};
```

PLUG_CB_UNLOADDLL

Called on DLL unloading (in the debug loop), before removing the DLL from the internal library list, before breaking on DLL unloading:

```
struct PLUG_CB_UNLOADDLL
{
    UNLOAD_DLL_DEBUG_INFO* UnloadDll;
};
```

PLUG_CB_OUTPUTDEBUGSTRING

Called on a DebugString event (in the debug loop), before dumping the string to the log, before breaking on a debug string:


```
struct PLUG_CB_OUTPUTDEBUGSTRING
{
    OUTPUT_DEBUG_STRING_INFO* DebugString;
};
```

PLUG_CB_EXCEPTION

Called on an unhandled (by the debugger) exception (in the debug loop), after setting the continue status, after locking the debugger to pause:

```
struct PLUG_CB_EXCEPTION
{
    EXCEPTION_DEBUG_INFO* Exception;
};
```

PLUG_CB_BREAKPOINT

Called on a normal/memory/hardware breakpoint (in the debug loop), after locking the debugger to pause:

```
struct PLUG_CB_BREAKPOINT
{
    BRIDGEBP* breakpoint;
};
```

PLUG_CB_PAUSEDEBUG

Called after the debugger has been locked to pause (in the debug loop), before any other callback that's before pausing the debugger:

```
struct PLUG_CB_PAUSEDEBUG
{
    void* reserved;
};
```

PLUG_CB_RESUMEDEBUG

Called after the debugger has been unlocked to resume (outside of the debug loop):

```
struct PLUG_CB_RESUMEDEBUG
{
    void* reserved;
};
```

PLUG_CB_STEPPED

Called after the debugger stepped (in the debug loop), after locking the debugger to pause:

```
struct PLUG_CB_STEPPED
{
    void* reserved;
};
```

PLUG_CB_ATTACH

Called before attaching to a process:

```
struct PLUG_CB_ATTACH
{
    DWORD dwProcessId;
};
```

PLUG_CB_DETACH

Called before detaching from the process:

```
struct PLUG_CB_DETACH
{
    PROCESS_INFORMATION* fdProcessInfo;
};
```

PLUG_CB_DEBUGEVENT

Called on any debug event, even the ones that are handled internally.

Avoid doing stuff that takes time here, this will slow the debugger down a lot!:

```
struct PLUG_CB_DEBUGEVENT
{
    DEBUG_EVENT* DebugEvent;
};
```

PLUG_CB_MENUENTRY

Called when a menu entry created by the plugin has been clicked, the GUI will resume when this callback returns:

```
struct PLUG_CB_MENUENTRY
{
    int hEntry;
};
```

PLUG_CB_WINEVENT

Called before TranslateMessage and DispatchMessage Windows functions (PreTranslateMessage).

Avoid calling user32 functions without precautions here, there will be a recursive call if you fail to take countermeasures:

```

struct PLUG_CB_WINEVENT
{
    MSG* message;
    long* result;
    bool retval; //only set this to true if you want Qt to ignore the event.
};

```

PLUG_CB_WINEVENTGLOBAL

Called before TranslateMessage and DispatchMessage Windows functions (PreTranslateMessage).

Avoid calling user32 functions without precautions here, there will be a recursive call if you fail to take countermeasures.

This function is global, so it also captures hotkeys (see Qt documentation). In Qt5 this function is almost never called, use *PLUG_CB_WINEVENT* instead.

```

struct PLUG_CB_WINEVENTGLOBAL
{
    MSG* message;
    bool retval; //only set this to true if you want Qt to ignore the event.
};

```

PLUG_CB_LOADSAVEDB

Load or save data to database. Data is retrieved or stored or retrieved in a JSON format:

Two constants are defined in the `_plugins.h` file for the loadSaveType:

`PLUG_DB_LOADSAVE_DATA` `PLUG_DB_LOADSAVE_ALL`

```

struct PLUG_CB_LOADSAVEDB
{
    json_t* root;
    int loadSaveType;
};

```

PLUG_CB_FILTERSYMBOL

Called before a symbol is emitted to the automatic labels. Set *retval* to *false* if you want to filter the symbol.

```

struct PLUG_CB_FILTERSYMBOL
{
    const char* symbol;
    bool retval;
};

```

PLUG_CB_TRACEEXECUTE

Called during conditional tracing. Set the *stop* member to *true* to stop tracing.

```
struct PLUG_CB_TRACEEXECUTE
{
    duint cip;
    bool stop;
};
```

1.5.2 Functions

This section describes functions in the bridge. Many are currently undocumented, help if you can!

Contents:

Bridge Functions

This section contains information about the Bridge functions of x64dbg.

Please note: Bridge functions are handled by x64dbg and should not normally be called by any third party program or plugin - they are included for documentation purposes.

Contents:

BridgeAlloc

Function description.

```
void* BridgeAlloc(
    size_t size // memory size to allocate
);
```

Parameters

size Memory size (in bytes) to allocate.

Return Value

Returns a pointer to the memory block allocated. If an error occurs allocating memory, then x64dbg is closed down.

Example

```
auto ptr = (char*)BridgeAlloc(128);
//do something with ptr
BridgeFree(ptr);
```

Related functions

- *BridgeFree*

BridgeFree

Function description.

```
void BridgeFree(  
    void* ptr // pointer to memory block to free  
);
```

Parameters

`ptr` Pointer to memory block to free

Return Value

This function does not return a value.

Example

```
auto ptr = (char*)BridgeAlloc(128);  
//do something with ptr  
BridgeFree(ptr);
```

Related functions

- *BridgeAlloc*

BridgeGetDbgVersion

Internal function, don't use!

```
int BridgeGetDbgVersion();
```

Parameters

This function has no parameters.

Return Value

Return an integer value representing the version of the x64dbg.

Example

```
int version = BridgeGetDbgVersion();
```

Related functions

- List of related functions

BridgeInit

Initializes the Bridge, defines the .ini file used for x64dbg and loads the main GUI and Debug functions. **Internal function, don't use!**

```
const wchar_t* BridgeInit();
```

Parameters

This function has no parameters.

Return Value

Returns 0 if successful, otherwise a string indicating the error that occurred.

Example

```
Example code.
```

Related functions

- List of related functions

BridgeSettingFlush

Flush the settings stored in memory to disk.

```
bool BridgeSettingFlush();
```

Parameters

This function has no parameters.

Return Value

This function returns true if successful or false otherwise.

Example

Example code.

Related functions

- *BridgeSettingGet*
- *BridgeSettingGetUint*
- *BridgeSettingSet*
- *BridgeSettingSetUint*
- *BridgeSettingRead*

BridgeSettingGet

Reads a string file from the settings.

```
bool BridgeSettingGet (
    const char* section, // ini section name to read
    const char* key, // ini key in the section to read
    char* value // string to hold the value read
);
```

Parameters

section Section name to read.

key Key in the section to read.

value Destination buffer. Should be of MAX_SETTING_SIZE.

Return Value

This function returns true if successful or false otherwise.

Example

Example code.

Related functions

- List of related functions

BridgeSettingGetUint

Reads an integer from the settings.

