nsbaci

0.2

Generated by Doxygen 1.16.1

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 nsbaci Namespace Reference

Root namespace for the nsbaci application.

Namespaces

- namespace factories

  Factories namespace for nsbaci.
- namespace compiler

  Compiler namespace containing all compilation-related stuff.
- namespace services

  Services namespace containing all backend service implementations.
- namespace types

  Type definitions namespace for nsbaci (runtime-specific).

Classes

- class Controller

  Central coordinator between UI and backend services.
- class Error

  Represents an error with a message and optional code.
- struct UIError

  UI-ready error representation for display in dialogs.
- struct BaseResult

  Base result structure for all service operations.

5.1.1 Detailed Description

Root namespace for the nsbaci application.

Main namespace for the nsbaci interpreter application.

Contains all components of the BACI concurrent programming interpreter including the controller, services, UI components, and type definitions.

5.2 nsbaci::compiler Namespace Reference

Compiler namespace containing all compilation-related stuff.

Classes

- struct CompilerResult

  Result of a compilation operation.
- class Compiler

  Abstract base class for all compilers.
- struct Instruction

  Represents a single instruction in the virtual machine.
- class Lexer

  Flex-based lexer for BACI source code.
- class NsbaciCompiler

  nsbaci compiler implementation using flex and bison.

Typedefs

- using Operand

  Operand types that an instruction can have.
- using **InstructionStream** = std::vector<Instruction>

  Vector of instructions representing a compiled program.

Enumerations

- enum class Opcode : uint8_t {
  LoadValue , **LoadAddress** , **LoadIndirect** , **LoadBlock** ,
  Store , **StoreKeep** , **PushLiteral** , **Index** ,
  CopyBlock , **ValueAt** , **MarkStack** , **UpdateDisplay** ,
  Add , **Sub** , **Mult** , **Div** ,
  Mod , **Negate** , **Complement** , **And** ,
  Or , **TestEQ** , **TestNE** , **TestLT** ,
  TestLE , **TestGT** , **TestGE** , **TestEqualKeep** ,
  Jump , **JumpZero** , **Call** , **ShortCall** ,
  ShortReturn , **ExitProc** , **ExitFunction** , **Halt** ,
  BeginFor , **EndFor** , **Cobegin** , **Coend** ,
  Create , **Suspend** , **Revive** , **WhichProc** ,
  Wait , **Signal** , **StoreSemaphore** , **EnterMonitor** ,
  ExitMonitor , **CallMonitorInit** , **ReturnMonitorInit** , **WaitCondition** ,
  SignalCondition , **Empty** , **Read** , **ReadIn** ,
  Write , **WriteIn** , **WriteString** , **WriteRawString** ,
  EolEof , **Sprintf** , **Sscanf** , **CopyString** ,
  CopyRawString , **ConcatString** , **ConcatRawString** , **CompareString** ,
  CompareRawString , **LengthString** , **MoveTo** , **MoveBy** ,
  ChangeColor , **MakeVisible** , **Remove** , **Random** ,
  Test , **_Count** }

  Opcodes for the BACI virtual machine instruction set.

Functions

- const char * opcodeName (Opcode op)

  Get the string name of an opcode (for debugging/display).

5.2.1 Detailed Description

Compiler namespace containing all compilation-related stuff.

Compiler namespace for nsbaci.

5.2.2 Typedef Documentation

5.2.2.1 Operand

```
using nsbaci::compiler::Operand
```

Initial value:

```
std::variant<std::monostate,
                          int32_t,
                          uint32_t,
                          std::string
                          >
```

Operand types that an instruction can have.

5.2.3 Function Documentation

5.2.3.1 opcodeName()

```
const char * nsbaci::compiler::opcodeName (
            Opcode op)
```

Get the string name of an opcode (for debugging/display).

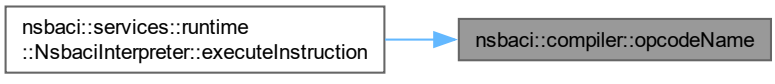
Parameters

op	The opcode to convert.

Returns

String representation of the opcode.

Here is the caller graph for this function:



5.3 nsbaci::factories Namespace Reference

Factories namespace for nsbaci.

Classes

- struct NsbaciCompiler
- class CompilerServiceFactory

  Factory for creating CompilerService instances.
- struct DefaultDrawingBackend
- class DrawingServiceFactory

  Factory for creating DrawingService instances.
- struct DefaultFileSystem
- class FileServiceFactory

  Factory for creating FileService instances.
- struct NsbaciRuntime
- class RuntimeServiceFactory

  Factory for creating RuntimeService instances.

Variables

- constexpr NsbaciCompiler **nsbaciCompiler** {}
- constexpr DefaultDrawingBackend **defaultDrawingBackend** {}
- constexpr DefaultFileSystem **defaultFileSystem** {}
- constexpr NsbaciRuntime **nsbaciRuntime** {}

5.3.1 Detailed Description

Factories namespace for nsbaci.

5.4 nsbaci::services Namespace Reference

Services namespace containing all backend service implementations.

Namespaces

- namespace runtime

  Runtime services namespace for nsbaci.

Classes

- class CompilerService

  service for compiling nsbaci (or maybe other stuff in the future) source code.
- class DrawingService

  Adapter service for graphical output backends.
- class FileService

  Service for handling file system operations on BACI source files.
- struct RuntimeResult

  Result of a runtime operation (step, run, etc.).
- class RuntimeService

  Service that manages program execution.

Enumerations

- enum class RuntimeState { Idle , Running , Paused , Halted }

  Possible states of the runtime service.

5.4.1 Detailed Description

Services namespace containing all backend service implementations.

Services namespace for nsbaci.

This namespace contains service classes that encapsulate specific functionality areas such as file I/O, compilation, and runtime execution.

5.4.2 Enumeration Type Documentation

5.4.2.1 RuntimeState

```
enum class nsbaci::services::RuntimeState  [strong]
```

Possible states of the runtime service.

Represents the lifecycle states of program execution:

- Idle: Initial state, no program loaded or execution completed
- Running: Active execution in progress
- Paused: Execution suspended, can be resumed or stepped
- Halted: Program has finished (reached Halt instruction)

Enumerator

Idle	No program loaded or ready to start.
Running	Program is actively executing.
Paused	Execution paused, can step or continue.
Halted	Program has finished execution.

5.5 nsbaci::services::runtime Namespace Reference

Runtime services namespace for nsbaci.

Classes

- class Interpreter

  Executes instructions for threads within a program context.
- class NsbaciInterpreter

  BACI-specific implementation of the Interpreter.
- class Program

  Represents a compiled program ready for execution.
- class NsbaciScheduler

  BACI-specific implementation of the Scheduler.
- class Scheduler

  Manages thread scheduling and state transitions.
- class Thread

  Represents a thread in the runtime service.

Typedefs

- using **OutputCallback** = std::function<void(const std::string&)>

  Callback type for output operations.
- using **InputRequestCallback** = std::function<void(const std::string&)>

  Callback type for input requests.

5.5.1 Detailed Description

Runtime services namespace for nsbaci.

5.6 nsbaci::types Namespace Reference

Type definitions namespace for nsbaci (runtime-specific).

Classes

- struct SymbolInfo

  Information about a variable/symbol.
- struct ErrorBase

  Base structure containing common error properties.
- struct CompileError

  Error payload for compilation errors.
- struct SaveError

  Error payload for file save errors.
- struct LoadError

  Error payload for file load errors.
- struct RuntimeError

  Error payload for runtime errors.
- struct Address

  Represents a memory address in the runtime.

Typedefs

- using **StackValue** = int32_t

  Stack value type (can hold int or address).
- using **Stack** = std::vector<StackValue>

  Runtime stack.
- using **Memory** = std::vector<int32_t>

  Memory block for runtime data.
- using **ThreadQueue** = std::queue<nsbaci::services::runtime::Thread>

  Queue of threads for scheduler operations.
- using **VarName** = std::string

  Type alias for variable names.
- using **MemoryAddr** = uint32_t

  Type alias for memory addresses.
- using **SymbolTable** = std::unordered_map<VarName, SymbolInfo>

  Lookup table mapping variable names to their symbol info.
- using **ErrMessage** = std::string
- using ErrorPayload

  Variant type for all possible error payloads.
- using **Text** = std::string

  Alias for text content (file contents, source code, etc.).
- using **File** = fs::path

  Alias for file system paths.
- using **ThreadID** = unsigned long long int
- using **Priority** = unsigned long int

Enumerations

- enum class ErrSeverity { **Warning** , **Error** , **Fatal** }

  Severity levels for errors.
- enum class ErrType {
  emptyPath , **invalidPath** , **invalidExtension** , **directoryNotFound** ,
  fileNotFound , **notARegularFile** , **permissionDenied** , **openFailed** ,
  readFailed , **writeFailed** , **compilationError** , **unknown** }

  Types of errors that can occur in the application.
- enum class ThreadState {
  Ready , Running , Blocked , Waiting ,
  IO , Terminated }

5.6.1 Detailed Description

Type definitions namespace for nsbaci (runtime-specific).

Type definitions namespace for nsbaci.

5.6.2 Typedef Documentation

5.6.2.1 ErrorPayload

```
using nsbaci::types::ErrorPayload
```

Initial value:

```
std::variant<SaveError, LoadError, CompileError, RuntimeError>
```

Variant type for all possible error payloads.

Can be used to create a more specific error message in the controller, for example when an error is of type compile↩ Error, the controller can specify the line, col, what might have happened...

5.6.3 Enumeration Type Documentation

5.6.3.1 ThreadState

```
enum class nsbaci::types::ThreadState  [strong]
```

Enumerator

Ready	Thread is ready to run.
Running	Thread is currently executing.
Blocked	Thread is blocked (e.g., waiting on semaphore).
Waiting	Thread is waiting for I/O.
IO	Thread is performing I/O.
Terminated	Thread has finished execution.

Chapter 6

Class Documentation

6.1 nsbaci::types::Address Struct Reference

Represents a memory address in the runtime.

`#include <runtimeTypes.h>`

Public Member Functions

- **Address** (unsigned long long int val)
- bool **operator==** (const Address &other) const
- bool **operator!=** (const Address &other) const
- bool **operator**< (const Address &other) const
- bool **operator**> (const Address &other) const
- bool **operator**<**=** (const Address &other) const
- bool **operator**>**=** (const Address &other) const
- Address **operator+** (const Address &other) const
- Address **operator-** (const Address &other) const
- Address **operator+** (unsigned long long int offset) const
- Address **operator-** (unsigned long long int offset) const
- Address & **operator+=** (const Address &other)
- Address & **operator-=** (const Address &other)
- Address & **operator+=** (unsigned long long int offset)
- Address & **operator-=** (unsigned long long int offset)
- Address & **operator++** ()
- Address **operator++** (int)
- Address & **operator--** ()
- Address **operator--** (int)
- **operator unsigned long long int** () const

Public Attributes

- unsigned long long int **value**

6.1.1 Detailed Description

Represents a memory address in the runtime.

It is a wrapper for a raw type, so it defines operators to work directly with the wrapper instead of the member

The documentation for this struct was generated from the following file:

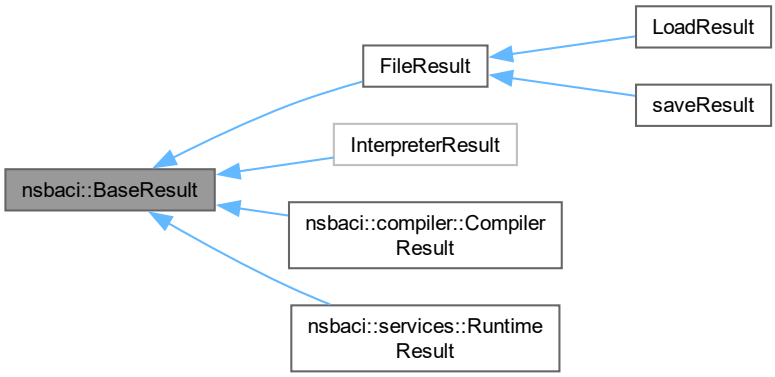
- runtimeTypes.h

6.2 nsbaci::BaseResult Struct Reference

Base result structure for all service operations.

`#include <baseResult.h>`

Inheritance diagram for nsbaci::BaseResult:



Public Member Functions

- **BaseResult** ()

  Default constructor creates a successful result.
- BaseResult (std::vector< nsbaci::Error > errs)

  Constructs a result from a vector of errors.
- BaseResult (nsbaci::Error error)

  Constructs a failed result from a single error.
- **BaseResult** (BaseResult &&) noexcept=default
- BaseResult & **operator=** (BaseResult &&) noexcept=default
- **BaseResult** (const BaseResult &)=default
- BaseResult & **operator=** (const BaseResult &)=default

Public Attributes

- bool **ok**

  True if the operation succeeded.
- std::vector< nsbaci::Error > **errors**

  Errors encountered (empty if ok is true).

6.2.1 Detailed Description

Base result structure for all service operations.

BaseResult provides a common interface for service operation results, encapsulating success/failure status and any associated error information. All service-specific result types should inherit from this base to ensure consistent error handling patterns throughout the application.

Invariant

If ok is false, errors vector contains at least one error.

If ok is true, errors vector is empty.

Usage example:

```
BaseResult result = someService.doOperation();
if (!result.ok) {
  for (const auto& err : result.errors) {
    handleError(err);
  }
}
```

6.2.2 Constructor & Destructor Documentation

6.2.2.1 BaseResult() [1/2]

```
nsbaci::BaseResult::BaseResult (
            std::vector< nsbaci::Error > errs)  [inline], [explicit]
```

Constructs a result from a vector of errors.

If the error vector is empty, the result is considered successful.

Parameters

errs	Vector of errors encountered during the operation.

6.2.2.2 BaseResult() [2/2]

```
nsbaci::BaseResult::BaseResult (
            nsbaci::Error error)  [inline], [explicit]
```

Constructs a failed result from a single error.

Parameters

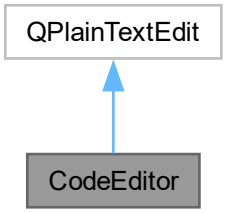
error	The error that caused the operation to fail.

