Sinker

widberg

Nov 25, 2023

CONTENTS

CHAPTER

ONE

SINKER SCRIPT

Sinker Script can be written out-of-line in a separate file with the `.skr` extension. It can also be written inline alongside source code in the same file where lines to be evaluated as Sinker Script begin with `//$`; the "$" is for the "S" in "Sinker", I'm too clever for my own good. Only whitespace is allowed before this token on a line, not unlike the C preprocessor. Any file with an extension other than `.skr` is assumed to be a source code file.

1.1 Language Elements

1.1.1 Directive

Every Sinker Script statement starts with a *directive*.

1.1.2 Identifier

An identifier is a token matching the Regex `[a-zA-Z_$][a-zA-Z0-9_$]*` that does not already have a meaning in Sinker Script.

1.1.3 Identifier Set

A non-empty comma-separated list of identifiers surrounded by square brackets or an asterisk surrounded by square brackets, a wildcard representing all variants.

1.1.4 Integer Literal

An integer literal is a decimal, hexadecimal, prefixed with `0x`, octal, prefixed with `0`, or binary, prefixed with `0b`, integer.

1.1.5 String Literal

A string literal is any sequence of characters enclosed in quotes. There are no escape sequences. Adjoining string literals will be treated as a single string literal i.e. `"sink" "er ro" "cks"` is equivalent to `"sinker rocks"`. This can be used to split a string literal across multiple lines.

1.1.6 Boolean Literal

A boolean literal is either `true` or `false`.

1.1.7 Expression

An expression is written in the *Sinker Script Expression Language*.

1.1.8 Comments

Any characters between the sequence `//` and the end of a line will be ignored in Sinker Script.

1.2 Directives

1.2.1 Module

A module corresponds to a target PE file, `.exe` or `.dll`, loaded into the injected process's memory space. The `lpModuleName` is the argument passed to `GetModuleHandle`; exclude this argument to pass `NULL` to `GetModuleHandle`.

```
module <module_name:identifier>, <lpModuleName:string>;
module <module_name:identifier>;
```

1.2.2 Variant

A variant corresponds to a known distribution of a target PE file, identified by its SHA256 hash or if an expression is able to be resolved. This makes it easier to provide known addresses for each binary variant. If a PE file does not match any of the known hashes and none of the expressions are able to be resolved, it will not have a variant name. The SHA256 hash of a file can be obtained using the `certUtil -hashfile C:\file.exe SHA256` command on Windows.

```
variant <module_name:identifier>, <variant_name:identifier>, <sha256_hash:string>;
variant <module_name:identifier>, <variant_name:identifier>, <expression:expression>;
```

1.2.3 Symbol

A symbol is a variant-agnostic representation of an address in a binary. A symbol can be pretty much anything: function, vtable, global data, etc. As you will soon see, Sinker is very versatile.

```
symbol <module_name:identifier>::<symbol_name:identifier>, <symbol_type:string>;
```

1.2.4 Address

An address provides instructions on how to calculate the address for a symbol based on its variant. These calculations are done using the *Sinker Script Expression Language*.

Address directives for each symbol are evaluated in the order they are declared. For each address directive where a `variant_name` in the set matches the current module's variant or the wildcard is used, the expression is evaluated. The first expression that is resolved will be the calculated address of the symbol. If all expressions are unresolved, the symbol is unresolved.

It is generally advisable to declare at least one address declaration where the variant is a wildcard. This is so that if a module does not have a variant name, due to no hashes matching or lack of variant declarations, there is still an opportunity to resolve an address for the symbol.

```
address <module_name:identifier>::<symbol_name:identifier>, <variant_names:identifier_set>,
<expression:expression>;
```

1.2.5 Set

A module or symbol can have arbitrary user-defined attributes associated with them. This can be used, for example, to mark a symbol as "required" and check at runtime if all "required" symbols have been resolved.

```
set <module_name:identifier>, <attribute_name:identifier>,
<value:boolean|integer|string>;
set <module_name:identifier>::<symbol_name:identifier>, <attribute_name:identifier>,
<value:boolean|integer|string>;
```

1.2.6 Tag

Modules or symbols can be grouped by arbitrary user-defined tags. These can be used, for example, to mark symbols as "hookable" and generate code to hook them at compile-time. Tags cannot have values and are accessible at compile-time, unlike attributes. Tags and attributes can be combined to generate code for all "hookable" symbols at compile time and then only hook the "enabled" symbols at runtime for example.

```
tag <module_name:identifier>, <tag_name:identifier>;
tag <module_name:identifier>::<symbol_name:identifier>, <tag_name:identifier>;
```

1.3 Sinker Script Expression Language

Any operation with an unresolved operand will evaluate as unresolved; or, in other words, if any part of an expression is unresolved then the whole expression is unresolved.