```
bool BridgeSettingGetUint(  
    const char* section, // ini section name to write to  
    const char* key, // ini key in the section to write  
    uint* value // an integer variable to hold the value read  
);
```

Parameters

section Section name to read.

key Key in the section to read.

value Destination value.

Return Value

This function returns true if successful or false otherwise.

Example

```
Example code.
```

Related functions

- List of related functions

BridgeSettingRead

Read the settings from disk (INI file).

```
bool BridgeSettingRead(  
    int* errorLine // line that error occurred on  
);
```

Parameters

errorLine Line that error occurred on.

Return Value

This function returns true if successful or false otherwise.

Example

Example code.

Related functions

- List of related functions

BridgeSettingSet

Writes a string value to the settings.

```
bool BridgeSettingSet(  
    const char* section, // ini section name to write to  
    const char* key, // ini key in the section to write  
    char* value // string value to write  
);
```

Parameters

section Section name to write to.

key Key in the section to write.

value New setting value.

Return Value

This function returns true if successful or false otherwise.

Example

Example code.

Related functions

- List of related functions

BridgeSettingSetUInt

Reads an integer from the settings.

```
bool BridgeSettingSetUInt(  
    const char* section, // ini section name to write to  
    const char* key, // ini key in the section to write  
    uint value // an integer variable to write  
);
```

Parameters

`section` Section name to read.

`key` Key in the section to read.

`value` New value.

Return Value

This function returns true if successful or false otherwise.

Example

```
Example code.
```

Related functions

- List of related functions

BridgeStart

Function description.

```
Function definition.
```

Parameters

`param1` Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

Debug Functions

This section contains information about debug functions of x64dbg.

Contents:

DbgArgumentAdd

This function will add an argument to the specified address range.

```
bool DbgArgumentAdd(duint start, duint end);
```

Parameters

start first address of the argument range.

end last address of the argument range.

Return Value

The function return TRUE if argument is successfully setted or FALSE otherwise.

Example

```
if(DbgArgumentAdd(0x00401000, 0x00401013))
    GuiAddLogMessage("Argument successfully setted\r\n");
else
    GuiAddLogMessage("Argument couldn't be set\r\n");
```

Related functions

- DbgArgumentDel
- DbgArgumentGet
- DbgArgumentOverlaps

DbgArgumentDel

This function deletes a previous setted argument at the specified address.

```
bool DbgArgumentDel(duint addr);
```

Parameters

addr Address of the argument to delete.

Return Value

The function return TRUE if argument is successfully deleted or FALSE otherwise.

Example

```
if (DbgArgumentDel (0x00401013))
    GuiAddLogMessage ("Argument successfully deleted\r\n");
else
    GuiAddLogMessage ("Argument couldn't be deleted\r\n");
```

Related functions

- DbgArgumentAdd
- DbgArgumentGet
- DbgArgumentOverlaps

DbgArgumentGet

This function gets the boundaries of the given argument location as start and end addresses.

```
bool DbgArgumentGet (duint addr, duint* start, duint* end);
```

Parameters

`addr` Address of the argument to fetch.

`start` Pointer to a duint variable that will hold the start address of the argument.

`end` Pointer to a duint variable that will hold the end address of the argument.

Return Value

The function return TRUE if the start and end addresses are found or FALSE otherwise. If TRUE, the variables `start` and `end` will hold the fetched values.

Example

```
duint start;
duint end;
std::string message;

if (DbgArgumentGet (0x00401000, &start, &end))
{
    sprintf_s (message.c_str(), MAX_PATH, "Argument range: %08X-%08X\r\n", start, end);
    GuiAddLogMessage (message);
}
```

```
else
{
    GuiAddLogMessage("Argument start and end addresses couldn't be get\r\n");
}
```

Related functions

- DbgArgumentAdd
- DbgArgumentDel
- DbgArgumentOverlaps

DbgArgumentOverlaps

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgAssembleAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgClearAutoBookmarkRange

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgClearAutoCommentRange

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgClearAutoFunctionRange

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgClearAutoLabelRange

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgClearBookmarkRange

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgClearCommentRange

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgClearLabelRange

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgCmdExec

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgCmdExecDirect

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgDelEncodeTypeRange

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgDelEncodeTypeSegment

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgDisasmAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgDisasmFastAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgExit

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgFunctionAdd

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgFunctionDel

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgFunctionGet

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgFunctionOverlaps

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgFunctions

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetArgTypeAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetBookmarkAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetBpList

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetBpxTypeAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetBranchDestination

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetCommentAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetEncodeSizeAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetEncodeTypeAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetEncodeTypeBuffer

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetFunctionTypeAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetLabelAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetLoopTypeAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetModuleAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetRegDump

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetStringAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetThreadList

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetTimeWastedCounter

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetWatchList

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetXrefCountAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgGetXrefTypeAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgInit

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgIsBpDisabled

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgIsDebugging

Determines if the debugger is currently debugging an opened file or attached process.

```
bool DbgIsDebugging();
```

Parameters

This function has no parameters.

Return Value

This function returns true if x64dbg is currently debugging, or false otherwise.

Example

```

if(!DbgIsDebugging())
{
    GuiAddLogMessage("You need to be debugging to use this option!\n");
    return false;
}

```

```

.data
szMsg db "You need to be debugging to use this option!",13,10,0 ; CRLF

.code
Invoke DbgIsDebugging
.IF eax == FALSE
    Invoke GuiAddLogMessage, Addr szMsg
.ENDIF

```

Related functions

- *DbgIsRunning*

DbgIsJumpGoingToExecute

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgIsRunLocked

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgIsRunning

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgIsValidExpression

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgLoopAdd

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgLoopDel

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgLoopGet

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgLoopOverlaps

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgMemFindBaseAddr

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgMemGetPageSize

Function description.

Function definition.

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

Get page base and size of selected instruction.

```
SELECTIONDATA sel;           // Define Address the slected line in the Disassembly
↪window ( begin , End )
    GuiSelectionGet(GUI_DISASSEMBLY, &sel); // Get the value of sel(begin addr , End
↪addr )
    quint pagesize = DbgMemGetPageSize(sel.start); // get the page size of the
↪section from the selected memory addr
    //Or use the following statement to get page base and size in one call.
    quint sctionbase = DbgMemFindBaseAddr(sel.start, &pagesize); // get the base of
↪this section ( begin addr of the section )
```

Related functions

- List of related functions

DbgMemIsValidReadPtr

Function description.

Function definition.

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgMemMap

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgMemRead

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgMemWrite

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgMenuEntryClicked

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgModBaseFromName

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgReleaseEncodeTypeBuffer

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgScriptAbort

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgScriptBpGet

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgScriptBpToggle

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgScriptCmdExec

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgScriptGetBranchInfo

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgScriptGetLineType

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgScriptLoad

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgScriptRun

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgScriptSetIp

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgScriptStep

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgScriptUnload

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgSetAutoBookmarkAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgSetAutoCommentAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgSetAutoFunctionAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgSetAutoLabelAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgSetBookmarkAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgSetCommentAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgSetEncodeType

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgSetLabelAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgSettingsUpdated

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgStackCommentGet

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgSymbolEnum

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgSymbolEnumFromCache

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgValFromString

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgValToString

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgWinEvent

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgWinEventGlobal

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgXrefAdd

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgXrefDelAll

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

DbgXrefGet

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GUI Functions

This section contains information about the Graphical User Interface GUI functions of x64dbg.

Contents:

GuiAddLogMessage

Adds a message to the log. The message will be shown in the log window and on the status bar at the bottom of x64dbg.