The documentation for this struct was generated from the following file:

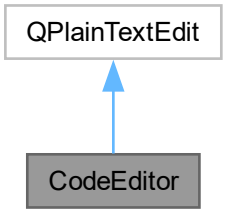
- baseResult.h

6.3 CodeEditor Class Reference

Inheritance diagram for CodeEditor:



Collaboration diagram for CodeEditor:



Public Member Functions

- **CodeEditor** (QWidget *parent=nullptr)
- void **lineNumberAreaPaintEvent** (QPaintEvent *event)
- int **lineNumberAreaWidth** ()

Protected Member Functions

- void **resizeEvent** (QResizeEvent ∗event) override

The documentation for this class was generated from the following files:

- codeeditor.h
- codeeditor.cpp

6.4 nsbaci::types::CompileError Struct Reference

Error payload for compilation errors.

`#include <errorTypes.h>`

Public Attributes

- int **line** = 0
- int **column** = 0

6.4.1 Detailed Description

Error payload for compilation errors.

The documentation for this struct was generated from the following file:

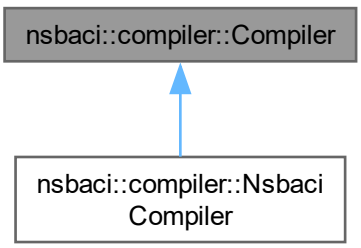
- errorTypes.h

6.5 nsbaci::compiler::Compiler Class Reference

Abstract base class for all compilers.

`#include <compiler.h>`

Inheritance diagram for nsbaci::compiler::Compiler:



Public Member Functions

- **Compiler** ()=default

  Default constructor.
- virtual ~**Compiler** ()=default

  Virtual destructor.
- virtual CompilerResult compile (const std::string &source)=0

  Compiles source code from a string.
- virtual CompilerResult compile (std::istream &input)=0

  Compiles source code from an input stream.

6.5.1 Detailed Description

Abstract base class for all compilers.

The Compiler interface defines the contract for compiling source code into p-code instructions. Implementations handle lexical analysis, parsing, semantic analysis, and code generation.

The compilation process produces:

- An instruction stream (p-code) for the virtual machine
- A symbol table mapping variable names to their types and addresses
- Compilation errors, if there are

Subclasses must implement both compile() overloads to support compilation from both strings and input streams.

6.5.2 Member Function Documentation

6.5.2.1 compile() [1/2]

```
virtual CompilerResult nsbaci::compiler::Compiler::compile (
            const std::string & source)  [pure virtual]
```

Compiles source code from a string.

Performs full compilation including lexical analysis, parsing, semantic analysis, and p-code generation.

Parameters

source	The BACI source code to compile.

Returns

CompilerResult containing instructions on success, or errors on failure.

Implemented in nsbaci::compiler::NsbaciCompiler.

6.5.2.2 compile() [2/2]

```
virtual CompilerResult nsbaci::compiler::Compiler::compile (
```

Parameters

input	The input stream containing BACI source code.

Returns

CompilerResult containing instructions on success, or errors on failure.

Implemented in nsbaci::compiler::NsbaciCompiler.

The documentation for this class was generated from the following file:

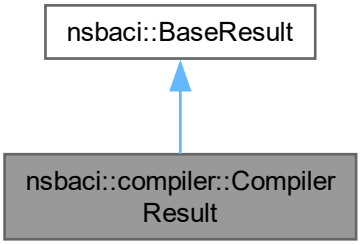
- compiler.h

6.6 nsbaci::compiler::CompilerResult Struct Reference

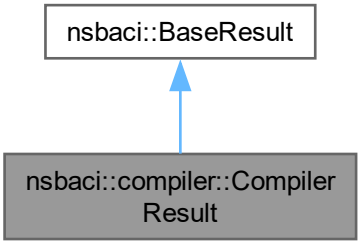
Result of a compilation operation.

`#include <compiler.h>`

Inheritance diagram for nsbaci::compiler::CompilerResult:



Collaboration diagram for nsbaci::compiler::CompilerResult:



Public Member Functions

- **CompilerResult** ()

  Default constructor creates a successful empty result.
- CompilerResult (std::vector< nsbaci::Error > errs)

  Constructs a failed result from a vector of errors.
- CompilerResult (nsbaci::Error error)

  Constructs a failed result from a single error.
- **CompilerResult** (CompilerResult &&) noexcept=default
- CompilerResult & **operator=** (CompilerResult &&) noexcept=default
- **CompilerResult** (const CompilerResult &)=default
- CompilerResult & **operator=** (const CompilerResult &)=default

Public Member Functions inherited from nsbaci::BaseResult

- **BaseResult** ()

  Default constructor creates a successful result.
- BaseResult (std::vector< nsbaci::Error > errs)

  Constructs a result from a vector of errors.
- BaseResult (nsbaci::Error error)

  Constructs a failed result from a single error.
- **BaseResult** (BaseResult &&) noexcept=default
- BaseResult & **operator=** (BaseResult &&) noexcept=default
- **BaseResult** (const BaseResult &)=default
- BaseResult & **operator=** (const BaseResult &)=default

Public Attributes

- InstructionStream **instructions**

  Generated p-code instructions.
- nsbaci::types::SymbolTable **symbols**

  Symbol table from compilation.

Public Attributes inherited from nsbaci::BaseResult

- bool **ok**

  True if the operation succeeded.
- std::vector< nsbaci::Error > **errors**

  Errors encountered (empty if ok is true).

6.6.1 Detailed Description

Result of a compilation operation.

Contains the outcome of a compilation attempt including success/failure status, any error messages, and on success, the generated instruction stream and symbol table.

Note

On failure, the instructions and symbols fields should not be used.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 CompilerResult() [1/2]

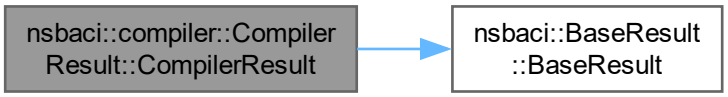
```
nsbaci::compiler::CompilerResult::CompilerResult (
            std::vector< nsbaci::Error > errs)  [inline], [explicit]
```

Constructs a failed result from a vector of errors.

Parameters

errs	Vector of compilation errors.

Here is the call graph for this function:



6.6.2.2 CompilerResult() [2/2]

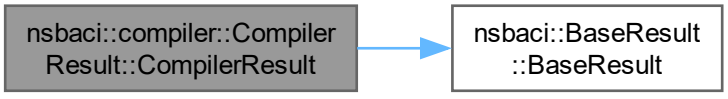
```
nsbaci::compiler::CompilerResult::CompilerResult (
            nsbaci::Error error)  [inline], [explicit]
```

Constructs a failed result from a single error.

Parameters

error	The compilation error.

Here is the call graph for this function:



The documentation for this struct was generated from the following file:

- compiler.h

6.7 nsbaci::services::CompilerService Class Reference

service for compiling nsbaci (or maybe other stuff in the future) source code.

`#include <compilerService.h>`

Public Member Functions

- CompilerService (std::unique_ptr< nsbaci::compiler::Compiler > c=std::make_unique< nsbaci::compiler::NsbaciCompiler >())

  Constructs the CompilerService with a compiler implementation.
- ~**CompilerService** ()=default

  Default destructor.
- **CompilerService** (const CompilerService &)=delete
- CompilerService & **operator=** (const CompilerService &)=delete
- **CompilerService** (CompilerService &&)=default
- CompilerService & **operator=** (CompilerService &&)=default
- nsbaci::compiler::CompilerResult compile (nsbaci::types::Text raw)

  Compiles nsbaci source code into p-code instructions.
- bool hasProgramReady () const

  Checks if a compiled program is available for execution.
- nsbaci::compiler::InstructionStream takeInstructions ()

  Retrieves and releases ownership of the compiled instructions.
- nsbaci::types::SymbolTable takeSymbols ()

  Retrieves and releases ownership of the symbol table.

6.7.1 Detailed Description

service for compiling nsbaci (or maybe other stuff in the future) source code.

The service functions like the following:

1. Call compile() with source code
2. Check hasProgramReady() to verify success
3. Call takeInstructions() and takeSymbols() to retrieve compiled data

After taking the instructions, the program is no longer considered ready until a new successful compilation occurs.

Usage example:

```
CompilerService cs;
auto result = cs.compile(sourceCode);
if (result.ok && cs.hasProgramReady()) {
  auto instructions = cs.takeInstructions();
  auto symbols = cs.takeSymbols();
  // Load into runtime...
}
```

6.7.2 Constructor & Destructor Documentation

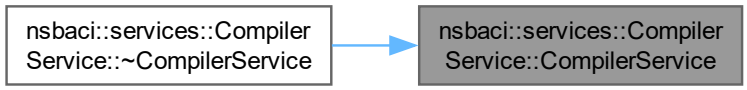
6.7.2.1 CompilerService()


```
nsbaci::services::CompilerService::CompilerService (
            std::unique_ptr< nsbaci::compiler::Compiler > c = std::make_unique<nsbaci::compiler::NsbaciCompi
```

Parameters

c	Unique pointer to a Compiler implementation. Defaults to NsbaciCompiler.

Here is the caller graph for this function:



6.7.3 Member Function Documentation

6.7.3.1 compile()

```
nsbaci::compiler::CompilerResult nsbaci::services::CompilerService::compile (
            nsbaci::types::Text raw)
```

Compiles nsbaci source code into p-code instructions.

Passes the source code to the underlying compiler and stores the results if compilation succeeds. The compiled instructions and symbols can then be retrieved using takeInstructions() and takeSymbols().

Parameters

raw	The nsbaci source code to compile.

Returns

CompilerResult containing success status and any compilation errors.

6.7.3.2 hasProgramReady()

```
bool nsbaci::services::CompilerService::hasProgramReady () const
```

Checks if a compiled program is available for execution.

Returns true after a successful compile() call and before takeInstructions() is called. Used to verify that valid instructions are available before attempting to load them into the runtime.

Returns

True if compiled instructions are available, false otherwise.

6.7.3.3 takeInstructions()

```
nsbaci::compiler::InstructionStream nsbaci::services::CompilerService::takeInstructions ()
```

Retrieves and releases ownership of the compiled instructions.

Moves the instruction stream out of the service. After this call, hasProgramReady() will return false until a new successful compilation.

Returns

The compiled instruction stream.

Warning

Only call when hasProgramReady() returns true.

6.7.3.4 takeSymbols()

```
nsbaci::types::SymbolTable nsbaci::services::CompilerService::takeSymbols ()
```

Retrieves and releases ownership of the symbol table.

Moves the symbol table out of the service. The symbol table contains information about all declared variables including names, types, and memory addresses.

Returns

The symbol table from the last successful compilation.

The documentation for this class was generated from the following files:

- compilerService.h
- compilerService.cpp

6.8 nsbaci::factories::CompilerServiceFactory Class Reference

Factory for creating CompilerService instances.

```
#include <compilerServiceFactory.h>
```

Static Public Member Functions

- static nsbaci::services::CompilerService **createService** (NsbaciCompiler t)

6.8.1 Detailed Description

Factory for creating CompilerService instances.

The documentation for this class was generated from the following files:

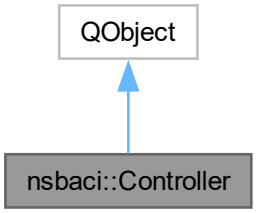
- compilerServiceFactory.h
- compilerServiceFactory.cpp

6.9 nsbaci::Controller Class Reference

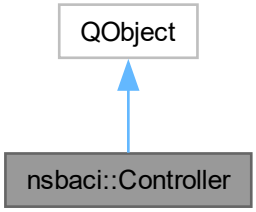
Central coordinator between UI and backend services.

`#include <controller.h>`

Inheritance diagram for nsbaci::Controller:



Collaboration diagram for nsbaci::Controller:



Public Slots

- void onSaveRequested (nsbaci::types::File file, nsbaci::types::Text contents)

  Handles a request to save source code to a file.
- void onOpenRequested (nsbaci::types::File file)

  Handles a request to open and load a source file.
- void onCompileRequested (nsbaci::types::Text contents)

  Handles a request to compile source code.
- void onRunRequested ()

  Handles a request to load and prepare a compiled program for execution.
- void onStepRequested ()

  Executes a single instruction across any ready thread.
- void onStepThreadRequested (nsbaci::types::ThreadID threadId)

  Executes a single instruction on a specific thread.
- void onRunContinueRequested ()

  Starts or resumes continuous execution mode.
- void onPauseRequested ()

  Pauses continuous execution.
- void onResetRequested ()

  Resets the runtime to initial state.
- void onStopRequested ()

  Stops execution and unloads the program.
- void onInputProvided (const QString &input)

  Provides user input to the runtime.

Signals

- void saveFailed (std::vector< UIError > errors)

  Emitted when a file save operation fails.
- void **saveSucceeded** ()

  Emitted when a file save operation succeeds.
- void loadFailed (std::vector< UIError > errors)

  Emitted when a file load operation fails.
- void loadSucceeded (const QString &contents)

  Emitted when a file load operation succeeds.
- void compileFailed (std::vector< UIError > errors)

  Emitted when compilation fails.
- void compileSucceeded ()

  Emitted when compilation succeeds.
- void runStarted (const QString &programName)

  Emitted when a program is loaded and ready for execution.
- void runtimeStateChanged (bool running, bool halted)

  Emitted when the runtime execution state changes.
- void threadsUpdated (const std::vector< nsbaci::ui::ThreadInfo > &threads)

  Emitted when thread information needs to be updated in the UI.
- void variablesUpdated (const std::vector< nsbaci::ui::VariableInfo > &variables)

  Emitted when variable information needs to be updated in the UI.
- void outputReceived (const QString &output)

  Emitted when the runtime produces output (cout, writeln, etc.).
- void inputRequested (const QString &prompt)

  Emitted when the runtime needs user input (cin, read, etc.).

Public Member Functions

- Controller (nsbaci::services::FileService &&f, nsbaci::services::CompilerService &&c, nsbaci::services::RuntimeService &&r, nsbaci::services::DrawingService &&d, QObject *parent=nullptr)

  Constructs the Controller with all required services.
- ~Controller ()=default

  Default destructor.

6.9.1 Detailed Description

Central coordinator between UI and backend services.

The Controller class is a QObject that serves as the main application controller, implementing the MVC pattern. It receives user actions through Qt slots, processes them using the specific backend services, and communicates results back to the UI through Qt signals.

The controller owns instances of all required services:

- FileService: Handles file system operations
- CompilerService: Compiles NsBaci source code to p-code
- RuntimeService: Executes compiled programs with thread scheduling
- DrawingService: Not yet implemented, but will be used as graphical API in the future

Execution modes supported:

- Single-step execution: Execute one instruction at a time
- Continuous execution: Run program with timer-based batching
- Thread-specific stepping: Execute a single thread's instruction

Note

The controller uses a QTimer for continuous execution to maintain UI responsiveness during long-running programs.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Controller()

```
nsbaci::Controller::Controller (
            nsbaci::services::FileService && f,
            nsbaci::services::CompilerService && c,
            nsbaci::services::RuntimeService && r,
            nsbaci::services::DrawingService && d,
            QObject * parent = nullptr)  [explicit]
```

Constructs the Controller with all required services.