1.3.1 Integer Literal

`integer`

An integer literal will be evaluated as its numeric value.

1.3.2 Identifier

`module_name`

A module's name will be evaluated as its relocated base address or unresolved if the module has not been concretized.

`module_name::symbol_name`

A symbol's name will be evaluated as its calculated address or unresolved.

1.3.3 GetProcAddress

`!module_name::lpProcName`

Use `GetProcAddress` to find `lpProcName` in `module_name`. If found this evaluates to the returned address, otherwise unresolved.

1.3.4 Pattern Match

```
{}
{ needle }
{ needle :  mask }
[filter]{ needle }
[filter]{ needle :  mask }
[filter]{}
```

Inspired by Frida's JavaScript API's `Memory.scan` which is in turn inspired by Radare2's `/x` command.

Filter

Filters are optional. If a filter is specified and no needle is specified, then the expression will evaluate to the first searched address matching the filter. This can be used to get the address of a module's text segment by filtering for it and not using a needle for example. The following filters are supported:

- No filter. Search all readable pages.
- `module_name` search all sections in the specified module.
- `module_name::"section_name"` search the section in the specified module.

A comma separated list of filters may be used. If the module in a filter has not been concretized then that filter is skipped. If none of them are then the expression is unresolved.

Needle

Searches for the first occurrence of the pattern in the module text segment and evaluates to the address of the first byte of the matched pattern. If no match is found, the pattern match evaluates to unresolved. A needle contains a series of the following:

- `XX` a hexadecimal byte value with no prefix. The search byte must equal this value.
- `??` the search byte may be equal to any value.
- `X?` lower nibble wildcard, the high nibble of the search byte must equal the high nibble of this value.
- `?X` upper nibble wildcard, the low nibble of the search byte must equal the low nibble of this value.
- `"string"` a string literal. Insert the ASCII bytes of the string into the needle.
- `&` the pattern match expression will evaluate to the address of byte following this if specified. Can only be used once. This can be used to match a whole jump instruction but evaluate as the address of the operand of the jump.

Mask

The mask is optional. The needle and mask must be the same length. Wildcards in the needle cannot be mixed with a mask.

- `XX` a hexadecimal byte value with no prefix. The needle and haystack will be AND'd with this value.

1.3.5 Operations

Parentheses

`(expression)`

Parentheses can be used to change the sequence of evaluation.

Mathematical Operations

`expression + expression`
`expression - expression`
`expression * expression`
`expression / expression` (Integer Division)
`expression % expression` (Modulo)

Bitwise Operations

`expression << expression` (Left Shift)
`expression >> expression` (Right Shift)
`expression & expression` (Bitwise AND)
`expression ^ expression` (Bitwise XOR)
`expression | expression` (Bitwise OR)
`~expression` (Bitwise NOT)

Mathematical operations are applied as if the expressions are integers; there is no pointer arithmetic in Sinker Script.

Indirection (dereference)

`*expression`

The expression to be dereferenced will be treated as a `void**`, the result of the dereference operation will be an address, `void*`. Smaller or larger values can be dereferenced by using this and then masking out bytes using `&` or combining multiple dereferences using << and `|`. A more comprehensive type system may be added in the future. System endianness will be used. If dereferencing the expression causes an access violation, the expression will evaluate to unresolved. From this definition of the Indirection operator, an easy way to raise an unresolved value arises, `*0`; I'm not sure why you would want to do this, but hey I can't stop you.

Array Subscripting

`expression1[expression2]`

Equivalent to `*(expression1 + expression2 * sizeof(void*))` where `sizeof(void*)` is the size, in bytes, of a pointer; note that `sizeof(void*)` is purely demonstrative of the behavior of the operation and not valid Sinker Script.