```
void GuiAddLogMessage(  
    const char* msg // string containing message to add to log  
);
```

Parameters

msg String containing the message to add to the log. Ensure that a carriage line and return feed are included with the string for it to properly display it. Encoding is UTF-8.

Return Value

This function does not return a value.

Example

```
GuiAddLogMessage("This text will be displayed in the log view.\n");
```

```
.data  
szMsg db "This text will be displayed in the log view",13,10,0 ; CRLF  
  
.code  
Invoke GuiAddLogMessage, Addr szMsg
```

Related functions

- *GuiLogClear*
- *GuiAddStatusBarMessage*

GuiAddQWidgetTab

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiAddRecentFile

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiAddStatusBarMessage

Shows text in the statusbar, which can be used to inform the user.

```
void GuiAddStatusBarMessage(const char* msg);
```

Parameters

msg String containing the message to add to the status bar.

Return Value

This function does not return a value.

Example

```
GuiAddStatusBarMessage("This text will be displayed in the statusbar.");
```

Related functions

- *GuiAddLogMessage*

GuiAutoCompleteAddCmd

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiAutoCompleteClearAll

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiAutoCompleteDelCmd

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiCloseQWidgetTab

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiDisasmAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiDisplayWarning

Shows a warning dialog with title text and main message text.

```
void GuiDisplayWarning(const char* title, const char* text)
```

Parameters

param1 Parameter description.

Return Value

This function does not return a value.

Example

```
GuiDisplayWarning("Warning!", "Operation cannot be reversed.");
```

Related functions

- *GuiAddLogMessage*
- *GuiAddStatusBarMessage*

GuiDumpAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```


Related functions

- List of related functions

GuiDumpAtN

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiExecuteOnGuiThread

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiFocusView

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiGetDebuggeeNotes

Returns into a variable a pointer to a string containing notes and information that a user has made relating to the target being debugged (the debuggee). The function `GuiGetGlobalNotes` can be used to get the global notes stored by a user.

```
void GuiGetDebuggeeNotes(char** text)
```

Parameters

`text` A variable that will contain a pointer to a buffer on return. The pointer returned points to a string that will contain the notes for the debuggee.

Return Value

This function does not return a value. The string containing the notes is returned via the pointer supplied via the `text` parameter.

Example

```
char* text = nullptr;
GuiGetDebuggeeNotes(&text);
if(text)
{
    \\ do something with text
}
```

Related functions

- *GuiSetDebuggeeNotes*
- *GuiGetGlobalNotes*
- *GuiSetGlobalNotes*

GuiGetDisassembly

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiGetGlobalNotes

Returns into a variable a pointer to a string containing global notes and information that a user has made. The function `GuiGetDebuggeeNotes` can be used to get the local notes stored by a user for the target being debugged (the debuggee).

```
void GuiGetGlobalNotes(char** text)
```

Parameters

`text` A variable that will contain a pointer to a buffer on return. The pointer returned points to a string that will contain the global notes.

Return Value

This function does not return a value. The string containing the notes is returned via the pointer supplied via the `text` parameter.

Example

```
char* text = nullptr;
GuiGetGlobalNotes(&text);
if(text)
{
    \\ do something with text
}
```

Related functions

- *GuiSetGlobalNotes*
- *GuiGetDebuggeeNotes*
- *GuiSetDebuggeeNotes*

GuiGetLineWindow

Function description.

```
Function definition.
```

Parameters

`param1` Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiGetWindowHandle

Obtains the main window handle for x64dg.

```
HWND GuiGetWindowHandle();
```

Parameters

This function has no parameters.

Return Value

Returns the main window handle for x6dbg as a HWND variable

Example

```
hWnd = GuiGetWindowHandle();
```

Related functions

GuiGraphAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiUpdateDisabled

Returns the status of the internal update flag, which can be disabled via `GuiUpdateDisable` function or enabled via the `GuiUpdateEnable` function.

```
bool GuiIsUpdateDisabled()
```

Parameters

This function has no parameters.

Return Value

Returns a boolean value indicating if the internal update flag is set to disabled. If it is set to disabled the value is `TRUE` otherwise updates are enabled and the value is `FALSE`.

Example

```
bool bUpdate = GuiIsUpdateDisabled();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*

- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiLoadGraph

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiLoadSourceFile

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiLogClear

Clears the log window of all text.

```
void GuiLogClear();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiLogClear();
```

Related functions

- [GuiAddLogMessage](#)

GuiMenuAdd

This function adds a new child menu to a menu.

```
int GuiMenuAdd(int hMenu, const char* title)
```

Parameters

`hMenu` Menu handle from a previously-added menu or from the main menu.

`title` A const char representing the text title of the menu item to be added.

Return Value

Returns the menu handle (unique), or -1 on failure.

Example

```
hNewMenu = GuiMenuAdd(hMenu, &szMenuTitle);
```

Related functions

- *GuiMenuAddEntry*
- *GuiMenuAddSeparator*
- *GuiMenuClear*
- *GuiMenuSetEntryIcon*
- *GuiMenuSetIcon*

Note: Plugin developers should make use of the plugin functions provided:

- _
- _
- _
- _
- _
- _

GuiMenuAddEntry

This function adds a menu entry to a menu.

```
int GuiMenuAddEntry(int hMenu, const char* title)
```

Parameters

`hMenu` Menu handle from a previously-added menu or from the main menu.

`title` A const char representing the text title of the menu item to be added.

Return Value

Returns the menu handle (unique), or -1 on failure.

Example

```
hNewMenuEntry = GuiMenuAddEntry(hMenu, &szMenuEntryText);
```

Related functions

- *GuiMenuAdd*
- *GuiMenuAddSeparator*
- *GuiMenuClear*
- *GuiMenuSetEntryIcon*
- *GuiMenuSetIcon*

Note: Plugin developers should make use of the plugin functions provided:

- _
- _
- _
- _
- _
- _

GuiMenuAddSeparator

Adds a menu separator to a menu.

```
void GuiMenuAddSeparator(int hMenu)
```

Parameters

hMenu Menu handle from a previously-added menu or from the main menu.

Return Value

This function does not return a value.

Example

```
hNewMenu = GuiMenuAdd(hMenu, &szMenuTitle);  
GuiMenuAddEntry(hNewMenu, &szMenuEntry1Text);  
GuiMenuAddEntry(hNewMenu, &szMenuEntry2Text);  
GuiMenuAddSeparator(hNewMenu);  
GuiMenuAddEntry(hNewMenu, &szMenuEntry3Text);  
GuiMenuAddEntry(hNewMenu, &szMenuEntry4Text);
```

Related functions

- *GuiMenuAdd*
- *GuiMenuAddEntry*
- *GuiMenuClear*
- *GuiMenuSetEntryIcon*
- *GuiMenuSetIcon*

Note: Plugin developers should make use of the plugin functions provided:

- _
- _
- _
- _
- _
- _

GuiMenuClear

This function removes all entries and child menus from a menu. It will not remove the menu itself.

```
void GuiMenuClear (int hMenu)
```

Parameters

hMenu Menu handle from a previously-added menu or from the main menu.