Takes ownership of the provided services via move semantics. Initializes the internal QTimer used for continuous execution mode.

Parameters

f	FileService instance for file operations.
c	CompilerService instance for compilation.
r	RuntimeService instance for program execution.
d	DrawingService instance for graphical output.
parent	Optional parent QObject for Qt memory management.

6.9.2.2 ~Controller()

```
nsbaci::Controller::~Controller ()  [default]
```

Default destructor.

The QTimer is automatically cleaned up through Qt's parent-child system.

6.9.3 Member Function Documentation

6.9.3.1 compileFailed

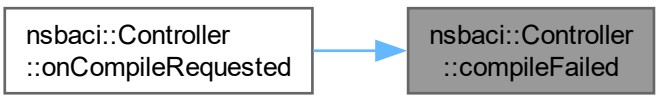
```
void nsbaci::Controller::compileFailed (
            std::vector< UIError > errors)  [signal]
```

Emitted when compilation fails.

Parameters

errors	List of compilation errors with line/column information.

Here is the caller graph for this function:

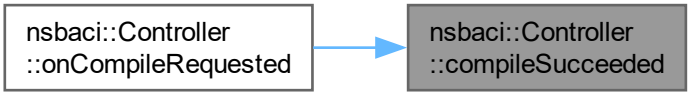


6.9.3.2 compileSucceeded

```
void nsbaci::Controller::compileSucceeded ()  [signal]
```

Emitted when compilation succeeds.

After this signal, the compiled program is ready to be loaded into the runtime via onRunRequested(). Here is the caller graph for this function:



6.9.3.3 inputRequested

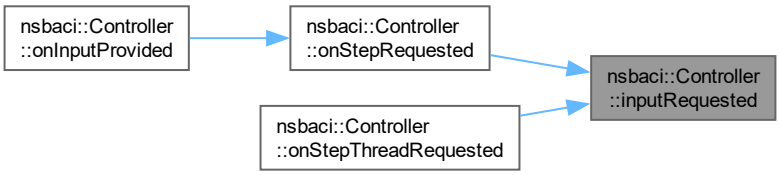
```
void nsbaci::Controller::inputRequested (
            const QString & prompt)  [signal]
```

Emitted when the runtime needs user input (cin, read, etc.).

Parameters

prompt	The prompt message to display to the user.

Here is the caller graph for this function:



6.9.3.4 loadFailed

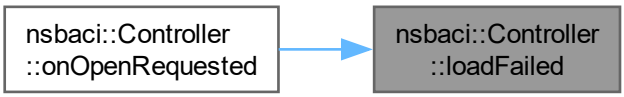
```
void nsbaci::Controller::loadFailed (
            std::vector< UIError > errors)  [signal]
```

Emitted when a file load operation fails.

Parameters

errors	List of errors describing what went wrong.

Here is the caller graph for this function:



6.9.3.5 loadSucceeded

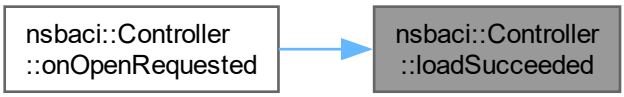
```
void nsbaci::Controller::loadSucceeded (
            const QString & contents)  [signal]
```

Emitted when a file load operation succeeds.

Parameters

contents	The loaded file contents as a QString.

Here is the caller graph for this function:



6.9.3.6 onCompileRequested

```
void nsbaci::Controller::onCompileRequested (
            nsbaci::types::Text contents)  [slot]
```

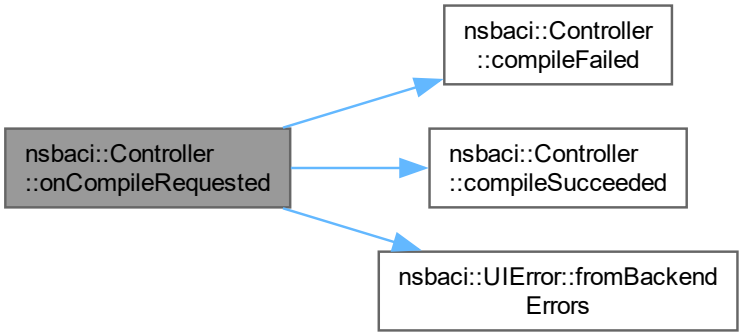
Handles a request to compile source code.

Compiles the provided source code into p-code instructions. On success, the instructions are stored in the CompilerService ready to be loaded into the runtime.

Parameters

contents	The BACI source code to compile.

Here is the call graph for this function:



6.9.3.7 onInputProvided

```
void nsbaci::Controller::onInputProvided (
            const QString & input)  [slot]
```

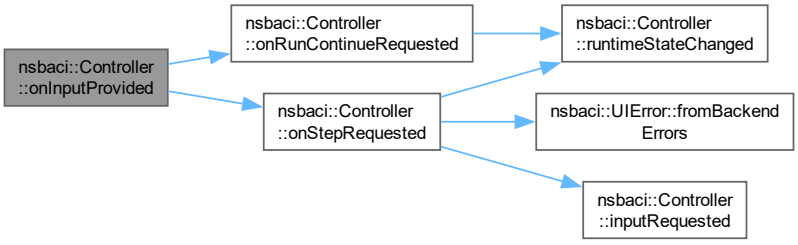
Provides user input to the runtime.

Called when the user enters input in response to an inputRequested signal. If the program was running continuously before the input request, execution is automatically resumed.

Parameters

input	The user-provided input string.

Here is the call graph for this function:



6.9.3.8 onOpenRequested

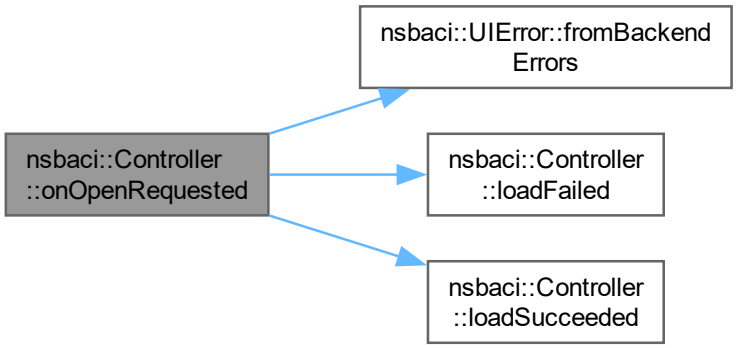
```
void nsbaci::Controller::onOpenRequested (
            nsbaci::types::File file)  [slot]
```

Handles a request to open and load a source file.

Parameters

file	The file path to load.

Here is the call graph for this function:

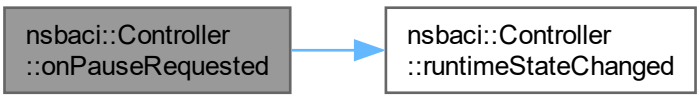


6.9.3.9 onPauseRequested

```
void nsbaci::Controller::onPauseRequested ()  [slot]
```

Pauses continuous execution.

Stops the execution timer but preserves program state for later resumption. Here is the call graph for this function:

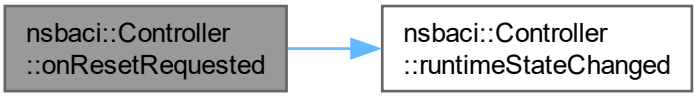


6.9.3.10 onResetRequested

```
void nsbaci::Controller::onResetRequested ()  [slot]
```

Resets the runtime to initial state.

Clears all thread state and memory but keeps the loaded program. The program can be re-run from the beginning. Here is the call graph for this function:

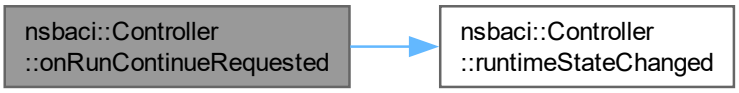


6.9.3.11 onRunContinueRequested

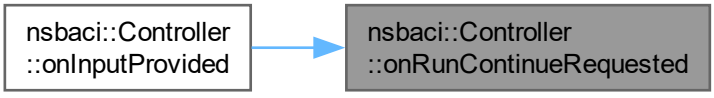
```
void nsbaci::Controller::onRunContinueRequested ()  [slot]
```

Starts or resumes continuous execution mode.

Begins timer-based execution where batches of instructions are executed periodically, allowing the UI to remain responsive. Here is the call graph for this function:



Here is the caller graph for this function:

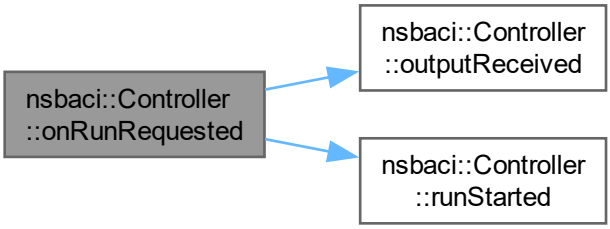


6.9.3.12 onRunRequested

```
void nsbaci::Controller::onRunRequested ()  [slot]
```

Handles a request to load and prepare a compiled program for execution.

Takes the compiled instructions and symbol table from the CompilerService and loads them into the Runtime↩ Service. Sets up the output callback for forwarding runtime output to the UI. Here is the call graph for this function:



6.9.3.13 onSaveRequested

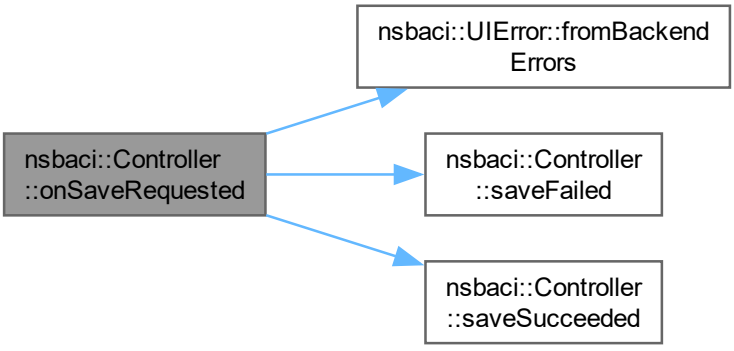
```
void nsbaci::Controller::onSaveRequested (
            nsbaci::types::File file,
            nsbaci::types::Text contents)  [slot]
```

Handles a request to save source code to a file.

Parameters

file	The target file path.
contents	The source code content to save.

Here is the call graph for this function:

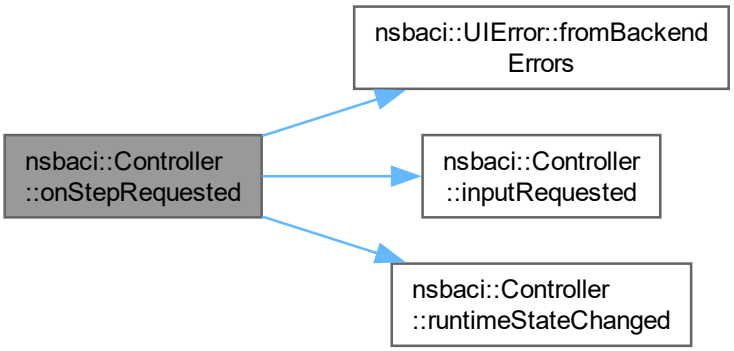


6.9.3.14 onStepRequested

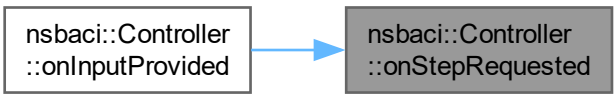
```
void nsbaci::Controller::onStepRequested ()  [slot]
```

Executes a single instruction across any ready thread.

The scheduler picks the next thread to run and executes one instruction. Updates the UI with new thread and variable states. Here is the call graph for this function:



Here is the caller graph for this function:



6.9.3.15 onStepThreadRequested

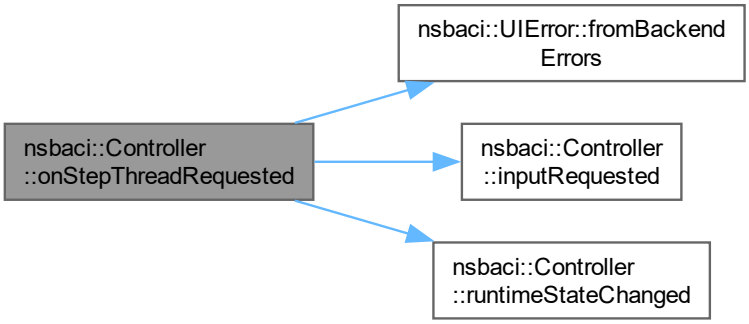
```
void nsbaci::Controller::onStepThreadRequested (
            nsbaci::types::ThreadID threadId)  [slot]
```

Executes a single instruction on a specific thread.

Parameters

threadId	The ID of the thread to step.

Here is the call graph for this function:



6.9.3.16 onStopRequested

```
void nsbaci::Controller::onStopRequested ()  [slot]
```

Stops execution and unloads the program.

Completely stops the runtime and marks no program as loaded.

6.9.3.17 outputReceived

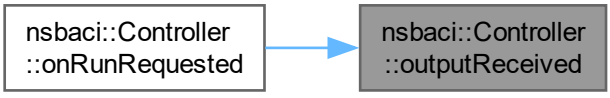
```
void nsbaci::Controller::outputReceived (
            const QString & output)  [signal]
```

Emitted when the runtime produces output (cout, writeln, etc.).

Parameters

output	The output string to display in the console.

Here is the caller graph for this function:



6.9.3.18 runStarted

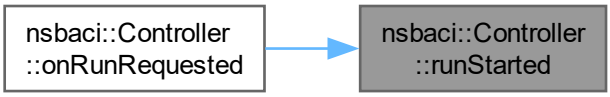
```
void nsbaci::Controller::runStarted (
            const QString & programName)  [signal]
```

Emitted when a program is loaded and ready for execution.

Parameters

programName	The name of the loaded program.