Pointer Path

`expression1->expression2`

Equivalent to `*expression1 + expression2`. This can be chained together multiple times for a LiveSplit Auto Splitter style pointer path i.e. `0xDEADBEEF->0xABCD->0x1234` will read an address at `0xDEADBEEF` then add `0xABCD` and read an address there, finally `0x1234` is added to that address.
Inspired by LiveSplit Auto Splitter Pointer Paths.

Relocate

`@expression`

This will subtract the symbol's module's preferred base address from the expression and then add the symbol's module's relocated base address to the expression.

Operator Precedence

Adapted from C Operator Precedence.

Precedence	Operator	Description	Associativity
1	`[]` `->`	Array Subscripting Pointer Path	Left-to-right

2	`!`
	`*`
	`@`
	`~`
		GetProcAddress
		Indirection (dereference)
		Relocate
		Bitwise NOT
			Right-to-left

3	`*`
	`/`
	`%`
		Multiplication
		Integer Division
		Modulo
			Left-to-right

4	`+` `-`	Addition Subtraction	Left-to-right
5	`<<` `>>`	Left Shift Right Shift	Left-to-right
6	`&`	Bitwise AND	Left-to-right
7	`^`	Bitwise XOR	Left-to-right
8	`|`	Bitwise OR	Left-to-right

CHAPTER

TWO

SINKER COMPILER

The Sinker Compiler can be used to amalgamate multiple Sinker DSL files into one file. It can also be used to generate a `.def` header file that can be included in C++ source code with useful macros and boilerplate code for the Sinker Runtime Library.

The compiler accepts a list of positional input source file paths that will be evaluated in the order the arguments are specified. The `-o <amalgamated_file_path>` option can be used to specify where the output file should go. If this option is missing, then no `.skr` file will be output. Finally, the `-d <def_file_path>` can be used to specify where the `.def` file should go. If this option is missing, then no `.def` file will be output.

2.1 .def

For the following `.skr` file

```
module crackme;
variant crackme, v1_0_0, "deadbeefdeadbeefdeadbeefdeadbeef"
symbol check_flag, "bool(*)(char const * flag)";
tag crackme::check_flag, hook;
address crackme::check_flag, v1_0_0, @0xc0dec0de;
```

The `.def` file will contain the following:

```
#ifndef SINKER_MODULE
#define SINKER_MODULE(module_name)
#endif
#ifndef SINKER_SYMBOL
#define SINKER_SYMBOL(module_name, symbol_name, symbol_type)
#endif
#ifndef SINKER_TAG_hook_SYMBOL
#define SINKER_TAG_hook_SYMBOL(module_name, symbol_name, symbol_type)
#endif

#ifndef SINKER_crackme_SYMBOL
#define SINKER_crackme_SYMBOL(symbol_name, symbol_type)
#endif
#ifndef SINKER_crackme_TAG_hook_SYMBOL
#define SINKER_crackme_TAG_hook_SYMBOL(symbol_name, symbol_type)
#endif
SINKER_MODULE(crackme)
SINKER_SYMBOL(crackme, check_flag, bool(*)(char const * flag))
```

(continues on next page)

(continued from previous page)

```
SINKER_TAG_hook_SYMBOL(crackme, check_flag, bool(*)(char const * flag))
SINKER_crackme_SYMBOL(check_flag, bool(*)(char const * flag))
SINKER_crackme_TAG_hook_SYMBOL(check_flag, bool(*)(char const * flag))
#undef SINKER_crackme_TAG_hook_SYMBOL
#undef SINKER_crackme_SYMBOL

#undef SINKER_TAG_hook_SYMBOL
#undef SINKER_MODULE
#undef SINKER_SYMBOL
```

and it can be used like the following simple example:

```
#include <stdio.h>
void print_modules() {
#define SINKER_MODULE(module_name) \
    puts(#module_name);
#include "a.def"
}

void print_symbols() {
#define SINKER_SYMBOL(module_name, symbol_name, symbol_type) \
    puts(#symbol_name);
#include "a.def"
}
```

The macros can be redefined and the `.def` can be included as many times as necessary.

