API Collection

Macros

Generate boilerplate code and perform other compile-time operations.

Topics

Essentials

Applying Macros

Use macros to generate repetitive code at compile time.

Getting Source Location Information

`macro file<T>() -> T`

Produces the path to the file in which it appears.

`macro fileID<T>() -> T`

Produces a unique identifier for the source file in which the macro appears.

`macro filePath<T>() -> T`

Produces the complete path to the file in which the macro appears.

`macro function<T>() -> T`

Produces the name of the declaration in which it appears.

`macro line<T>() -> T`

Produces the line number on which it appears.

`macro column<T>() -> T`

Produces the column number in which the macro begins.

Generating Compile-Time Diagnostics

`macro warning(String)`

Produces the given warning message during compilation.

`macro error(String)`

Emits the given message as a fatal error and terminates the compilation process.

Writing Custom Macros

`macro externalMacro<T>(module: String, type: String) -> T`

Specifies the module and type name for a macro’s implementation.

Accessing the Dynamic Shared Object Handle

`macro dsohandle() -> UnsafeRawPointer`

Produces the dynamic shared object (DSO) handle in use where the macro appears.

See Also

Programming Tasks

Input and Output

Print values to the console, read from and write to text streams, and use command line arguments.

Debugging and Reflection

Fortify your code with runtime checks, and examine your values’ runtime representation.

Concurrency

Perform asynchronous and parallel operations.

Key-Path Expressions

Use key-path expressions to access properties dynamically.

Manual Memory Management

Allocate and manage memory manually.

Type Casting and Existential Types

Perform casts between types or represent values of any type.

C Interoperability

Use imported C types or call C variadic functions.

Operator Declarations

Work with prefix, postfix, and infix operators.