Return Value

This function does not return a value.

Example

```
hNewMenu = GuiMenuAdd(hMenu, &szMenuTitle);
GuiMenuClear(hNewMenu);
```

Related functions

- *GuiMenuAdd*
- *GuiMenuAddEntry*
- *GuiMenuAddSeparator*
- *GuiMenuSetEntryIcon*

- *GuiMenuSetIcon*

Note: Plugin developers should make use of the plugin functions provided:

- _
- _
- _
- _
- _
- _

GuiMenuSetEntryIcon

Sets an icon for a specified menu entry.

```
void GuiMenuSetEntryIcon(int hEntry, const ICONDATA* icon)
```

Parameters

`hEntry` Parameter description.

`icon`

Return Value

This function does not return a value.

Example

```
ICONDATA rocket;  
rocket.data = icon_rocket;  
rocket.size = sizeof(icon_rocket);  
hNewMenuEntry = GuiMenuAddEntry(hMenu, &szMenuEntryText);  
GuiMenuSetEntryIcon(hNewMenuEntry, &rocket);
```

Related functions

- *GuiMenuAdd*
- *GuiMenuAddEntry*
- *GuiMenuAddSeparator*
- *GuiMenuClear*
- *GuiMenuSetIcon*

Note: Plugin developers should make use of the plugin functions provided:

- _

- _
- _
- _
- _
- _

GuiMenuSetIcon

Sets an icon for a specified menu.

```
void GuiMenuSetIcon(int hMenu, const ICONDATA* icon)
```

Parameters

hMenu Menu handle from a previously-added menu or from the main menu.

icon

Return Value

This function does not return a value.

Example

```
ICONDATA rocket;
rocket.data = icon_rocket;
rocket.size = sizeof(icon_rocket);
hNewMenuEntry = GuiMenuAddEntry(hMenu, &szMenuEntryText);
GuiMenuSetIcon(hMenuDisasm, &rocket);
```

Related functions

- *GuiMenuAdd*
- *GuiMenuAddEntry*
- *GuiMenuAddSeparator*
- *GuiMenuClear*
- *GuiMenuSetEntryIcon*

Note: Plugin developers should make use of the plugin functions provided:

- _
- _
- _
- _

- _
- _

GuiReferenceAddColumn

Adds a column to the current Reference View instance.

```
void GuiReferenceAddColumn(int width, const char* title);
```

Parameters

`width` An integer indicating the width of the column to add.

`title` A const char representing the column's title name to add.

Return Value

This function does not return a value.

Example

```
GuiReferenceAddColumn(8, &sztitle);
```

Related functions

- *GuiReferenceDeleteAllColumns*
- *GuiReferenceGetCellContent*
- *GuiReferenceGetRowCount*
- *GuiReferenceInitialize*
- *GuiReferenceReloadData*
- *GuiReferenceSetCellContent*
- *GuiReferenceSetCurrentTaskProgress*
- *GuiReferenceSetProgress*
- *GuiReferenceSetRowCount*
- *GuiReferenceSetSearchStartCol*
- *GuiReferenceSetSingleSelection*

GuiReferenceDeleteAllColumns

Removes all columns from the current Reference View instance.

```
void GuiReferenceDeleteAllColumns ();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiReferenceDeleteAllColumns ();
```

Related functions

- *GuiReferenceAddColumn*
- *GuiReferenceGetCellContent*
- *GuiReferenceGetRowCount*
- *GuiReferenceInitialize*
- *GuiReferenceReloadData*
- *GuiReferenceSetCellContent*
- *GuiReferenceSetCurrentTaskProgress*
- *GuiReferenceSetProgress*
- *GuiReferenceSetRowCount*
- *GuiReferenceSetSearchStartCol*
- *GuiReferenceSetSingleSelection*

GuiReferenceGetCellContent

Retrieves the data stored in a specified cell in the current Reference View instance, based on the supplied row and column parameters.

```
char* GuiReferenceGetCellContent (int row, int col)
```

Parameters

`row` An integer representing the row for the cell for which the data is fetched.

`col` An integer representing the column for the cell for which the data is fetched.

Return Value

The return value is a pointer to a char representing the data (typically a string) that was stored at the specified row/column of the current Reference View instance. NULL if there was no data or the row/column specified was incorrect.

Example

```
Data = GuiReferenceGetCellContent (0, 0);
```

Related functions

- *GuiReferenceAddColumn*
- *GuiReferenceDeleteAllColumns*
- *GuiReferenceGetRowCount*
- *GuiReferenceInitialize*
- *GuiReferenceReloadData*
- *GuiReferenceSetCellContent*
- *GuiReferenceSetCurrentTaskProgress*
- *GuiReferenceSetProgress*
- *GuiReferenceSetRowCount*
- *GuiReferenceSetSearchStartCol*
- *GuiReferenceSetSingleSelection*

GuiReferenceGetRowCount

Gets the total rows count in the current Reference View instance.

```
int GuiReferenceGetRowCount ()
```

Parameters

This function has no parameters.

Return Value

Returns an integer value representing the total rows in the current Reference View instance.

Example


```
int iTotalRows = GuiReferenceGetRowCount();
```

Related functions

- *GuiReferenceAddColumn*
- *GuiReferenceDeleteAllColumns*
- *GuiReferenceGetCellContent*
- *GuiReferenceInitialize*
- *GuiReferenceReloadData*
- *GuiReferenceSetCellContent*
- *GuiReferenceSetCurrentTaskProgress*
- *GuiReferenceSetProgress*
- *GuiReferenceSetRowCount*
- *GuiReferenceSetSearchStartCol*
- *GuiReferenceSetSingleSelection*

GuiReferenceInitialize

Initializes and creates a new instance of a Reference View. A new tab will appear under Reference Views entitled as per the name parameter. Columns can be added to this instance via the `GuiReferenceAddColumn` function. Rows can be inserted by using the `GuiReferenceSetRowCount` function and data for each cell (row and column) can be added via the `GuiReferenceSetCellContent` function. See the appropriate functions for more details.

```
void GuiReferenceInitialize(const char* name)
```

Parameters

name A const char representing the text string to name the Reference View instance.

Return Value

This function does not return a value.

Example

```
GuiReferenceInitialize("Code Caves");
```

Related functions

- *GuiReferenceAddColumn*
- *GuiReferenceDeleteAllColumns*
- *GuiReferenceGetCellContent*
- *GuiReferenceGetRowCount*
- *GuiReferenceReloadData*
- *GuiReferenceSetCellContent*
- *GuiReferenceSetCurrentTaskProgress*
- *GuiReferenceSetProgress*
- *GuiReferenceSetRowCount*
- *GuiReferenceSetSearchStartCol*
- *GuiReferenceSetSingleSelection*

GuiReferenceReloadData

Reloads (refreshes) the data in the current Reference View instance.

```
void GuiReferenceReloadData ();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiReferenceReloadData ();
```

Related functions

- *GuiReferenceAddColumn*
- *GuiReferenceDeleteAllColumns*
- *GuiReferenceGetCellContent*
- *GuiReferenceGetRowCount*
- *GuiReferenceInitialize*
- *GuiReferenceSetCellContent*

- *GuiReferenceSetCurrentTaskProgress*
- *GuiReferenceSetProgress*
- *GuiReferenceSetRowCount*
- *GuiReferenceSetSearchStartCol*
- *GuiReferenceSetSingleSelection*

GuiReferenceSetCellContent

Sets a specified cell's data content in the Reference View based on the supplied row and column parameters.

```
void GuiReferenceSetCellContent(int row, int col, const char* str);
```

Parameters

`row` integer representing the row to set data for.