CHAPTER

THREE

SINKER RUNTIME LIBRARY

class *sinker*::**Action**

Subclassed by *sinker::ActionInstall*, *sinker::ActionUninstall*

Public Functions

virtual void **act**() = 0

class *sinker*::**ActionInstall** : public *sinker*::*Action*

Public Functions

inline **ActionInstall**(*Installable* *installable)

inline virtual void **act**() override

Private Members

Installable ***installable** = nullptr

class *sinker*::**ActionUninstall** : public *sinker*::*Action*

Public Functions

inline **ActionUninstall**(*Uninstallable* *uninstallable)

inline virtual void **act**() override

Private Members

Uninstallable ***uninstallable** = nullptr

class *sinker*::**Attributable**

Subclassed by *sinker::Module*, *sinker::Symbol*

Public Functions

template<typename **T**>
std::optional<*T*> **get_attribute**(std::string_view attribute_name) const

template<typename **T**>
void **set_attribute**(std::string const &attribute_name, *T* value)

std::map<std::string, *attribute_value_t*, std::less<>> const &**get_attributes**() const

Private Members

std::map<std::string, *attribute_value_t*, std::less<>> **attributes**

class *sinker*::**BinaryOperatorExpression** : private *sinker*::*Expression*

Public Functions

inline **BinaryOperatorExpression**(std::shared_ptr<*Expression*> lhs, std::shared_ptr<*Expression*> rhs, *BinaryOperator* binary_operator)

inline virtual std::optional<*expression_value_t*> **calculate**(*Symbol* *symbol) const override

inline virtual void **dump**(std::ostream &out) const override

Private Members

std::shared_ptr<*Expression*> **lhs**

std::shared_ptr<*Expression*> **rhs**

BinaryOperator **binary_operator**

class *sinker*::**Context**

Public Functions

inline **Context**()

Context(const *Context*&) = delete

Context &**operator=**(const *Context*&) = delete

inline std::vector<*Module**> const &**get_modules**() const

Module ***get_module**(std::string_view module_name)

void **emplace_module**(std::string_view name, std::optional<std::string> lpModuleName)

void **dump**(std::ostream &out) const

void **dump_def**(std::ostream &out) const

bool **interpret**(std::istream &input_stream, *Language* language, std::string input_filename, bool debug = false)

bool **interpret**(const char *input, std::size_t size, *Language* language, std::string input_filename, bool debug = false)

bool **interpret**(const std::string &input, *Language* language, std::string input_filename, bool debug = false)

void **add_module_tag**(std::string const &tag)

void **add_symbol_tag**(std::string const &tag)

identifier_set_t const &**get_symbol_tags**() const

~Context()

Private Members

std::vector<*Module**> **modules**

identifier_set_t **module_tags**

identifier_set_t **symbol_tags**

template<typename **T**>

class `sinker`**::Detour** : public *sinker*::*Installable*, public *sinker*::*Uninstallable*

Public Functions

inline **Detour**(*T* &real, *T* wrap)

inline virtual void **install**() override

inline virtual void **uninstall**() override

Private Members

T ***real** = {}

T **wrap** = {}

class *sinker*::**Expression**

Subclassed by *sinker::BinaryOperatorExpression*, *sinker::GetProcAddressExpression*, *sinker::IntegerExpression*, *sinker::ModuleExpression*, *sinker::PatternMatchExpression*, *sinker::SymbolExpression*, *sinker::UnaryOperatorExpression*

Public Functions

virtual std::optional<*expression_value_t*> **calculate**(*Symbol* *symbol) const = 0

virtual void **dump**(std::ostream &out) const = 0

inline virtual **~Expression**()

class *sinker*::**GetProcAddressExpression** : private *sinker*::*Expression*

Public Functions

inline **GetProcAddressExpression**(*Module* *module, std::string const &lpProcName)

inline virtual std::optional<*expression_value_t*> **calculate**(*Symbol* *symbol) const override

inline virtual void **dump**(std::ostream &out) const override

Private Members

Module ***module**

std::string **lpProcName**

class *sinker*::**Installable**

Subclassed by *sinker::Detour< T >*, *sinker::Patch< T >*, *sinker::Patch< T[N]>*