Here is the caller graph for this function:



6.9.3.19 runtimeStateChanged

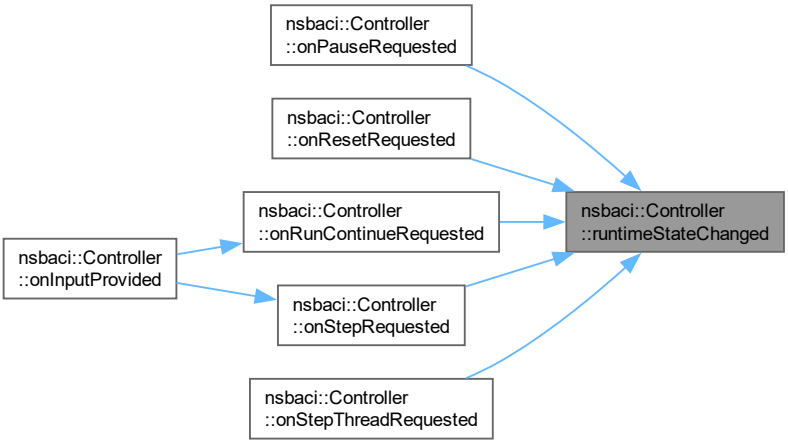
```
void nsbaci::Controller::runtimeStateChanged (
            bool running,
            bool halted)  [signal]
```

Emitted when the runtime execution state changes.

Parameters

running	True if the program is currently executing continuously.
halted	True if the program has terminated (reached Halt instruction).

Here is the caller graph for this function:



6.9.3.20 saveFailed

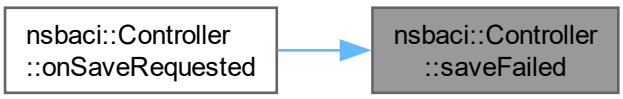
```
void nsbaci::Controller::saveFailed (
            std::vector< UIError > errors)  [signal]
```

Emitted when a file save operation fails.

Parameters

errors	List of errors describing what went wrong.

Here is the caller graph for this function:



6.9.3.21 threadsUpdated

```
void nsbaci::Controller::threadsUpdated (
            const std::vector< nsbaci::ui::ThreadInfo > & threads)  [signal]
```

Emitted when thread information needs to be updated in the UI.

Parameters

threads	Current state of all threads including PC, state, and current instruction.

6.9.3.22 variablesUpdated

```
void nsbaci::Controller::variablesUpdated (
            const std::vector< nsbaci::ui::VariableInfo > & variables)  [signal]
```

Emitted when variable information needs to be updated in the UI.

Parameters

variables	Current values of all program variables.

The documentation for this class was generated from the following files:

- controller.h
- controller.cpp

6.10 nsbaci::factories::DefaultDrawingBackend Struct Reference

The documentation for this struct was generated from the following file:

- drawingServiceFactory.h

6.11 nsbaci::factories::DefaultFileSystem Struct Reference

The documentation for this struct was generated from the following file:

- fileServiceFactory.h

6.12 nsbaci::services::DrawingService Class Reference

Adapter service for graphical output backends.

```
#include <drawingService.h>
```

Public Member Functions

- **DrawingService** (const DrawingService &)=delete
- DrawingService & **operator=** (const DrawingService &)=delete
- **DrawingService** (DrawingService &&)=default
- DrawingService & **operator=** (DrawingService &&)=default

6.12.1 Detailed Description

Adapter service for graphical output backends.

The documentation for this class was generated from the following file:

- drawingService.h

6.13 nsbaci::factories::DrawingServiceFactory Class Reference

Factory for creating DrawingService instances.

`#include <drawingServiceFactory.h>`

Static Public Member Functions

- static nsbaci::services::DrawingService **createService** (DefaultDrawingBackend t)

6.13.1 Detailed Description

Factory for creating DrawingService instances.

The documentation for this class was generated from the following files:

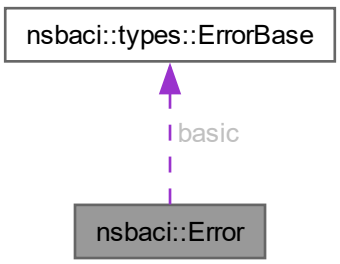
- drawingServiceFactory.h
- drawingServiceFactory.cpp

6.14 nsbaci::Error Class Reference

Represents an error with a message and optional code.

`#include <error.h>`

Collaboration diagram for nsbaci::Error:



Public Attributes

- nsbaci::types::ErrorBase **basic**
- nsbaci::types::ErrorPayload **payload**

6.14.1 Detailed Description

Represents an error with a message and optional code.

The documentation for this class was generated from the following file:

- error.h

6.15 nsbaci::types::ErrorBase Struct Reference

Base structure containing common error properties.

`#include <errorTypes.h>`

Public Attributes

- ErrSeverity **severity**
- ErrMessage **message**
- ErrType **type**

6.15.1 Detailed Description

Base structure containing common error properties.

The documentation for this struct was generated from the following file:

- errorTypes.h

6.16 nsbaci::ui::ErrorDialogFactory Class Reference

Factory for creating error dialogs from UIError objects.

`#include <errorDialogFactory.h>`

Public Types

- using DialogInvoker = std::function<QMessageBox::StandardButton()>

  Callable type that shows a dialog when invoked.

Static Public Member Functions

- static DialogInvoker getDialogFromUIError (const UIError &error, QWidget *parent=nullptr)

  Creates a dialog invoker from a UIError.
- static std::vector< DialogInvoker > getDialogsFromUIErrors (const std::vector< UIError > &errors, QWidget *parent=nullptr)

  Creates dialog invokers for multiple UIErrors.
- static DialogInvoker getSuccessDialog (const QString &title, const QString &message, QWidget *parent=nullptr)

  Creates a success message dialog invoker.
- static void showErrors (const std::vector< UIError > &errors, QWidget *parent=nullptr)

  Shows all error dialogs sequentially.
- static QMessageBox::StandardButton showError (const UIError &error, QWidget *parent=nullptr)

  Shows a single error dialog immediately.
- static void showSuccess (const QString &title, const QString &message, QWidget *parent=nullptr)

  Shows a success message dialog immediately.

6.16.1 Detailed Description

Factory for creating error dialogs from UIError objects.

Provides static methods to convert UIError objects into QMessageBox dialogs. Supports two modes:

- **Deferred**: Returns a DialogInvoker callable for later invocation
- **Immediate**: Shows the dialog right away via convenience methods

See also

DialogInvoker

6.16.2 Member Typedef Documentation

6.16.2.1 DialogInvoker

```
using nsbaci::ui::ErrorDialogFactory::DialogInvoker = std::function<QMessageBox::Standard↩
Button()>
```

Callable type that shows a dialog when invoked.

A DialogInvoker is a packaged dialog that can be invoked at any time. When called, it displays the dialog (blocking) and returns which button the user clicked. Think of it as a "suspended dialog" or "dialog builder" that you can trigger when ready.

Returns

The button that was clicked (QMessageBox::StandardButton).

6.16.3 Member Function Documentation

6.16.3.1 getDialogFromUIError()

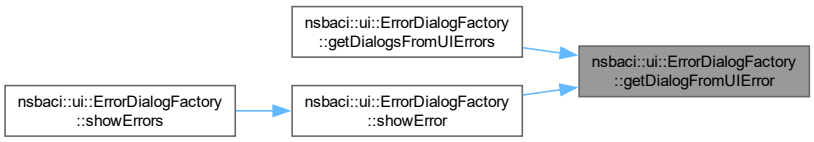
Parameters

error	The UIError to display.
parent	Parent widget for the dialog.

Returns

A callable that shows the dialog when invoked.

Here is the caller graph for this function:



6.16.3.2 getDialogsFromUIErrors()

```
std::vector< ErrorDialogFactory::DialogInvoker > nsbaci::ui::ErrorDialogFactory::getDialogs↩
FromUIErrors (
            const std::vector< UIError > & errors,
            QWidget * parent = nullptr)  [static]
```

Creates dialog invokers for multiple UIErrors.

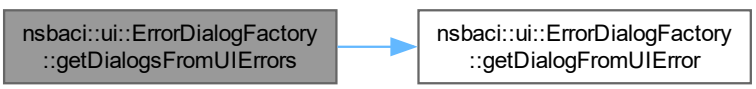
Parameters

errors	Vector of UIErrors to convert.
parent	Parent widget for all dialogs.

Returns

Vector of callables, one per error.

Here is the call graph for this function:



6.16.3.3 getSuccessDialog()

```
ErrorDialogFactory::DialogInvoker nsbaci::ui::ErrorDialogFactory::getSuccessDialog (
            const QString & title,
            const QString & message,
            QWidget * parent = nullptr)  [static]
```

Creates a success message dialog invoker.

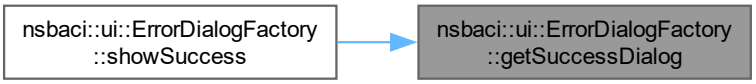
Parameters

title	Dialog title.
message	Success message body.
parent	Parent widget for the dialog.

Returns

A callable that shows the success dialog when invoked.

Here is the caller graph for this function:



6.16.3.4 showError()

```
QMessageBox::StandardButton nsbaci::ui::ErrorDialogFactory::showError (
            const UIError & error,
            QWidget * parent = nullptr)  [static]
```

Shows a single error dialog immediately.

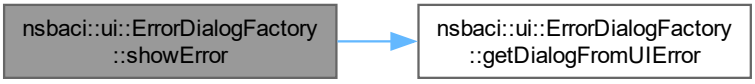
Parameters

error	The UIError to display.
parent	Parent widget for the dialog.

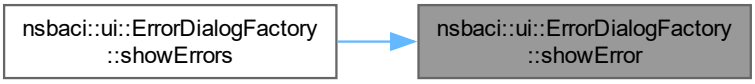
Returns

The button that was clicked.

Here is the call graph for this function:



Here is the caller graph for this function:



6.16.3.5 showErrors()

```
void nsbaci::ui::ErrorDialogFactory::showErrors (
            const std::vector< UIError > & errors,
            QWidget * parent = nullptr)  [static]
```

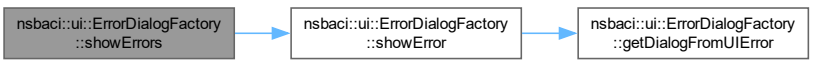
Shows all error dialogs sequentially.

Convenience method that creates and immediately shows dialogs for all provided UIErrors.

Parameters

errors	Vector of UIErrors to display.
parent	Parent widget for all dialogs.

Here is the call graph for this function:



6.16.3.6 showSuccess()

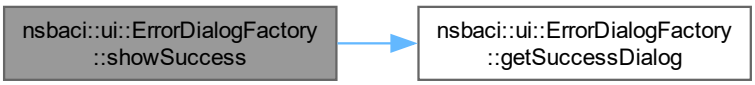
```
void nsbaci::ui::ErrorDialogFactory::showSuccess (
            const QString & title,
            const QString & message,
            QWidget * parent = nullptr)  [static]
```

Shows a success message dialog immediately.

Parameters

title	Dialog title.
message	Success message body.
parent	Parent widget for the dialog.

Here is the call graph for this function:



The documentation for this class was generated from the following files:

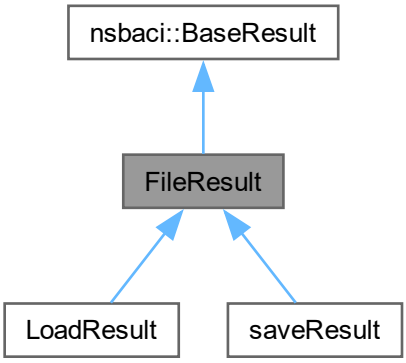
- errorDialogFactory.h
- errorDialogFactory.cpp

6.17 FileResult Struct Reference

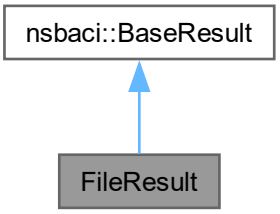
Base result type for file operations.

```
#include <fileService.h>
```

Inheritance diagram for FileResult:



Collaboration diagram for FileResult:



Public Member Functions

- **FileResult** ()

  Default constructor creates a successful result.
- FileResult (std::vector< nsbaci::Error > errs)

  Constructs a result from a vector of errors.
- FileResult (nsbaci::Error error)

  Constructs a failed result from a single error.
- **FileResult** (FileResult &&) noexcept=default
- FileResult & **operator=** (FileResult &&) noexcept=default
- **FileResult** (const FileResult &)=default
- FileResult & **operator=** (const FileResult &)=default

Public Member Functions inherited from nsbaci::BaseResult

- **BaseResult** ()

  Default constructor creates a successful result.
- BaseResult (std::vector< nsbaci::Error > errs)

  Constructs a result from a vector of errors.
- BaseResult (nsbaci::Error error)

  Constructs a failed result from a single error.
- **BaseResult** (BaseResult &&) noexcept=default
- BaseResult & **operator=** (BaseResult &&) noexcept=default
- **BaseResult** (const BaseResult &)=default
- BaseResult & **operator=** (const BaseResult &)=default

Additional Inherited Members

Public Attributes inherited from nsbaci::BaseResult

- bool **ok**

  True if the operation succeeded.
- std::vector< nsbaci::Error > **errors**

  Errors encountered (empty if ok is true).

6.17.1 Detailed Description

Base result type for file operations.

Extends BaseResult with file-specific semantics. All file operation results inherit from this type.

Invariant

If ok is false, errors vector contains at least one error.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 FileResult() [1/2]

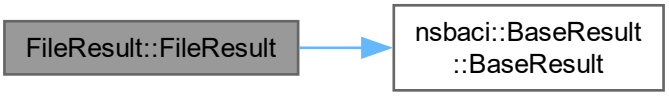
```
FileResult::FileResult (
            std::vector< nsbaci::Error > errs)  [inline], [explicit]
```

Constructs a result from a vector of errors.

Parameters

errs	Vector of errors encountered during the operation.

Here is the call graph for this function:



6.17.2.2 FileResult() [2/2]

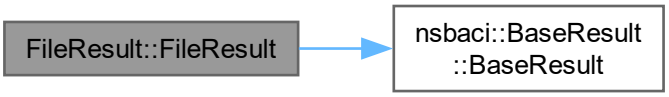
```
FileResult::FileResult (
            nsbaci::Error error)  [inline], [explicit]
```

Constructs a failed result from a single error.

Parameters

error	The error that caused the operation to fail.

Here is the call graph for this function:



The documentation for this struct was generated from the following file:

- fileService.h

6.18 nsbaci::services::FileService Class Reference

Service for handling file system operations on BACI source files.

```
#include <fileService.h>
```

Public Member Functions

- saveResult save (nsbaci::types::Text contents, nsbaci::types::File file)

  Saves source code content to a file.
- LoadResult load (nsbaci::types::File file)

  Loads source code content from a file.
- **FileService** ()=default

  Default constructor.
- **FileService** (const FileService &)=delete
- FileService & **operator=** (const FileService &)=delete
- **FileService** (FileService &&)=default
- FileService & **operator=** (FileService &&)=default

6.18.1 Detailed Description

Service for handling file system operations on BACI source files.