`col` integer representing the column to set data for.

`str` `const char*` representing the string data to set at the row, col specified.

Return Value

This function does not return a value.

Example

```
const char szRefStart = "Start";
const char szRefFinish = "Finish";
const char szRefType = "Type";
GuiReferenceInitialize("Some Information"); // Add Reference View Header Title
GuiReferenceAddColumn(2 * sizeof(DWORD), &szRefStart); // Add column Name
GuiReferenceAddColumn(2 * sizeof(DWORD), &szRefFinish); // Add column Name
GuiReferenceAddColumn(8, &szRefType); // Add column Name
GuiReferenceSetRowCount(2); // add 2 rows
int iRow = 0;
GuiReferenceSetCellContent(iRow, 0, &szCodeCaveStartAddress); // add start address
GuiReferenceSetCellContent(iRow, 1, &szCodeCaveFinishAddress); // add finish address
GuiReferenceSetCellContent(iRow, 2, &szNop); // add type
iRow = iRow + 1; // Increment rows
// get variables to convert to strings (szCodeCaveStartAddress, ↵
↵szCodeCaveFinishAddress etc)
// add to next row's columns
```

Notes

The Reference View must be initialized beforehand, and any columns required added before adding any rows and setting data for them.

Ensure you increment the row counter after you have set all data for all columns in a particular row, otherwise you will just overwrite any data you have set previously.

`GuiReferenceSetRowCount` needs to be called before setting cell contents - to update the reference view with a total count of rows, for example to add 5 rows: `GuiReferenceSetRowCount(5)`, if you then decide to add another row later on then you would specify `GuiReferenceSetRowCount(6)`

Ideally you will use some variable that is incremented to automatically keep track of total rows added.

Related functions

- *GuiReferenceAddColumn*
- *GuiReferenceDeleteAllColumns*
- *GuiReferenceGetCellContent*
- *GuiReferenceGetRowCount*
- *GuiReferenceInitialize*
- *GuiReferenceReloadData*
- *GuiReferenceSetCurrentTaskProgress*
- *GuiReferenceSetProgress*
- *GuiReferenceSetRowCount*
- *GuiReferenceSetSearchStartCol*
- *GuiReferenceSetSingleSelection*

GuiReferenceSetCurrentTaskProgress

Sets the percentage bar and some status text in the current Reference View instance. This can indicate to the user that an operation is taking place, e.g. searching/sorting.

```
void GuiReferenceSetCurrentTaskProgress(int progress, const char* taskTitle);
```

Parameters

`progress` An integer representing the value of the percentage bar.

`taskTitle` A const char representing a text string to indicate status or progress to the user.

Return Value

This function does not return a value.

Example

```
GuiReferenceSetCurrentTaskProgress(0,"Starting Search, Please Wait...");  
// do something  
GuiReferenceSetCurrentTaskProgress(50,"Searching, Please Wait...");  
// do something else  
GuiReferenceSetCurrentTaskProgress(100,"Finished Searching.");  
// finished
```

Related functions

- *GuiReferenceAddColumn*
- *GuiReferenceDeleteAllColumns*
- *GuiReferenceGetCellContent*
- *GuiReferenceGetRowCount*
- *GuiReferenceInitialize*
- *GuiReferenceReloadData*
- *GuiReferenceSetCellContent*
- *GuiReferenceSetProgress*
- *GuiReferenceSetRowCount*
- *GuiReferenceSetSearchStartCol*
- *GuiReferenceSetSingleSelection*

GuiReferenceSetProgress

Sets the progress bar in the Reference View instance to a specific percentage value. This can be used to indicate to the user an operation is occurring, e.g. searching/sorting etc.

```
void GuiReferenceSetProgress(int progress);
```

Parameters

`progress` An integer representing the percentage value to set the progress bar to.

Return Value

This function does not return a value.

Example

```
GuiReferenceSetProgress(0);  
// do something  
GuiReferenceSetProgress(50);  
// do something else
```

```
GuiReferenceSetProgress(100);  
// tell user operation has ended
```

Related functions

- *GuiReferenceAddColumn*
- *GuiReferenceDeleteAllColumns*
- *GuiReferenceGetCellContent*
- *GuiReferenceGetRowCount*
- *GuiReferenceInitialize*
- *GuiReferenceReloadData*
- *GuiReferenceSetCellContent*
- *GuiReferenceSetCurrentTaskProgress*
- *GuiReferenceSetRowCount*
- *GuiReferenceSetSearchStartCol*
- *GuiReferenceSetSingleSelection*

GuiReferenceSetRowCount

Sets the total number of rows that the Reference View will contain.

```
void GuiReferenceSetRowCount(int count);
```

Parameters

count integer representing the total number of rows that the current Reference View will contain.

Return Value

This function does not return a value.

Example

```
GuiReferenceSetRowCount(5);
```

Related functions

- *GuiReferenceAddColumn*
- *GuiReferenceDeleteAllColumns*
- *GuiReferenceGetCellContent*

- *GuiReferenceGetRowCount*
- *GuiReferenceInitialize*
- *GuiReferenceReloadData*
- *GuiReferenceSetCellContent*
- *GuiReferenceSetCurrentTaskProgress*
- *GuiReferenceSetProgress*
- *GuiReferenceSetSearchStartCol*
- *GuiReferenceSetSingleSelection*

GuiReferenceSetSearchStartCol

Sets the search starting column in the current Reference View instance.

```
void GuiReferenceSetSearchStartCol (int col);
```

Parameters

`col` An integer representing the 0 based column to use for searching.

Return Value

This function does not return a value.

Example

```
GuiReferenceSetSearchStartCol (1);
```

Related functions

- *GuiReferenceAddColumn*
- *GuiReferenceDeleteAllColumns*
- *GuiReferenceGetCellContent*
- *GuiReferenceGetRowCount*
- *GuiReferenceInitialize*
- *GuiReferenceReloadData*
- *GuiReferenceSetCellContent*
- *GuiReferenceSetCurrentTaskProgress*
- *GuiReferenceSetProgress*
- *GuiReferenceSetRowCount*
- *GuiReferenceSetSingleSelection*

GuiReferenceSetSingleSelection

Sets the currently selected row in the Reference View instance.

```
void GuiReferenceSetSingleSelection(int index, bool scroll);
```

Parameters

`index` integer representing the row index to set the current selection to.

`scroll` a boolean value indicating if the selected index should be scrolled into view if it is not currently.

Return Value

This function does not return a value.