Public Functions

virtual void **install**() = 0

class *sinker*::**IntegerExpression** : private *sinker*::*Expression*

Public Functions

inline **IntegerExpression**(*expression_value_t* value)

inline virtual std::optional<*expression_value_t*> **calculate**(*Symbol* *symbol) const override

inline virtual void **dump**(std::ostream &out) const override

Private Members

expression_value_t **value**

struct *sinker*::**MaskedByte**

Public Members

std::uint8_t **value**

std::uint8_t **mask**

class *sinker*::**Module** : public *sinker*::*Attributable*

Public Functions

Module(const *Module*&) = delete

Module &**operator=**(const *Module*&) = delete

Module(*Module*&&) = default

Module &**operator=**(*Module* &&mE) = default

std::string const &**get_name**() const

std::string const &**get_real_variant**() const

Symbol ***get_symbol**(std::string_view symbol_name)

void **emplace_symbol**(std::string const &name, std::string const &type)

void **add_variant**(std::string const &name, std::string const &hash)

bool **has_variant**(std::string_view name) const

void **dump**(std::ostream &out) const

void **dump_def**(std::ostream &out) const

std::optional<*expression_value_t*> **get_preferred_base_address**() const

std::optional<*expression_value_t*> **get_relocated_base_address**() const

HMODULE **get_hModule**() const

void **add_tag**(std::string const &tag)

Context ***get_context**() const

bool **concretize**()

bool **is_concrete**() const

Private Functions

inline **Module**(std::string_view name, std::optional<std::string> lpModuleName, *Context* *context)

Private Members

Context ***context**

std::string **name**

std::optional<std::string> **lpModuleName**

std::optional<*expression_value_t*> **preferred_base_address**

std::optional<*expression_value_t*> **relocated_base_address**

std::vector<*Symbol*> **symbols**

std::map<std::string, std::string, std::less<>> **variants**

std::string **real_variant**

HMODULE **hModule** = 0

identifier_set_t **tags**

Friends

friend class Context

class *sinker*::**ModuleExpression** : private *sinker*::*Expression*

Public Functions

inline **ModuleExpression**(*Module* *module)

inline virtual std::optional<*expression_value_t*> **calculate**(*Symbol* *symbol) const override

inline virtual void **dump**(std::ostream &out) const override

Private Members

Module ***module**

template<typename **T**>

class *sinker*::**Patch** : public *sinker*::*Installable*, public *sinker*::*Uninstallable*

Public Functions

inline **Patch**(*T* *dst, *T* *src)

inline virtual void **install**() override

inline virtual void **uninstall**() override

Private Members

T ***dst** = {}

T ***src** = {}

T **backup** = {}

template<typename **T**, std::size_t **N**>

class *sinker*::**Patch**<*T*[*N*]> : public *sinker*::*Installable*, public *sinker*::*Uninstallable*

Public Functions

inline **Patch**(*T* *dst, *T* *src)

inline virtual void **install**() override

inline virtual void **uninstall**() override

Private Members

T ***dst** = {}

T ***src** = {}

T **backup**[*N*] = {}

class *sinker*::**PatternMatchExact** : private *sinker*::*PatternMatchFragment*

Public Functions

inline **PatternMatchExact**(const std::vector<std::uint8_t> &value)

inline virtual void ***search**(void *begin, void *end) const override

inline virtual bool **begins_with**(void *begin, void *end) const override

inline virtual std::size_t **size**() const override

inline virtual *PatternMatchType* **type**() const override

inline virtual bool **collision**(void *address) const

Private Members

std::vector<std::uint8_t> **value**

class *sinker*::**PatternMatchExpression** : private *sinker*::*Expression*

Public Functions

inline **PatternMatchExpression**(std::vector<*MaskedByte*> const &needle, *expression_value_t* offset = 0, std::vector<*PatternMatchFilter*> const &filters = {})

inline virtual std::optional<*expression_value_t*> **calculate**(*Symbol* *symbol) const override

inline virtual void **dump**(std::ostream &out) const override

Private Members

std::vector<*PatternMatchFilter*> **filters**

std::vector<*MaskedByte*> **needle**

expression_value_t **offset**

class *sinker*::**PatternMatchFilter**

Public Functions

inline **PatternMatchFilter**(const *Module* *module = nullptr, std::optional<std::string> const &section_name = {})

inline const *Module* ***get_module**() const

inline std::optional<std::string> const &**get_section_name**() const

Private Members

const *Module* ***module**

std::optional<std::string> **section_name**

class *sinker*::**PatternMatchFragment**

Subclassed by *sinker::PatternMatchExact*, *sinker::PatternMatchMask*, *sinker::PatternMatchWildcard*