FileService provides methods for saving and loading BACI source code files. It enforces the .nsb file extension and provides detailed error reporting for various failure scenarios including:

- Empty or invalid file paths
- Invalid file extensions (must be .nsb)
- Non-existent directories or files
- Permission and I/O errors

The service is designed to be move-only (non-copyable) to ensure single ownership and prevent accidental resource duplication.

Usage example:

```
FileService fs;

// Save a file
auto saveRes = fs.save(sourceCode, "program.nsb");
if (!saveRes.ok) {
  // Handle save error
}

// Load a file
auto loadRes = fs.load("program.nsb");
if (loadRes.ok) {
  std::string code = loadRes.contents;
}
```

6.18.2 Member Function Documentation

6.18.2.1 load()

```
LoadResult nsbaci::services::FileService::load (
            nsbaci::types::File file)
```

Loads source code content from a file.

Validates the file path, extension, and existence before reading. Returns the complete file contents on success.

Parameters

file	The source file path to load (must have .nsb extension).

Returns

LoadResult containing file contents on success, or error details on failure.

6.18.2.2 save()

```
SaveResult nsbaci::services::FileService::save (
            nsbaci::types::Text contents,
            nsbaci::types::File file)
```

Saves source code content to a file.

Parameters

contents	The source code text to save.
file	The target file path (must have .nsb extension).

Returns

saveResult indicating success or containing error details.

Note

The parent directory must exist; this method does not create directories.

The documentation for this class was generated from the following files:

- fileService.h
- fileService.cpp

6.19 nsbaci::factories::FileServiceFactory Class Reference

Factory for creating FileService instances.

`#include <fileServiceFactory.h>`

Static Public Member Functions

- static nsbaci::services::FileService **createService** (DefaultFileSystem t)

6.19.1 Detailed Description

Factory for creating FileService instances.

The documentation for this class was generated from the following files:

- fileServiceFactory.h
- fileServiceFactory.cpp

6.20 nsbaci::compiler::Instruction Struct Reference

Represents a single instruction in the virtual machine.

`#include <instruction.h>`

Public Member Functions

- **Instruction** (Opcode op)
- **Instruction** (Opcode op, int32_t op1)
- **Instruction** (Opcode op, uint32_t op1)
- **Instruction** (Opcode op, std::string op1)
- **Instruction** (Opcode op, int32_t op1, int32_t op2)

Public Attributes

- Opcode **opcode**
- Operand **operand1**
- Operand **operand2**

6.20.1 Detailed Description

Represents a single instruction in the virtual machine.

The documentation for this struct was generated from the following file:

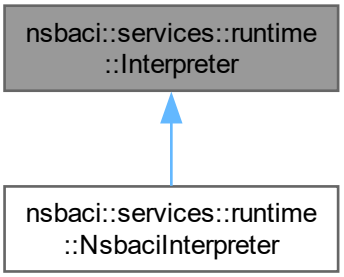
- instruction.h

6.21 nsbaci::services::runtime::Interpreter Class Reference

Executes instructions for threads within a program context.

`#include <interpreter.h>`

Inheritance diagram for nsbaci::services::runtime::Interpreter:



Public Member Functions

- virtual InterpreterResult executeInstruction (Thread &t, Program &program)=0

  Executes the current instruction for the given thread with the program context.
- virtual void provideInput (const std::string &input)=0

  Provide input to a thread waiting for input.
- virtual bool isWaitingForInput () const =0

  Check if interpreter is waiting for input.
- virtual void setOutputCallback (OutputCallback callback)=0

  Set the output callback for print operations.

6.21.1 Detailed Description

Executes instructions for threads within a program context.

The Interpreter is responsible for fetching and executing instructions based on the current thread's program counter.

6.21.2 Member Function Documentation

6.21.2.1 executeInstruction()

```
virtual InterpreterResult nsbaci::services::runtime::Interpreter::executeInstruction (
            Thread & t,
            Program & program)  [pure virtual]
```

Executes the current instruction for the given thread with the program context.

Parameters

t	The thread whose instruction should be executed.
program	The program context in which to execute the instruction

Implemented in nsbaci::services::runtime::NsbaciInterpreter.

6.21.2.2 isWaitingForInput()

```
virtual bool nsbaci::services::runtime::Interpreter::isWaitingForInput () const  [pure virtual]
```

Check if interpreter is waiting for input.

Returns

True if waiting for input.

Implemented in nsbaci::services::runtime::NsbaciInterpreter.

6.21.2.3 provideInput()


```
virtual void nsbaci::services::runtime::Interpreter::provideInput (
            const std::string & input)  [pure virtual]
```

Parameters

input	The input string.

Implemented in nsbaci::services::runtime::NsbaciInterpreter.

6.21.2.4 setOutputCallback()

```
virtual void nsbaci::services::runtime::Interpreter::setOutputCallback (
            OutputCallback callback)  [pure virtual]
```

Set the output callback for print operations.

Parameters

callback	Function to call when output is produced.

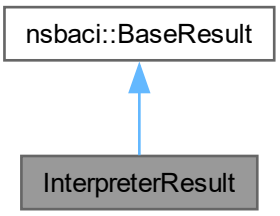
Implemented in nsbaci::services::runtime::NsbaciInterpreter.

The documentation for this class was generated from the following file:

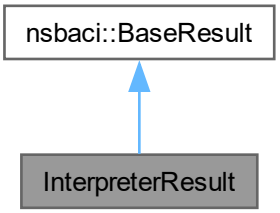
- interpreter.h

6.22 InterpreterResult Struct Reference

Inheritance diagram for InterpreterResult:



Collaboration diagram for InterpreterResult:



Public Member Functions

- **InterpreterResult** (std::vector< nsbaci::Error > errs)
- **InterpreterResult** (nsbaci::Error error)
- **InterpreterResult** (InterpreterResult &&) noexcept=default
- InterpreterResult & **operator=** (InterpreterResult &&) noexcept=default
- **InterpreterResult** (const InterpreterResult &)=default
- InterpreterResult & **operator=** (const InterpreterResult &)=default

Public Member Functions inherited from nsbaci::BaseResult

- **BaseResult** ()

  Default constructor creates a successful result.
- BaseResult (std::vector< nsbaci::Error > errs)

  Constructs a result from a vector of errors.
- BaseResult (nsbaci::Error error)

  Constructs a failed result from a single error.
- **BaseResult** (BaseResult &&) noexcept=default
- BaseResult & **operator=** (BaseResult &&) noexcept=default
- **BaseResult** (const BaseResult &)=default
- BaseResult & **operator=** (const BaseResult &)=default

Public Attributes

- bool **needsInput** = false

  Thread is waiting for input.
- std::string **inputPrompt**

  Prompt to show for input.
- std::string **output**

  Output produced by this instruction.

Public Attributes inherited from nsbaci::BaseResult

- bool **ok**

  True if the operation succeeded.
- std::vector< nsbaci::Error > **errors**

  Errors encountered (empty if ok is true).

The documentation for this struct was generated from the following file:

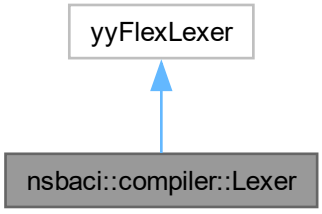
- interpreter.h

6.23 nsbaci::compiler::Lexer Class Reference

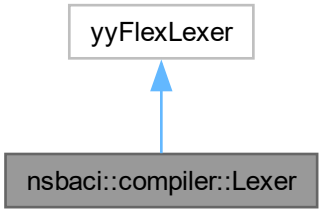
Flex-based lexer for BACI source code.

`#include <lexer.h>`

Inheritance diagram for nsbaci::compiler::Lexer:



Collaboration diagram for nsbaci::compiler::Lexer:



Public Member Functions

- **Lexer** (std::istream *in=nullptr)
- int **yylex** (Parser::semantic_type *yylval, Parser::location_type *yylloc)

6.23.1 Detailed Description

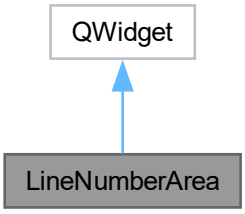
Flex-based lexer for BACI source code.

The documentation for this class was generated from the following file:

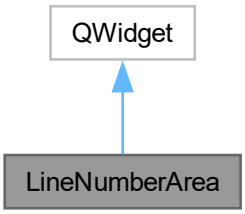
- lexer.h

6.24 LineNumberArea Class Reference

Inheritance diagram for LineNumberArea:



Collaboration diagram for LineNumberArea:



Public Member Functions

- **LineNumberArea** (CodeEditor *editor)
- QSize **sizeHint** () const override

Protected Member Functions

- void **paintEvent** (QPaintEvent ∗event) override

The documentation for this class was generated from the following file:

- codeeditor.h

6.25 nsbaci::types::LoadError Struct Reference

Error payload for file load errors.

`#include <errorTypes.h>`

Public Attributes

- File **associatedFile**

6.25.1 Detailed Description

Error payload for file load errors.

The documentation for this struct was generated from the following file:

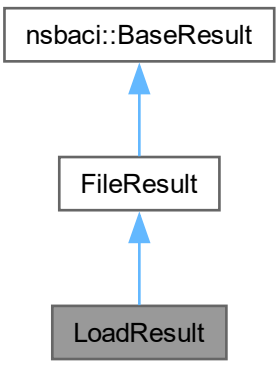
- errorTypes.h

6.26 LoadResult Struct Reference

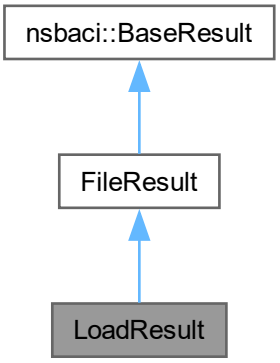
Result type for file load operations.

`#include <fileService.h>`

Inheritance diagram for LoadResult:



Collaboration diagram for LoadResult:



Public Member Functions

- **LoadResult** ()

  Default constructor creates a successful but empty result.
- LoadResult (nsbaci::types::Text conts, nsbaci::types::File name)

  Constructs a successful result with file contents.
- LoadResult (std::vector< nsbaci::Error > errs)

  Constructs a result from a vector of errors.
- LoadResult (nsbaci::Error error)

  Constructs a failed result from a single error.
- **LoadResult** (LoadResult &&) noexcept=default
- LoadResult & **operator=** (LoadResult &&) noexcept=default
- **LoadResult** (const LoadResult &)=default
- LoadResult & **operator=** (const LoadResult &)=default

Public Member Functions inherited from FileResult

- **FileResult** ()

  Default constructor creates a successful result.
- FileResult (std::vector< nsbaci::Error > errs)

  Constructs a result from a vector of errors.
- FileResult (nsbaci::Error error)

  Constructs a failed result from a single error.
- **FileResult** (FileResult &&) noexcept=default
- FileResult & **operator=** (FileResult &&) noexcept=default
- **FileResult** (const FileResult &)=default
- FileResult & **operator=** (const FileResult &)=default

Public Member Functions inherited from nsbaci::BaseResult

- **BaseResult** ()

  Default constructor creates a successful result.
- BaseResult (std::vector< nsbaci::Error > errs)

  Constructs a result from a vector of errors.
- BaseResult (nsbaci::Error error)

  Constructs a failed result from a single error.
- **BaseResult** (BaseResult &&) noexcept=default
- BaseResult & **operator=** (BaseResult &&) noexcept=default
- **BaseResult** (const BaseResult &)=default
- BaseResult & **operator=** (const BaseResult &)=default

Public Attributes

- nsbaci::types::Text **contents**

  The loaded file contents.
- nsbaci::types::File **fileName**

  The filename for display purposes.

Public Attributes inherited from nsbaci::BaseResult

- bool **ok**

  True if the operation succeeded.
- std::vector< nsbaci::Error > **errors**

  Errors encountered (empty if ok is true).

6.26.1 Detailed Description

Result type for file load operations.

Extends FileResult with the loaded file contents and filename. On successful load, contains the file contents as a string and the filename for display purposes.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 LoadResult() [1/3]

```
LoadResult::LoadResult (
            nsbaci::types::Text conts,
            nsbaci::types::File name)  [inline]
```

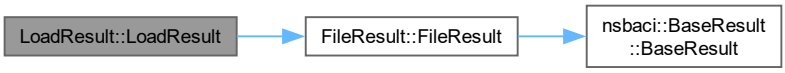
Constructs a successful result with file contents.

Parameters

conts	The loaded file contents.

name	The filename (without path) for display.

Here is the call graph for this function:



6.26.2.2 LoadResult() [2/3]

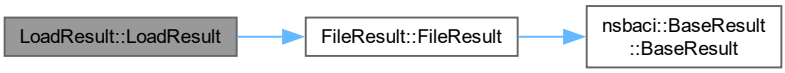
```
LoadResult::LoadResult (
            std::vector< nsbaci::Error > errs)  [inline], [explicit]
```

Constructs a result from a vector of errors.

Parameters

errs	Vector of errors encountered during the load.

Here is the call graph for this function:



6.26.2.3 LoadResult() [3/3]

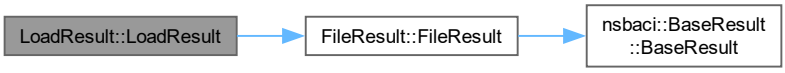
```
LoadResult::LoadResult (
            nsbaci::Error error)  [inline], [explicit]
```

Constructs a failed result from a single error.

Parameters

error	The error that caused the load to fail.