Example

```
GuiReferenceSetSingleSelection(0, true);
```

Related functions

- *GuiReferenceAddColumn*
- *GuiReferenceDeleteAllColumns*
- *GuiReferenceGetCellContent*
- *GuiReferenceGetRowCount*
- *GuiReferenceInitialize*
- *GuiReferenceReloadData*
- *GuiReferenceSetCellContent*
- *GuiReferenceSetCurrentTaskProgress*
- *GuiReferenceSetProgress*
- *GuiReferenceSetRowCount*
- *GuiReferenceSetSearchStartCol*

GuiRegisterScriptLanguage

Function description.

```
Function definition.
```


Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiRepaintTableView

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiScriptAdd

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiScriptClear

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiScriptEnableHighlighting

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiScriptError

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiScriptMessage

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiScriptMsgyn

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiScriptSetInfoLine

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiScriptSetIp

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiScriptSetTitle

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiSelectionGet

Gets the currently selected line (or lines) of the specified GUI view and returns the information as start and end addresses into a SELECTIONDATA variable.

```
bool GuiSelectionGet(int hWindow, SELECTIONDATA* selection)
```

Parameters

hWindow an integer representing one of the following supported GUI views: GUI_DISASSEMBLY, GUI_DUMP, GUI_STACK.

selection a SELECTIONDATA structure variable that stores the start and end address of the current selection.

Return Value

Return TRUE if successful or FALSE otherwise.

Example

```
SELECTIONDATA sel;  
GuiSelectionGet(GUI_DISASSEMBLY, &sel)  
sprintf(msg, "%p - %p", sel.start, sel.end);
```

Related functions

- *GuiSelectionSet*

GuiSelectionSet

Sets the currently selected line (or lines) of the specified GUI view based on the start and end addresses stored in a SELECTIONDATA variable.

```
bool GuiSelectionSet(int hWindow, const SELECTIONDATA* selection)
```

Parameters

`hWindow` an integer representing one of the following supported GUI views: GUI_DISASSEMBLY, GUI_DUMP, GUI_STACK.

`selection` a SELECTIONDATA structure variable that stores the start and end address of the current selection.

Return Value

Return TRUE if successful or FALSE otherwise.

Example

```
SELECTIONDATA sel;
GuiSelectionGet(GUI_DISASSEMBLY, &sel)
sel.end += 4; //expand selection
GuiSelectionSet(GUI_DISASSEMBLY, &sel)
```

Related functions

- *GuiSelectionGet*

GuiSetDebuggeeNotes

Sets the notes of the target being debugged (the debuggee), based on the text variable passed to the function. The text variable is a pointer to a string containing the information to set as the debuggee's notes.

```
void GuiSetDebuggeeNotes(char** text)
```

Parameters

`text` A variable that contains a pointer to a string that contains the text to set as the debuggee notes.

Return Value

This function does not return a value.

Example

```
const char* text = json_string_value(json_object_get(root, "notes"));
GuiSetDebuggeeNotes(text);
```

Related functions

- *GuiGetDebuggeeNotes*
- *GuiGetGlobalNotes*
- *GuiSetGlobalNotes*

GuiSetDebugState

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiSetGlobalNotes

Sets the global notes, based on the text variable passed to the function. The text variable is a pointer to a string containing the information to set as the global notes.


```
void GuiSetGlobalNotes(char** text)
```

Parameters

text A variable that contains a pointer to a string that contains the text to set as the global notes.

Return Value

This function does not return a value.

Example

```
notesFile = String(szProgramDir) + "\\notes.txt";
String text;
if(!FileExists(notesFile.c_str()) || FileHelper::ReadAllText(notesFile, text))
    GuiSetGlobalNotes(text.c_str());
```

Related functions

- *GuiGetGlobalNotes*
- *GuiGetDebuggeeNotes*
- *GuiSetDebuggeeNotes*

GuiSetLastException

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiShowCpu

Switches the focus and view of the GUI to the main disassembly window (CPU tab)

```
void GuiShowCpu();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiShowCpu();
```

Related functions

GuiShowQWidgetTab

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiStackDumpAt

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiSymbolLogAdd

Function description.

```
void GuiSymbolLogAdd(const char* message);
```

Parameters

message String containing the message to add to the symbol log. Ensure that a carriage line and return feed are included with the string for it to properly display it. Encoding is UTF-8.

Return Value

This function does not return a value.

Example

```
GuiSymbolLogAdd (&szMsg) ;
```

Related functions

- *GuiSymbolLogClear*
- *GuiSymbolRefreshCurrent*
- *GuiSymbolSetProgress*
- *GuiSymbolUpdateModuleList*

GuiSymbolLogClear

Function description.

```
void GuiSymbolLogClear ();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiSymbolLogClear ();
```

Related functions

- *GuiSymbolLogAdd*
- *GuiSymbolRefreshCurrent*
- *GuiSymbolSetProgress*
- *GuiSymbolUpdateModuleList*

GuiSymbolRefreshCurrent

Refreshes the symbol view list of symbols and exports.

```
void GuiSymbolRefreshCurrent ();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiSymbolRefreshCurrent ();
```

Related functions

- *GuiSymbolLogAdd*
- *GuiSymbolLogClear*
- *GuiSymbolSetProgress*
- *GuiSymbolUpdateModuleList*

GuiSymbolSetProgress

Sets the progress bar in the symbol view based on the integer value supplied. This can be used to convey to the user an operation and how close it is to completion, for example with searches.

```
void GuiSymbolSetProgress (int percent);
```

Parameters

percent an integer representing the percentage to set for the progress bar.

Return Value

This function does not return a value.

Example

```
GuiSymbolSetProgress (50);
```

Related functions

- *GuiSymbolLogAdd*
- *GuiSymbolLogClear*
- *GuiSymbolRefreshCurrent*
- *GuiSymbolUpdateModuleList*

GuiSymbolUpdateModuleList

Refreshes the symbol view modules list.

```
void GuiSymbolUpdateModuleList(int count, SYMBOLMODULEINFO* modules)
```

Parameters

count An integer representing the number of symbol module's to update.

modules A SYMBOLMODULEINFO variable that will hold the symbol module information.

Return Value

This function does not return a value.

Example

```
// Build the vector of modules
std::vector<SYMBOLMODULEINFO> modList;

if(!SymGetModuleList(&modList))
{
    GuiSymbolUpdateModuleList(0, nullptr);
    return;
}

// Create a new array to be sent to the GUI thread
size_t moduleCount = modList.size();
SYMBOLMODULEINFO* data = (SYMBOLMODULEINFO*)BridgeAlloc(moduleCount *
↳ sizeof(SYMBOLMODULEINFO));

// Direct copy from std::vector data
memcpy(data, modList.data(), moduleCount * sizeof(SYMBOLMODULEINFO));

// Send the module data to the GUI for updating
GuiSymbolUpdateModuleList((int)moduleCount, data);
```

Related functions

- *GuiSymbolLogAdd*

- *GuiSymbolLogClear*
- *GuiSymbolRefreshCurrent*
- *GuiSymbolSetProgress*

GuiUnregisterScriptLanguage

Function description.

```
Function definition.
```

Parameters

param1 Parameter description.

Return Value

Return value description.

Example

```
Example code.
```

Related functions

- List of related functions

GuiUpdateAllViews

Refreshes the contents of all opened views (tabs)

```
void GuiUpdateAllViews ();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateAllViews ();
```

Related functions

- *GuiUpdateEnable*
- *GuiUpdateDisable*
- *GuiIsUpdateDisabled*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateArgumentWidget

Refreshes the contents of the arguments widget.

```
void GuiUpdateArgumentWidget ();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateArgumentWidget ();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateBreakpointsView

Refreshes the contents of the breakpoints view.