Public Functions

inline virtual **~PatternMatchFragment**()

virtual void ***search**(void *begin, void *end) const = 0

virtual bool **begins_with**(void *begin, void *end) const = 0

virtual bool **collision**(void *address) const = 0

virtual std::size_t **size**() const = 0

virtual *PatternMatchType* **type**() const = 0

class *sinker*::**PatternMatchMask** : private *sinker*::*PatternMatchFragment*

Public Functions

inline **PatternMatchMask**(std::vector<*MaskedByte*> const &value)

inline virtual void ***search**(void *begin, void *end) const override

inline virtual bool **begins_with**(void *begin, void *end) const override

inline virtual std::size_t **size**() const override

inline virtual *PatternMatchType* **type**() const override

inline virtual bool **collision**(void *address) const

Private Members

std::vector<*MaskedByte*> **value**

class *sinker*::**PatternMatchNeedle**

Public Functions

inline **PatternMatchNeedle**(std::vector<*MaskedByte*> const &needle)

inline void ***search**(void *begin, void *end) const

inline virtual bool **collision**(void *address) const

Private Members

std::vector<std::unique_ptr<*PatternMatchFragment*>> **fragments** = {}

std::size_t **size** = 0

std::size_t **offset** = 0

std::size_t **index** = 0

class *sinker*::**PatternMatchWildcard** : private *sinker*::*PatternMatchFragment*

Public Functions

inline **PatternMatchWildcard**(std::size_t size)

inline virtual void ***search**(void *begin, void *end) const override

inline virtual bool **begins_with**(void *begin, void *end) const override

inline virtual std::size_t **size**() const override

inline virtual *PatternMatchType* **type**() const override

inline virtual bool **collision**(void *address) const

Private Members

std::size_t **s**

template<std::size_t **S** = 32, std::uint8_t **C** = 0xEF, bool **D** = true>

class *sinker*::**StackCheck**

Public Functions

StackCheck()

bool **good**() const

~StackCheck()

Private Members

std::uint8_t **buffer**[*S*]

class *sinker*::**Symbol** : public *sinker*::*Attributable*

Public Functions

Symbol(const *Symbol*&) = delete

Symbol &**operator=**(const *Symbol*&) = delete

Symbol(*Symbol*&&) = default

Symbol &**operator=**(*Symbol* &&mE) = default

inline std::string const &**get_name**() const

template<typename **T**>
std::optional<*T*> **calculate_address**()

template<typename **T**>

std::optional<*T*> **get_cached_calculated_address**() const

Module ***get_module**() const

void **add_address**(*identifier_set_t* const &variant_set, std::shared_ptr<*Expression*> expression)

void **dump**(std::ostream &out) const

void **dump_def**(std::ostream &out) const

void **add_tag**(std::string const &tag)

Private Functions

inline **Symbol**(std::string const &name, std::string const &type, *Module* *module)

Private Members

std::optional<*expression_value_t*> **cached_calculated_address**

std::string **name**

std::string **type**

Module ***module**

std::vector<std::pair<*identifier_set_t*, std::shared_ptr<*Expression*>>> **addresses**

identifier_set_t **tags**

Friends

friend class Module

class *sinker*::**SymbolExpression** : private *sinker*::*Expression*

Public Functions

inline **SymbolExpression**(*Symbol* *symbol)

inline virtual std::optional<*expression_value_t*> **calculate**(*Symbol* *_symbol) const override

inline virtual void **dump**(std::ostream &out) const override

Private Members

Symbol ***symbol**

class *sinker*::**Transaction**

Public Functions

inline **Transaction**()

inline void **add**(*Action* *action)

inline long **commit**()

Private Members

std::vector<*Action**> **actions**

class *sinker*::**UnaryOperatorExpression** : private *sinker*::*Expression*

Public Functions

inline **UnaryOperatorExpression**(std::shared_ptr<*Expression*> expression, *UnaryOperator* unary_operator)

inline virtual std::optional<*expression_value_t*> **calculate**(*Symbol* *symbol) const override

inline virtual void **dump**(std::ostream &out) const override

Private Members

std::shared_ptr<*Expression*> **expression**

UnaryOperator **unary_operator**

class *sinker*::**Uninstallable**

Subclassed by *sinker::Detour< T >*, *sinker::Patch< T >*, *sinker::Patch< T[N]>*