Here is the call graph for this function:

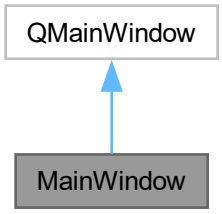


The documentation for this struct was generated from the following file:

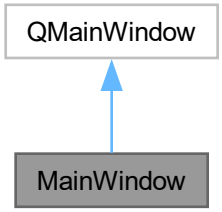
- fileService.h

6.27 MainWindow Class Reference

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Public Slots

- void **setEditorContents** (const QString &contents)
- void **setStatusMessage** (const QString &message)
- void **setCurrentFile** (const QString &fileName, bool modified=false)
- void **onSaveSucceeded** ()
- void **onSaveFailed** (std::vector< nsbaci::UIError > errors)
- void **onLoadSucceeded** (const QString &contents)
- void **onLoadFailed** (std::vector< nsbaci::UIError > errors)
- void **onCompileSucceeded** ()
- void **onCompileFailed** (std::vector< nsbaci::UIError > errors)
- void **onRunStarted** (const QString &programName)
- void **onRuntimeStateChanged** (bool running, bool halted)
- void **onThreadsUpdated** (const std::vector< nsbaci::ui::ThreadInfo > &threads)
- void **onVariablesUpdated** (const std::vector< nsbaci::ui::VariableInfo > &variables)
- void **onOutputReceived** (const QString &output)
- void **onInputRequested** (const QString &prompt)

Signals

- void **saveRequested** (const QString &filePath, const QString &contents)
- void **openRequested** (const QString &filePath)
- void **compileRequested** (const QString &contents)
- void **runRequested** ()
- void **stepRequested** ()
- void **stepThreadRequested** (nsbaci::types::ThreadID threadId)
- void **runContinueRequested** ()
- void **pauseRequested** ()
- void **resetRequested** ()
- void **stopRequested** ()
- void **inputProvided** (const QString &input)

Public Member Functions

- **MainWindow** (QWidget *parent=nullptr)

The documentation for this class was generated from the following files:

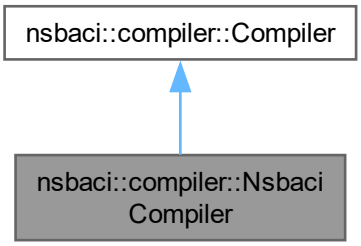
- mainwindow.h
- mainwindow.cpp

6.28 nsbaci::compiler::NsbaciCompiler Class Reference

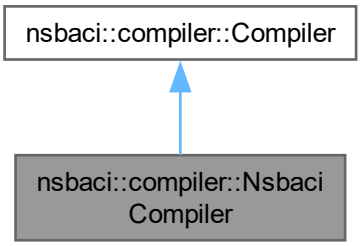
nsbaci compiler implementation using flex and bison.

`#include <nsbaciCompiler.h>`

Inheritance diagram for nsbaci::compiler::NsbaciCompiler:



Collaboration diagram for nsbaci::compiler::NsbaciCompiler:



Public Member Functions

- **NsbaciCompiler** ()=default

  Default constructor.
- ∼**NsbaciCompiler** () override=default

  Destructor.
- CompilerResult compile (const std::string &source) override

  Compiles nsbaci source code from a string.
- CompilerResult compile (std::istream &input) override

  Compiles nsbaci source code from an input stream.

Public Member Functions inherited from nsbaci::compiler::Compiler

- **Compiler** ()=default

  Default constructor.
- virtual ~**Compiler** ()=default

  Virtual destructor.

6.28.1 Detailed Description

nsbaci compiler implementation using flex and bison.

Usage example:

```
NsbaciCompiler compiler;
auto result = compiler.compile("int x = 5; cout « x « endl;");
if (result.ok) {
  // result.instructions contains the p-code
  // result.symbols contains variable information
} else {
  // Handle compilation errors
}
```

6.28.2 Member Function Documentation

6.28.2.1 compile() [1/2]

```
CompilerResult nsbaci::compiler::NsbaciCompiler::compile (
            const std::string & source)  [override], [virtual]
```

Compiles nsbaci source code from a string.

Creates a string stream from the source and delegates to the stream compile method.

Parameters

source	The nsbaci source code to compile.

Returns

CompilerResult with instructions and symbols on success, or detailed error information on failure.

Implements nsbaci::compiler::Compiler.

Here is the call graph for this function:

Here is the caller graph for this function:

6.28.2.2 compile() [2/2]

```
CompilerResult nsbaci::compiler::NsbaciCompiler::compile (
            std::istream & input)  [override], [virtual]
```

Compiles nsbaci source code from an input stream.

Performs the actual compilation by:

1. Creating a Lexer to tokenize the input
2. Creating a Parser with the lexer
3. Running the parser to generate instructions
4. Collecting any errors that occurred

Parameters

input	The input stream containing nsbaci source code.

Returns

CompilerResult with instructions and symbols on success, or detailed error information on failure.

Implements nsbaci::compiler::Compiler.

The documentation for this class was generated from the following files:

- nsbaciCompiler.h
- nsbaciCompiler.cpp

6.29 nsbaci::factories::NsbaciCompiler Struct Reference

The documentation for this struct was generated from the following file:

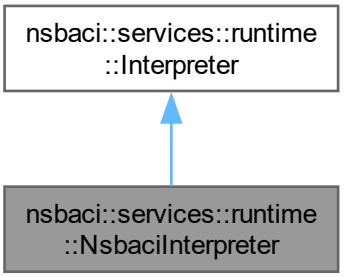
- compilerServiceFactory.h

6.30 nsbaci::services::runtime::NsbaciInterpreter Class Reference

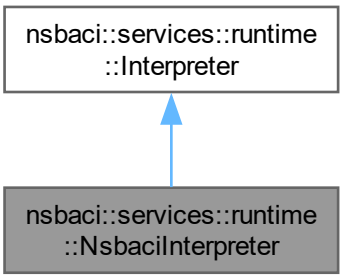
BACI-specific implementation of the Interpreter.

`#include <nsbaciInterpreter.h>`

Inheritance diagram for nsbaci::services::runtime::NsbaciInterpreter:



Collaboration diagram for nsbaci::services::runtime::NsbaciInterpreter:



Public Member Functions

- InterpreterResult executeInstruction (Thread &t, Program &program) override

  Executes the current instruction for the given thread with the program context.
- void provideInput (const std::string &input) override

  Provide input to a thread waiting for input.
- bool isWaitingForInput () const override

  Check if interpreter is waiting for input.
- void setOutputCallback (OutputCallback callback) override

  Set the output callback for print operations.

6.30.1 Detailed Description

BACI-specific implementation of the Interpreter.

NsbaciInterpreter executes BACI p-code instructions, managing the execution state and interaction with the program context.

6.30.2 Member Function Documentation

6.30.2.1 executeInstruction()

```
InterpreterResult nsbaci::services::runtime::NsbaciInterpreter::executeInstruction (
            Thread & t,
            Program & program)  [override], [virtual]
```

Executes the current instruction for the given thread with the program context.

Parameters

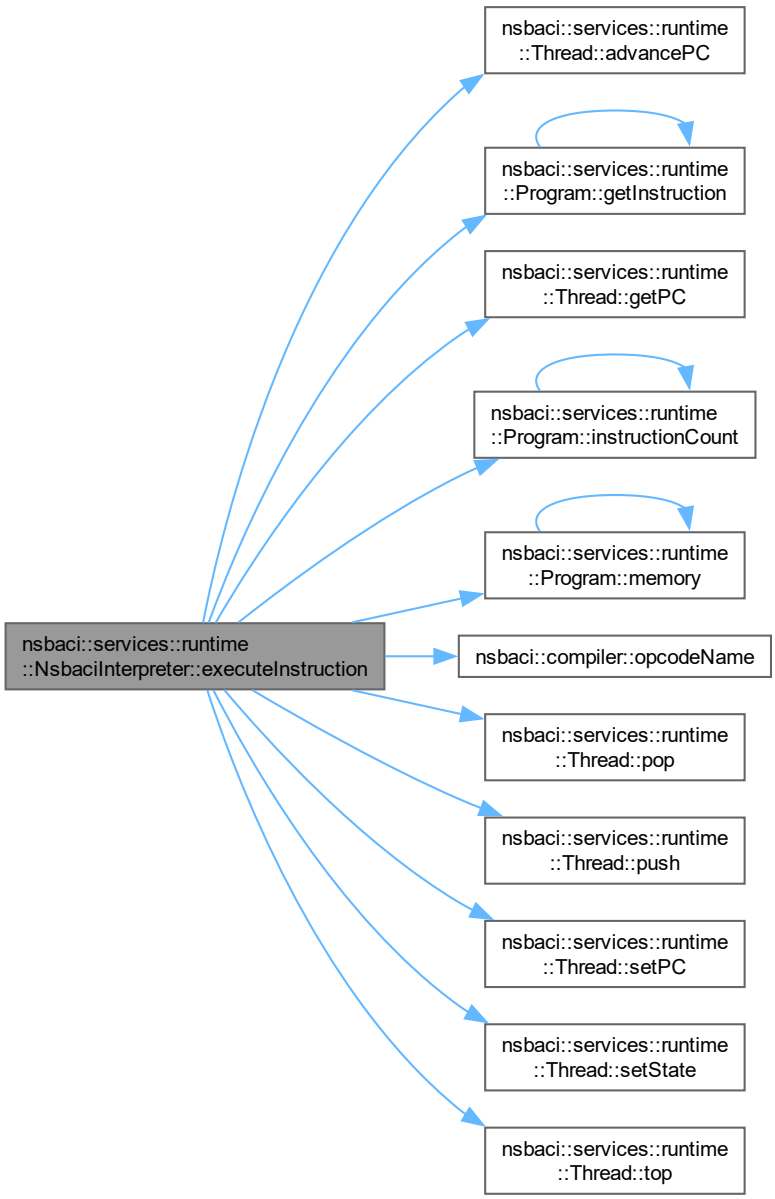
t	The thread whose instruction should be executed.
program	The program context in which to execute the instruction.

Returns

InterpreterResult indicating success or any errors encountered.

Implements nsbaci::services::runtime::Interpreter.

Here is the call graph for this function:



6.30.2.2 isWaitingForInput()

```
bool nsbaci::services::runtime::NsbaciInterpreter::isWaitingForInput () const  [override],
[virtual]
```

Check if interpreter is waiting for input.

Returns

True if waiting for input.

Implements nsbaci::services::runtime::Interpreter.

6.30.2.3 provideInput()

```
void nsbaci::services::runtime::NsbaciInterpreter::provideInput (
            const std::string & input)  [override], [virtual]
```

Provide input to a thread waiting for input.

Parameters

input	The input string.

Implements nsbaci::services::runtime::Interpreter.

6.30.2.4 setOutputCallback()

```
void nsbaci::services::runtime::NsbaciInterpreter::setOutputCallback (
            OutputCallback callback)  [override], [virtual]
```

Set the output callback for print operations.

Parameters

callback	Function to call when output is produced.

Implements nsbaci::services::runtime::Interpreter.

The documentation for this class was generated from the following files:

- nsbaciInterpreter.h
- nsbaciInterpreter.cpp

6.31 nsbaci::factories::NsbaciRuntime Struct Reference

The documentation for this struct was generated from the following file:

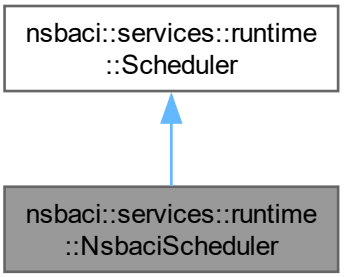
- runtimeServiceFactory.h

6.32 nsbaci::services::runtime::NsbaciScheduler Class Reference

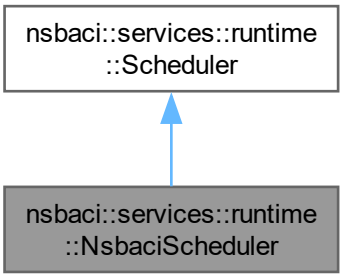
BACI-specific implementation of the Scheduler.

`#include <nsbaciScheduler.h>`

Inheritance diagram for nsbaci::services::runtime::NsbaciScheduler:



Collaboration diagram for nsbaci::services::runtime::NsbaciScheduler:



Public Member Functions

- Thread * pickNext () override
  Pick the next thread to run.
- void addThread (Thread thread) override
  Add a new thread to the scheduler.
- void blockCurrent () override
  Block the currently running thread.
- void unblock (nsbaci::types::ThreadID threadId) override
  Move a thread from blocked to ready state.

- void yield () override

  Yield the current thread (move to back of ready queue).
- void terminateCurrent () override

  Terminate the currently running thread.
- bool hasThreads () const override

  Check if there are any threads left to run.
- Thread * current () override

  Get the currently running thread.
- void clear () override

  Clear all threads and reset scheduler state.
- void unblockIO () override

  Move all I/O waiting threads back to ready queue.
- const std::vector< Thread > & getThreads () const override

  Get all threads managed by the scheduler.

Additional Inherited Members

Protected Attributes inherited from nsbaci::services::runtime::Scheduler

- std::vector< Thread > **threads**

  All threads owned by scheduler.
- std::vector< size_t > **readyQueue**

  Indices of ready threads.
- std::vector< size_t > **blockedQueue**

  Indices of blocked threads.
- std::vector< size_t > **ioQueue**

  Indices of I/O waiting threads.
- std::optional< size_t > **runningIndex**

  Index of currently running thread.

6.32.1 Detailed Description

BACI-specific implementation of the Scheduler.

NsbaciScheduler implements a round-robin scheduling algorithm with support for blocked, ready, running, and I/↩ O waiting states. Threads are selected randomly from the ready queue to simulate non-deterministic concurrent execution.

6.32.2 Member Function Documentation

6.32.2.1 addThread()

```
void nsbaci::services::runtime::NsbaciScheduler::addThread (
            Thread thread)  [override], [virtual]
```

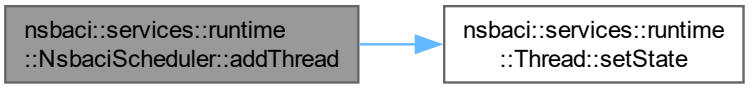
Add a new thread to the scheduler.

Parameters

thread	The thread to add.

Implements nsbaci::services::runtime::Scheduler.

Here is the call graph for this function:



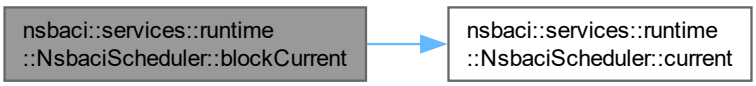
6.32.2.2 blockCurrent()

```
void nsbaci::services::runtime::NsbaciScheduler::blockCurrent ()  [override], [virtual]
```

Block the currently running thread.

Implements nsbaci::services::runtime::Scheduler.

Here is the call graph for this function:



6.32.2.3 clear()

```
void nsbaci::services::runtime::NsbaciScheduler::clear ()  [override], [virtual]
```

Clear all threads and reset scheduler state.

Implements nsbaci::services::runtime::Scheduler.