```
void GuiUpdateBreakpointsView();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateBreakpointsView();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateCallStack

Refreshes the contents of the call stack widget.

```
void GuiUpdateCallStack();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateCallStack();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateDisable

Sets an internal variable to disable the update of views. To re-enable updating after processing or performing some operation, use the `GuiUpdateEnable` function.

```
void GuiUpdateDisable();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateDisable();
```

Related functions

- *GuiIsUpdateDisabled*
- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateDisassemblyView

Refreshes the contents of the disassembly view (CPU tab).

```
void GuiUpdateDisassemblyView();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateDisassemblyView();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateDumpView

Refreshes the contents of the dump views.

```
void GuiUpdateDumpView();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateDumpView();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateEnable

Sets an internal variable to enable (or re-enable if previously disabled via `GuiUpdateDisable`) the refresh of all views. The `updateNow` parameter can be used to force the update straight away, otherwise the updates of views will continue as per normal message scheduling, now that they have been enabled with `GuiUpdateEnable`.

```
void GuiUpdateEnable (bool updateNow);
```

Parameters

`updateNow` is a boolean value indicating if the update of all views should occur straight away.

Return Value

This function does not return a value.

Example

```
GuiUpdateEnable (bool updateNow) ;
```

Related functions

- *GuiIsUpdateDisabled*
- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateGraphView

Refreshes the contents of the Graph view.

```
void GuiUpdateGraphView() ;
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateGraphView();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateMemoryView

Refreshes the contents of the memory view.

```
void GuiUpdateMemoryView();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateMemoryView();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdatePatches

Refreshes the contents of the patches.

```
void GuiUpdatePatches();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdatePatches ();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateRegisterView

Refreshes the contents of the registers view.

```
void GuiUpdateRegisterView ();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateRegisterView();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateSEHChain

Refreshes the contents of the SEH Chain.

```
void GuiUpdateSEHChain();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateSEHChain();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateSideBar

Refreshes the contents of the Sidebar.

```
void GuiUpdateSideBar();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateSideBar();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateThreadView

Refreshes the contents of the threads view.

```
void GuiUpdateThreadView();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateThreadView();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateTimeWastedCounter

Refreshes the time wasted counter.

```
void GuiUpdateTimeWastedCounter();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateTimeWastedCounter();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateWatchView

Refreshes the contents of the watchdog view.

```
void GuiUpdateWatchView();
```

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
GuiUpdateWatchView();
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

GuiUpdateWindowTitle

Updates the x64dbg window title with a string to be appended to the title text. Typically the string is a filename.

```
void GuiUpdateWindowTitle(const char* filename)
```

Parameters

`filename` a const char variable to be appended to the x64dbg title bar.

Return Value

This function does not return a value.

Example

```
GuiUpdateWindowTitle(" ");  
GuiUpdateWindowTitle(szFileName);
```

Related functions

- *GuiUpdateAllViews*
- *GuiUpdateArgumentWidget*
- *GuiUpdateBreakpointsView*
- *GuiUpdateCallStack*
- *GuiUpdateDisable*
- *GuiUpdateDisassemblyView*
- *GuiUpdateDumpView*
- *GuiUpdateEnable*
- *GuiUpdateGraphView*
- *GuiUpdateMemoryView*
- *GuiUpdatePatches*
- *GuiUpdateRegisterView*
- *GuiUpdateSEHChain*
- *GuiUpdateSideBar*
- *GuiUpdateThreadView*
- *GuiUpdateTimeWastedCounter*
- *GuiUpdateWatchView*
- *GuiUpdateWindowTitle*

1.5.3 x64dbg trace file format specification

x64dbg trace file is a binary file that contains all information about program execution. Each trace file is composed of 3 parts: **Magic word**, **JSON header** and **binary trace blocks**. x64dbg trace file is little-endian.

Magic word

Every trace file will begin with 4 bytes, "TRAC" (encoded in ASCII).

Header

Header is located after header at offset 4. It is composed of a 4-byte length field, followed by a JSON blob. The JSON blob might not be null-terminated and might not be aligned to 4-byte boundary.

Binary trace blocks

Binary trace data is immediately after header without any padding and might not be aligned to 4-byte boundary. It is defined as a sequence of blocks. Currently, only block type 0 is defined.

Every block is started with a 1-byte type number. This type number must be 0, which means it is a block that describes an instruction traced.

If the type number is 0, then the block will contain the following data:

```
struct {
    uint8_t BlockType; //BlockType is 0, indicating it describes an instruction_
    ↪execution.
    uint8_t RegisterChanges;
    uint8_t MemoryAccesses;
    uint8_t BlockFlagsAndOpcodeSize; //Bitfield

    DWORD ThreadId;
    uint8_t Opcode[];

    uint8_t RegisterChangePosition[];
    duint RegisterChangeNewData[];

    uint8_t MemoryAccessFlags[];
    duint MemoryAccessAddress[];
    duint MemoryAccessOldData[];
    duint MemoryAccessNewData[];
};
```

RegisterChanges is a unsigned byte that counts the number of elements in the array RegisterChangePosition and RegisterChangeNewData.

MemoryAccesses is a unsigned byte that counts the number of elements in the array MemoryAccessFlags.

BlockFlagsAndOpcodeSize is a bitfield. The most significant bit is ThreadId bit. When this bit is set, ThreadId field is available and indicates the thread id which executed the instruction. When this bit is clear, the thread id that executed the instruction is the same as last instruction, so it is not stored in file. The least 4 significant bits specify the length of Opcode field, in number of bytes. Other bits are reserved and set to 0. Opcode field contains the opcode of current instruction.

RegisterChangePosition is an array of unsigned bytes. Each element indicates a pointer-sized integer in struct REGDUMP that is updated after execution of current instruction, as an offset to previous location. The absolute index is computed by adding the absolute index of previous element +1 (or 0 if it is first element) with this relative index. RegisterChangeNewData is an array of pointer-sized integers that contains the new value of register. REGDUMP structure is given below.