Public Functions

virtual void **uninstall**() = 0

namespace **sinker**

Typedefs

typedef unsigned long long **expression_value_t**

typedef std::variant<*expression_value_t*, bool, std::string> **attribute_value_t**

typedef std::set<std::string> **identifier_set_t**

Enums

enum class **Language**

Values:

enumerator **SINKER**

enumerator **SOURCE_CODE**

enum class **UnaryOperator**

Values:

enumerator **PARENTHESES**

enumerator **INDIRECTION**

enumerator **RELOCATION**

enumerator **BITWISE_NOT**

enum class **BinaryOperator**

Values:

enumerator **ADDITION**

enumerator **SUBTRACTION**

enumerator **MULTIPLICATION**

enumerator **INTEGER_DIVISION**

enumerator **MODULO**

enumerator **BITWISE_AND**

enumerator **BITWISE_OR**

enumerator **BITWISE_XOR**

enumerator **BITWISE_SHIFT_LEFT**

enumerator **BITWISE_SHIFT_RIGHT**

enumerator **ARRAY_SUBSCRIPT**

enumerator **POINTER_PATH**

enum class **PatternMatchType**

Values:

enumerator **EXACT**

enumerator **MASK**

enumerator **WILDCARD**

enumerator **COUNT**

Functions

std::ostream &**operator<<**(std::ostream &out, *attribute_value_t* const &attribute_value)

std::ostream &**operator<<**(std::ostream &os, *Expression* const &expression)

std::ostream &**operator<<**(std::ostream &os, *Context* const &context)

std::ostream &**operator<<**(std::ostream &os, *Symbol* const &symbol)

std::ostream &**operator<<**(std::ostream &os, *Module* const &module)

inline std::optional<*expression_value_t*> **CheckedDereference**(*expression_value_t* value)

file **sinker.hpp**

#include <>#include <>#include <>#include <>#include <>#include <>#include <>#include <>#include <>#include <>#include <>#include <>#include <>#include <>#include <>#include <>#include <>#include “”

Defines

PROPAGATE_UNRESOLVED(x)

dir
/home/docs/checkouts/readthedocs.org/user_builds/sinker/checkouts/latest/sinker/include

dir /home/docs/checkouts/readthedocs.org/user_builds/sinker/checkouts/latest/sinker

dir /home/docs/checkouts/readthedocs.org/user_builds/sinker/checkouts/latest/sinker/include/sinker

CHAPTER

FOUR

CMAKE INTEGRATION

4.1 Adding Sinker to Your CMake Project

The Sinker source code can be included in your project via a Git submodule or via CMake's `FetchContent` module.

4.1.1 Git Submodule

First, run the command

```
git submodule add https://github.com/widberg/sinker.git
```

then add the following to your `CMakeLists.txt`

```
add_subdirectory(sinker)
```

4.1.2 FetchContent

If you don't want to use a Git submodule then add the following to your `CMakeLists.txt`

```
Include(FetchContent)

FetchContent_Declare(
    Sinker
    GIT_REPOSITORY https://github.com/widberg/sinker.git
    GIT_TAG        some_commit_hash
)

FetchContent_MakeAvailable(Sinker)
```

4.2 Sinker Compiler Target

You can define a target that runs the Sinker compiler on a list of input files and outputs a `.skr` file and a `.def` file.

```
add_sinker_target(my_sinker_target
    INPUT
        my_sinker_modules.skr
        main.cpp
        instrument.cpp
        my_other_source.cpp
    OUTPUT ${CMAKE_CURRENT_BINARY_DIR}/my_sinker_target.skr
    DEFINITIONS ${CMAKE_CURRENT_BINARY_DIR}/include/my_sinker_target.def
)
```

The list of all sources for a target can be retrieved with `get_target_property`

```
get_target_property(MY_TARGET_SOURCES my_target SOURCES)
add_sinker_target(my_sinker_target
    INPUT
        ${MY_TARGET_SOURCES}
    OUTPUT ${CMAKE_CURRENT_BINARY_DIR}/my_sinker_target.skr
    DEFINITIONS ${CMAKE_CURRENT_BINARY_DIR}/include/my_sinker_target.def
)
```

To access the `.def` file with includes in my_target, you can use the following:

```
target_include_directories(my_target
    PRIVATE ${CMAKE_CURRENT_BINARY_DIR}/include
)
```

4.3 Sinker Runtime Library

The Sinker runtime library can be linked with your target by adding the following to your `CMakeLists.txt`

```
target_link_libraries(my_target
    PRIVATE sinker
)
```

INDEX