6.32.2.4 current()

```
Thread * nsbaci::services::runtime::NsbaciScheduler::current ()  [override], [virtual]
```

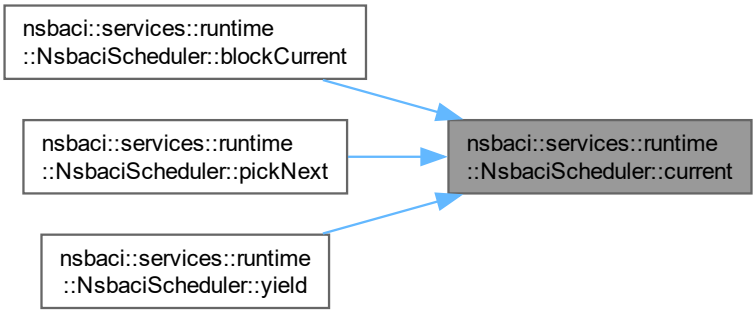
Get the currently running thread.

Returns

Pointer to current thread, or nullptr if none.

Implements nsbaci::services::runtime::Scheduler.

Here is the caller graph for this function:



6.32.2.5 getThreads()

```
const std::vector< Thread > & nsbaci::services::runtime::NsbaciScheduler::getThreads () const
[override], [virtual]
```

Get all threads managed by the scheduler.

Returns

Const reference to the threads vector.

Implements nsbaci::services::runtime::Scheduler.

6.32.2.6 hasThreads()

```
bool nsbaci::services::runtime::NsbaciScheduler::hasThreads () const  [override], [virtual]
```

Check if there are any threads left to run.

Returns

True if there are threads in any queue.

Implements nsbaci::services::runtime::Scheduler.

6.32.2.7 pickNext()

```
Thread * nsbaci::services::runtime::NsbaciScheduler::pickNext ()  [override], [virtual]
```

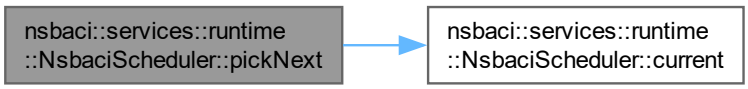
Pick the next thread to run.

Returns

Pointer to the next thread, or nullptr if no threads are ready.

Implements nsbaci::services::runtime::Scheduler.

Here is the call graph for this function:



6.32.2.8 terminateCurrent()

```
void nsbaci::services::runtime::NsbaciScheduler::terminateCurrent ()  [override], [virtual]
```

Terminate the currently running thread.

Implements nsbaci::services::runtime::Scheduler.

6.32.2.9 unblock()

```
void nsbaci::services::runtime::NsbaciScheduler::unblock (
            nsbaci::types::ThreadID threadId)  [override], [virtual]
```

Move a thread from blocked to ready state.

Parameters

threadId	The ID of the thread to unblock.

Implements nsbaci::services::runtime::Scheduler.

6.32.2.10 unblockIO()

```
void nsbaci::services::runtime::NsbaciScheduler::unblockIO ()  [override], [virtual]
```

Move all I/O waiting threads back to ready queue.

Called when input becomes available.

Implements nsbaci::services::runtime::Scheduler.

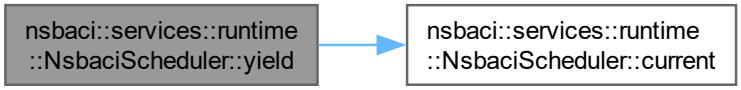
6.32.2.11 yield()

```
void nsbaci::services::runtime::NsbaciScheduler::yield ()  [override], [virtual]
```

Yield the current thread (move to back of ready queue).

Implements nsbaci::services::runtime::Scheduler.

Here is the call graph for this function:



The documentation for this class was generated from the following files:

- nsbaciScheduler.h
- nsbaciScheduler.cpp

6.33 nsbaci::services::runtime::Program Class Reference

Represents a compiled program ready for execution.

```
#include <program.h>
```

Public Member Functions

- **Program** (nsbaci::compiler::InstructionStream i)
- **Program** (nsbaci::compiler::InstructionStream i, nsbaci::types::SymbolTable s)
- **Program** (const Program &)=delete
- Program & **operator=** (const Program &)=delete
- **Program** (Program &&)=default
- Program & **operator=** (Program &&)=default
- const nsbaci::compiler::Instruction & getInstruction (uint32_t addr) const

  Gets instruction at the given address.
- size_t instructionCount () const

  Gets the total number of instructions.
- nsbaci::types::Memory & memory ()

  Access to global memory.
- const nsbaci::types::Memory & **memory** () const
- const nsbaci::types::SymbolTable & symbols () const

  Access to symbol table.
- void addSymbol (nsbaci::types::SymbolInfo info)

  Add a symbol to the symbol table.
- int32_t readMemory (nsbaci::types::MemoryAddr addr) const

  Read a value from memory.
- void writeMemory (nsbaci::types::MemoryAddr addr, int32_t value)

  Write a value to memory.

6.33.1 Detailed Description

Represents a compiled program ready for execution.

The Program class contains the instruction vector, memory tables, and other data structures needed for program execution.

6.33.2 Member Function Documentation

6.33.2.1 addSymbol()

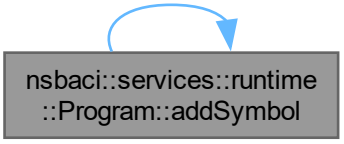
```
void nsbaci::services::runtime::Program::addSymbol (
            nsbaci::types::SymbolInfo info)
```

Add a symbol to the symbol table.

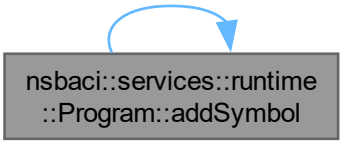
Parameters

info	The symbol information to add.

Here is the call graph for this function:



Here is the caller graph for this function:



6.33.2.2 getInstruction()

```
const nsbaci::compiler::Instruction & nsbaci::services::runtime::Program::getInstruction (
            uint32_t addr) const
```

Gets instruction at the given address.

Parameters

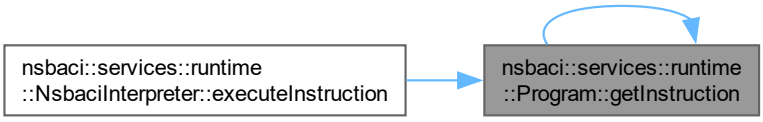
addr	The instruction address.

Returns

Reference to the instruction.

Here is the call graph for this function:

Here is the caller graph for this function:



6.33.2.3 instructionCount()

```
size_t nsbaci::services::runtime::Program::instructionCount () const
```

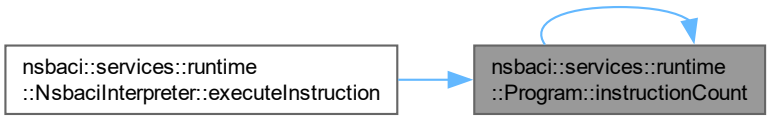
Gets the total number of instructions.

Returns

Number of instructions in the program.

Here is the call graph for this function:

Here is the caller graph for this function:



6.33.2.4 memory()

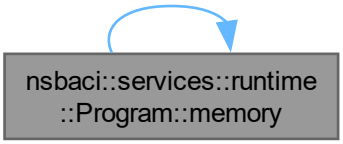
```
nsbaci::types::Memory & nsbaci::services::runtime::Program::memory ()
```

Access to global memory.

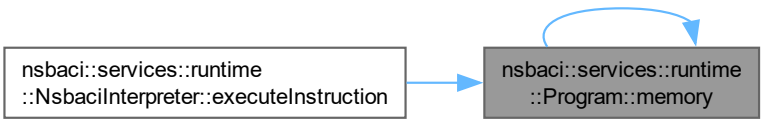
Returns

Reference to memory.

Here is the call graph for this function:



Here is the caller graph for this function:



6.33.2.5 readMemory()


```
int32_t nsbaci::services::runtime::Program::readMemory (
            nsbaci::types::MemoryAddr addr) const
```

Parameters

addr	Memory address to read from.

Returns

Value at the address.

Here is the call graph for this function:

Here is the caller graph for this function:

6.33.2.6 symbols()

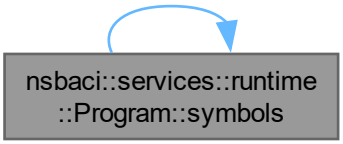
```
const nsbaci::types::SymbolTable & nsbaci::services::runtime::Program::symbols () const
```

Access to symbol table.

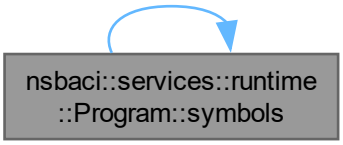
Returns

Const reference to symbol table.

Here is the call graph for this function:



Here is the caller graph for this function:



6.33.2.7 writeMemory()

```
void nsbaci::services::runtime::Program::writeMemory (
            nsbaci::types::MemoryAddr addr,
            int32_t value)
```

Write a value to memory.

Parameters

addr	Memory address to write to.
value	Value to write.

Here is the call graph for this function:

Here is the caller graph for this function:

The documentation for this class was generated from the following files:

- program.h
- program.cpp

6.34 nsbaci::types::RuntimeError Struct Reference

Error payload for runtime errors.

`#include <errorTypes.h>`

6.34.1 Detailed Description

Error payload for runtime errors.

The documentation for this struct was generated from the following file:

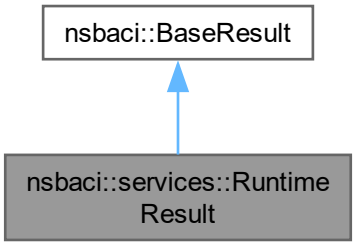
- errorTypes.h

6.35 nsbaci::services::RuntimeResult Struct Reference

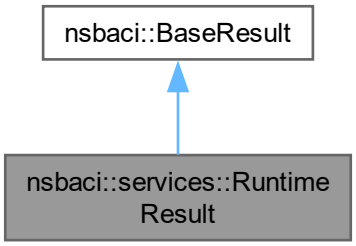
Result of a runtime operation (step, run, etc.).

`#include <runtimeService.h>`

Inheritance diagram for nsbaci::services::RuntimeResult:



Collaboration diagram for nsbaci::services::RuntimeResult:



Public Member Functions

- **RuntimeResult** ()
  Default constructor creates a successful result.
- RuntimeResult (std::vector< nsbaci::Error > errs)
  Constructs a result from a vector of errors.
- RuntimeResult (nsbaci::Error error)
  Constructs a failed result from a single error.
- **RuntimeResult** (RuntimeResult &&) noexcept=default
- RuntimeResult & **operator=** (RuntimeResult &&) noexcept=default
- **RuntimeResult** (const RuntimeResult &)=default
- RuntimeResult & **operator=** (const RuntimeResult &)=default

Public Member Functions inherited from nsbaci::BaseResult

- **BaseResult** ()

  Default constructor creates a successful result.
- BaseResult (std::vector< nsbaci::Error > errs)

  Constructs a result from a vector of errors.
- BaseResult (nsbaci::Error error)

  Constructs a failed result from a single error.
- **BaseResult** (BaseResult &&) noexcept=default
- BaseResult & **operator=** (BaseResult &&) noexcept=default
- **BaseResult** (const BaseResult &)=default
- BaseResult & **operator=** (const BaseResult &)=default

Public Attributes

- bool **halted** = false

  True if program has terminated.
- bool **needsInput** = false

  True if waiting for user input.
- std::string **inputPrompt**

  Prompt to show for input.
- std::string **output**

  Output produced by this step.

Public Attributes inherited from nsbaci::BaseResult

- bool **ok**

  True if the operation succeeded.
- std::vector< nsbaci::Error > **errors**

  Errors encountered (empty if ok is true).

6.35.1 Detailed Description

Result of a runtime operation (step, run, etc.).

6.35.2 Constructor & Destructor Documentation

6.35.2.1 RuntimeResult() [1/2]

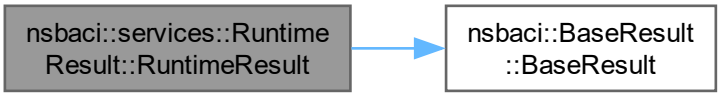
```
nsbaci::services::RuntimeResult::RuntimeResult (
            std::vector< nsbaci::Error > errs)  [inline], [explicit]
```

Constructs a result from a vector of errors.

Parameters

errs	Vector of runtime errors.

Here is the call graph for this function:



6.35.2.2 RuntimeResult() [2/2]

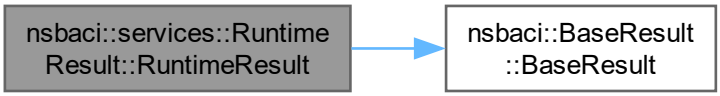
```
nsbaci::services::RuntimeResult::RuntimeResult (
            nsbaci::Error error)  [inline], [explicit]
```

Constructs a failed result from a single error.

Parameters

error	The runtime error that occurred.

Here is the call graph for this function:



The documentation for this struct was generated from the following file:

- runtimeService.h

6.36 nsbaci::services::RuntimeService Class Reference

Service that manages program execution.

```
#include <runtimeService.h>
```

Public Member Functions

- RuntimeService ()=default

  Default constructor creates an uninitialized service.
- RuntimeService (std::unique_ptr< runtime::Interpreter > i, std::unique_ptr< runtime::Scheduler > s)

  Constructs a RuntimeService with interpreter and scheduler.
- ~**RuntimeService** ()=default

  Default destructor.
- **RuntimeService** (const RuntimeService &)=delete
- RuntimeService & **operator=** (const RuntimeService &)=delete
- **RuntimeService** (RuntimeService &&)=default
- RuntimeService & **operator=** (RuntimeService &&)=default
- void loadProgram (runtime::Program &&p)

  Loads a compiled program for execution.
- void reset ()

  Resets the runtime to initial state.
- RuntimeResult step ()

  Executes a single instruction for any ready thread.
- RuntimeResult stepThread (nsbaci::types::ThreadID threadId)

  Executes a single instruction for a specific thread.
- RuntimeResult run (size_t maxSteps=0)

  Runs the program until halted, error, or step limit.
- void pause ()

  Pauses continuous execution.
- RuntimeState getState () const

  Gets the current runtime state.
- bool isHalted () const

  Checks if the program has finished execution.
- size_t threadCount () const

  Gets the number of active threads.
- const std::vector< runtime::Thread > & getThreads () const

  Gets all threads from the scheduler.
- const runtime::Program & getProgram () const

  Gets the loaded program.
- void provideInput (const std::string &input)

  Provides input to the runtime.
- bool isWaitingForInput () const

  Checks if the runtime is waiting for user input.
- void setOutputCallback (runtime::OutputCallback callback)

  Sets the callback for output operations.

6.36.1 Detailed Description

Service that manages program execution.