```
typedef struct
{
    REGISTERCONTEXT regcontext;
    FLAGS flags;
    X87FPUREGISTER x87FPURegisters[8];
    unsigned long long mmx[8];
    MXCSRFIELDS MxCsrFields;
    X87STATUSWORDFIELDS x87StatusWordFields;
    X87CONTROLWORDFIELDS x87ControlWordFields;
    LASTERROR lastError;
    //LASTSTATUS lastStatus; //This field is not supported and not included in trace_
    ↪file.
} REGDUMP;
```

For example, `ccx` is the second member of `regcontext`. On x64 architecture, it is at byte offset 8 and on x86 architecture it is at byte offset 4. On both architecture, it is at index 1 and `cax` is at index 0. Therefore, when `RegisterChangePosition[0] = 0`, `RegisterChangeNewData[0]` contains the new value of `cax`. If `RegisterChangePosition[1] = 0`, `RegisterChangeNewData[1]` contains the new value of `ccx`, since the absolute index is computed by $0+0+1=1$. The use of relative indexing helps achieve better data compression if a lossless compression is then applied to trace file, and also allow future expansion of `REGDUMP` structure without increasing size of `RegisterChanges` and `RegisterChangePosition` beyond a byte.

x64dbg will save all registers at the start of trace, and every 512 instructions (this number might be changed in future versions to have different tradeoff between speed and space). A block with all registers saved will have `RegisterChanges=172` on 64-bit platform and 216 on 32-bit platform. This allows x64dbg trace file to be randomly accessed. x64dbg might be unable to open a trace file that has a sequence of instruction longer than an implementation-defined limit without all registers saved.

`MemoryAccessFlags` is an array of bytes that indicates properties of memory access. Currently, only bit 0 is defined and all other bits are reserved and set to 0. When bit 0 is set, it indicates the memory is not changed (This could mean it is read, or it is overwritten with identical value), so `MemoryAccessNewData` will not have an entry for this memory access.

`MemoryAccessAddress` is an array of pointers that indicates the address of memory access.

`MemoryAccessOldData` is an array of pointer-sized integers that stores the old content of memory.

`MemoryAccessNewData` is an array of pointer-sized integers that stores the new content of memory.

1.6 Licenses

Licenses:

1.6.1 License of x64dbg

```
GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

THIS DOCUMENT HAS BEEN MODIFIED
```

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

1. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

1. Treatment of plugins

This license does not affect plugins, i.e., dynamically linked libraries, that use the provided plugin interface mechanism of x64dbg for contributing additional features to the x64dbg project and can only be run from x64dbg. In fact you are allowed to create and share (non-)commercial, non-standalone closed-source plugins for x64dbg under your own license.

1. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

1. Protecting Users’ Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work’s users, your or third parties’ legal rights to forbid circumvention of technological measures.

1. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

1. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, **and** giving a relevant date.
- b) The work must carry prominent notices stating that it **is** released under this License **and any** conditions added under section 7. This requirement modifies the requirement **in** section 4 to "keep intact all notices".
- c) You must license the entire work, **as** a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along **with any** applicable section 7 additional terms, to the whole of the work, **and all** its parts, regardless of how they are packaged. This License gives no permission to license the work **in any** other way, but it does **not** invalidate such permission **if** you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, **if** the Program has interactive interfaces that do **not** display Appropriate Legal Notices, your work need **not** make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

1. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the **object** code **in, or** embodied **in**, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used **for** software interchange.
- b) Convey the **object** code **in, or** embodied **in**, a physical product (including a physical distribution medium), accompanied by a written offer, valid **for** at least three years **and** valid **for as** long **as** you offer spare parts **or** customer support **for** that product model, to give anyone who possesses the **object** code either (1) a copy of the Corresponding Source **for all** the software **in** the product that **is** covered by this License, on a durable physical medium customarily used **for** software interchange, **for** a price no

more than your reasonable cost of physically performing this conveying of source, **or** (2) access to copy the Corresponding Source **from a** network server at no charge.

c) Convey individual copies of the **object** code **with** a copy of the written offer to provide the Corresponding Source. This alternative **is** allowed only occasionally **and** noncommercially, **and** only **if** you received the **object** code **with** such an offer, **in** accord **with** subsection 6b.

d) Convey the **object** code by offering access **from a** designated place (gratis **or for** a charge), **and** offer equivalent access to the Corresponding Source **in** the same way through the same place at no further charge. You need **not** require recipients to copy the Corresponding Source along **with** the **object** code. If the place to copy the **object** code **is** a network server, the Corresponding Source may be on a different server (operated by you **or** a third party) that supports equivalent copying facilities, provided you maintain clear directions **next** to the **object** code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it **is** available **for as long as** needed to satisfy these requirements.

e) Convey the **object** code using peer-to-peer transmission, provided you inform other peers where the **object** code **and** Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

1. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty **or** limiting liability differently **from the** terms of sections 15 **and** 16 of this License; **or**
- b) Requiring preservation of specified reasonable legal notices **or** author attributions **in** that material **or in** the Appropriate Legal Notices displayed by works containing it; **or**
- c) Prohibiting misrepresentation of the origin of that material, **or** requiring that modified versions of such material be marked **in** reasonable ways **as** different **from the** original version; **or**
- d) Limiting the use **for** publicity purposes of names of licensors **or** authors of the material; **or**
- e) Declining to grant rights under trademark law **for** use of some trade names, trademarks, **or** service marks; **or**
- f) Requiring indemnification of licensors **and** authors of that material by anyone who conveys the material (**or** modified versions of it) **with** contractual assumptions of liability to the recipient, **for** any liability that these contractual assumptions directly impose on those licensors **and** authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

1. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

1. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

1. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

1. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange,

in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

1. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

1. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

1. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

1. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

1. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

```
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program> Copyright (C) <year> <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
```

```
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands `show w'` and `show c'` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

1.6.2 TitanEngine Update

```
GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007
```

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

1. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

1. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

1. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license document.

1. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d) Do one of the following:

```
0) Convey the Minimal Corresponding Source under the terms of this
License, and the Corresponding Application Code in a form
suitable for, and under terms that permit, the user to
recombine or relink the Application with a modified version of
the Linked Version to produce a modified Combined Work, in the
manner specified by section 6 of the GNU GPL for conveying
Corresponding Source.
```

```
1) Use a suitable shared library mechanism for linking with the
Library. A suitable mechanism is one that (a) uses at run time
a copy of the Library already present on the user's computer
system, and (b) will operate properly with a modified version
of the Library that is interface-compatible with the Linked
Version.
```

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

1. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

1. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy’s public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

1.6.3 Zydis

The MIT License (MIT)

Copyright (c) 2018 Florian Bernd Copyright (c) 2018 Joel Höner

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.6.4 XEDParse

GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

1. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

1. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

- b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

1. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

- b) Accompany the object code with a copy of the GNU GPL and this license document.

1. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this License, **and** the Corresponding Application Code **in** a form suitable **for**, **and** under terms that permit, the user to recombine **or** relink the Application **with** a modified version of the Linked Version to produce a modified Combined Work, **in** the manner specified by section 6 of the GNU GPL **for** conveying Corresponding Source.

1) Use a suitable shared library mechanism **for** linking **with** the Library. A suitable mechanism **is** one that (a) uses at run time a copy of the Library already present on the user's **computer** system, **and** (b) will operate properly **with** a modified version of the Library that **is** interface-compatible **with** the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

1. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
- b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

1. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy’s public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

1.6.5 asmjit

Copyright (c) 2008-2017, Petr Kobilicek

This software is provided ‘as-is’, without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

1.6.6 jansson

Copyright (c) 2009-2018 Petri Lehtinen petri@digip.org

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.6.7 LZ4

LZ4 Library Copyright (c) 2011-2014, Yann Collet All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.6.8 Snowman Licensing Information

Snowman uses source code of multiple projects and, as a whole, is licensed under the terms of the GNU General Public License as published by the Free Software Foundation, version 3 or any later version. In addition, you can use all the code and changes contributed to the project and not explicitly copyrighted otherwise, e.g., by a note in a commit message, under the following terms.

Copyright (c) 2010 Yegor Derevenets <yegor.derevenets@gmail.com> All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 2

Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)