The service is move-only to ensure single ownership of the interpreter and scheduler components.

Usage example:

```
RuntimeService rs(std::make_unique<NsbaciInterpreter>(),
                  std::make_unique<NsbaciScheduler>());
rs.loadProgram(std::move(compiledProgram));
rs.setOutputCallback([](const std::string& out) { std::cout << out; });

while (!rs.isHalted()) {
  auto result = rs.step();
  if (result.needsInput) {
    rs.provideInput(getUserInput());
  }
}
```

6.36.2 Constructor & Destructor Documentation

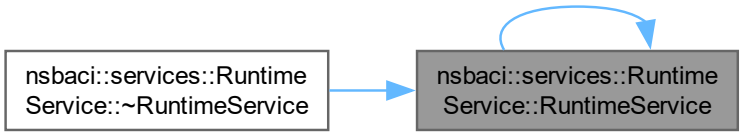
6.36.2.1 RuntimeService() [1/2]

```
nsbaci::services::RuntimeService::RuntimeService ()  [default]
```

Default constructor creates an uninitialized service.

A service created this way must have its interpreter and scheduler set before use, typically via move assignment from a factory-created instance. Here is the call graph for this function:

Here is the caller graph for this function:



6.36.2.2 RuntimeService() [2/2]

```
nsbaci::services::RuntimeService::RuntimeService (
            std::unique_ptr< runtime::Interpreter > i,
            std::unique_ptr< runtime::Scheduler > s)
```

Constructs a RuntimeService with interpreter and scheduler.

Parameters

i	Unique pointer to the interpreter implementation.
s	Unique pointer to the scheduler implementation.

6.36.3 Member Function Documentation

6.36.3.1 getProgram()

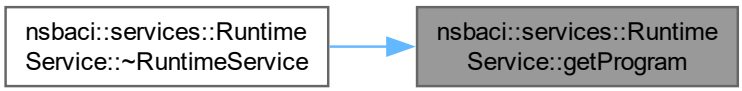
```
const runtime::Program & nsbaci::services::RuntimeService::getProgram () const
```

Gets the loaded program.

Returns

Const reference to the program for accessing instructions and memory.

Here is the caller graph for this function:



6.36.3.2 getState()

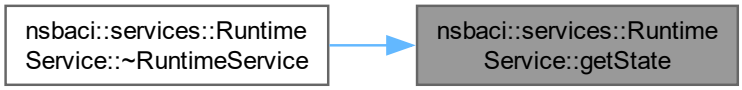
```
RuntimeState nsbaci::services::RuntimeService::getState () const
```

Gets the current runtime state.

Returns

The current RuntimeState value.

Here is the caller graph for this function:



6.36.3.3 getThreads()

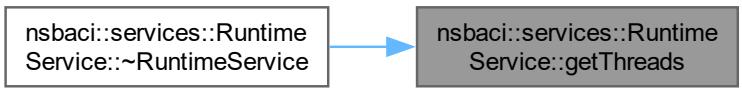
```
const std::vector< runtime::Thread > & nsbaci::services::RuntimeService::getThreads () const
```

Gets all threads from the scheduler.

Returns

Const reference to the threads vector for UI display.

Here is the caller graph for this function:



6.36.3.4 isHalted()

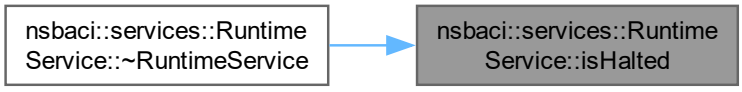
```
bool nsbaci::services::RuntimeService::isHalted () const
```

Checks if the program has finished execution.

Returns

True if state is Halted.

Here is the caller graph for this function:



6.36.3.5 isWaitingForInput()

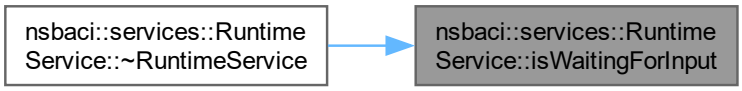
```
bool nsbaci::services::RuntimeService::isWaitingForInput () const
```

Checks if the runtime is waiting for user input.

Returns

True if a Read instruction is blocking execution.

Here is the caller graph for this function:



6.36.3.6 loadProgram()

```
void nsbaci::services::RuntimeService::loadProgram (
            runtime::Program && p)
```

Loads a compiled program for execution.

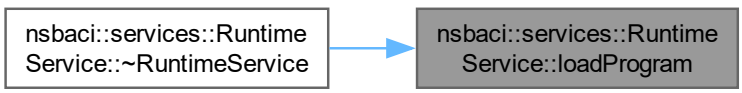
Initializes the runtime with the program's instructions, symbol table, and memory. Creates the initial main thread and sets state to Paused ready for execution.

Parameters

p	The compiled program to load (which must be moved into the service).

Here is the call graph for this function:

Here is the caller graph for this function:

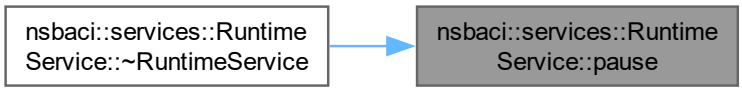


6.36.3.7 pause()

```
void nsbaci::services::RuntimeService::pause ()
```

Pauses continuous execution.

Only affects state if currently Running; changes state to Paused. Here is the caller graph for this function:



6.36.3.8 provideInput()

```
void nsbaci::services::RuntimeService::provideInput (
            const std::string & input)
```

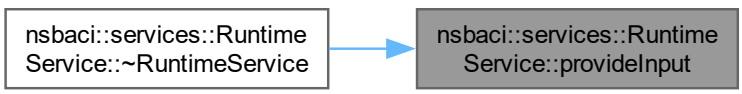
Provides input to the runtime.

Called when the user provides input in response to a Read instruction. The input is stored and will be consumed on the next execution step.

Parameters

input	The user-provided input string.

Here is the caller graph for this function:

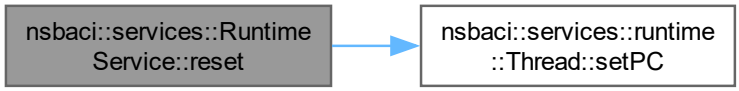


6.36.3.9 reset()

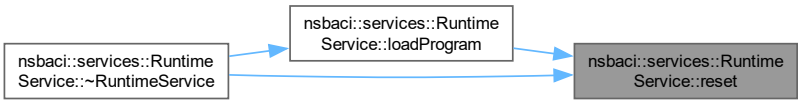
```
void nsbaci::services::RuntimeService::reset ()
```

Resets the runtime to initial state.

Clears all threads, resets memory, and sets state to Idle. The program must be reloaded before execution can continue. Here is the call graph for this function:



Here is the caller graph for this function:



6.36.3.10 run()

```
RuntimeResult nsbaci::services::RuntimeService::run (
            size_t maxSteps = 0)
```

Runs the program until halted, error, or step limit.

Executes instructions continuously until:

- The program halts (reaches Halt instruction)
- An error occurs
- The maximum step count is reached
- Input is required

Parameters

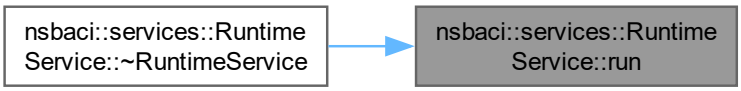
maxSteps	Maximum instructions to execute (0 = unlimited).

Returns

RuntimeResult with final execution state.

Here is the call graph for this function:

Here is the caller graph for this function:



6.36.3.11 setOutputCallback()

```
void nsbaci::services::RuntimeService::setOutputCallback (
            runtime::OutputCallback callback)
```

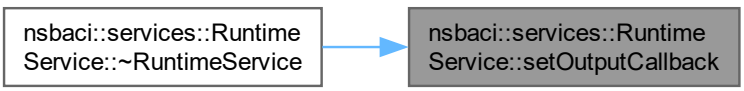
Sets the callback for output operations.

The callback is invoked whenever the program produces output (Write, Writeln, WriteRawString instructions).

Parameters

callback	Function to call with output strings.

Here is the caller graph for this function:



6.36.3.12 step()

```
RuntimeResult nsbaci::services::RuntimeService::step ()
```

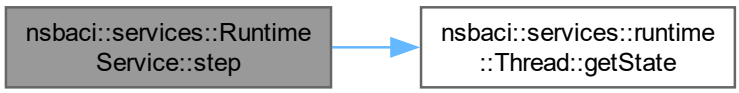
Executes a single instruction for any ready thread.

The scheduler picks the next thread to run and the interpreter executes one instruction from that thread's current position.

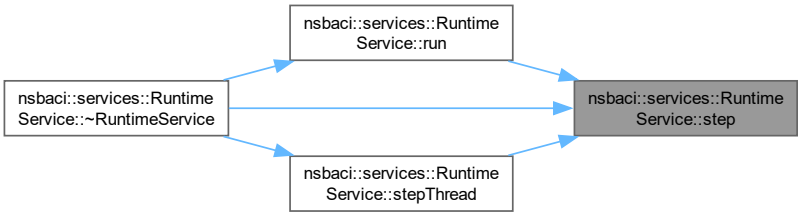
Returns

RuntimeResult with execution outcome, output, and I/O state.

Here is the call graph for this function:



Here is the caller graph for this function:



6.36.3.13 stepThread()

```
RuntimeResult nsbaci::services::RuntimeService::stepThread (
            nsbaci::types::ThreadID threadId)
```

Executes a single instruction for a specific thread.

Allows targeted debugging by stepping only the specified thread.

Parameters

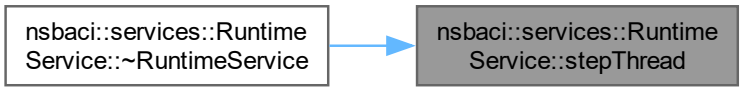
threadId	The ID of the thread to step.

Returns

RuntimeResult with execution outcome.

Here is the call graph for this function:

Here is the caller graph for this function:



6.36.3.14 threadCount()

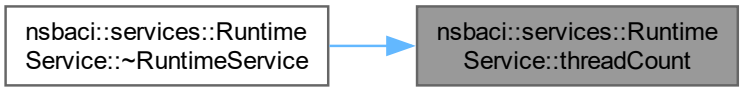
```
size_t nsbaci::services::RuntimeService::threadCount () const
```

Gets the number of active threads.

Returns

Count of threads in the scheduler.

Here is the caller graph for this function:



The documentation for this class was generated from the following files:

- runtimeService.h
- runtimeService.cpp

6.37 nsbaci::factories::RuntimeServiceFactory Class Reference

Factory for creating RuntimeService instances.

`#include <runtimeServiceFactory.h>`

Static Public Member Functions

- static nsbaci::services::RuntimeService **createService** (NsbaciRuntime t)

6.37.1 Detailed Description

Factory for creating RuntimeService instances.

The documentation for this class was generated from the following files:

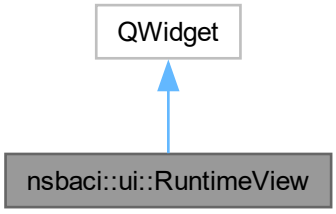
- runtimeServiceFactory.h
- runtimeServiceFactory.cpp

6.38 nsbaci::ui::RuntimeView Class Reference

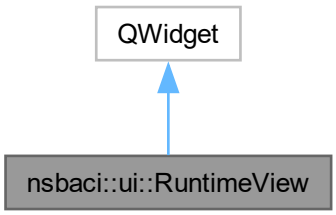
Widget displaying runtime execution state.

`#include <runtimeView.h>`

Inheritance diagram for nsbaci::ui::RuntimeView:



Collaboration diagram for nsbaci::ui::RuntimeView:



Public Slots

- void **updateThreads** (const std::vector< ThreadInfo > &threads)
- void **updateVariables** (const std::vector< VariableInfo > &variables)
- void **updateCurrentInstruction** (const QString &instruction)
- void **updateExecutionState** (bool running, bool halted)
- void **appendOutput** (const QString &text)
- void **requestInput** (const QString &prompt)
- void **clearConsole** ()
- void **onProgramLoaded** (const QString &programName)
- void **onProgramHalted** ()

Signals

- void **stepRequested** ()
- void **stepThreadRequested** (nsbaci::types::ThreadID threadId)
- void **runRequested** ()
- void **pauseRequested** ()
- void **resetRequested** ()
- void **stopRequested** ()
- void **inputProvided** (const QString &input)

Public Member Functions

- **RuntimeView** (QWidget *parent=nullptr)

6.38.1 Detailed Description

Widget displaying runtime execution state.

Shows:

- Thread list with states and current instruction
- Variables/memory watch panel
- I/O console for program input/output
- Execution controls (step, run, pause, reset)

The documentation for this class was generated from the following files:

- runtimeView.h
- runtimeView.cpp

6.39 nsbaci::types::SaveError Struct Reference

Error payload for file save errors.

`#include <errorTypes.h>`

Public Attributes

- File **associatedFile**

6.39.1 Detailed Description

Error payload for file save errors.

The documentation for this struct was generated from the following file:

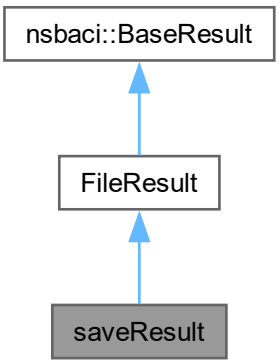
- errorTypes.h

6.40 saveResult Struct Reference

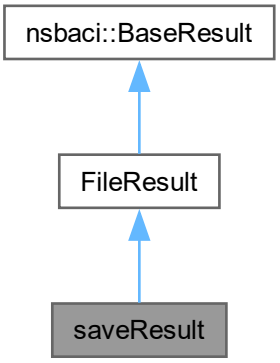
Result type for file save operations.

`#include <fileService.h>`

Inheritance diagram for saveResult:



Collaboration diagram for saveResult:



Public Member Functions

- **saveResult** ()

  Default constructor creates a successful result.
- saveResult (std::vector< nsbaci::Error > errs)

  Constructs a result from a vector of errors.
- saveResult (nsbaci::Error error)

  Constructs a failed result from a single error.
- **saveResult** (saveResult &&) noexcept=default
- saveResult & **operator=** (saveResult &&) noexcept=default
- **saveResult** (const saveResult &)=default
- saveResult & **operator=** (const saveResult &)=default

Public Member Functions inherited from FileResult

- **FileResult** ()

  Default constructor creates a successful result.
- FileResult (std::vector< nsbaci::Error > errs)

  Constructs a result from a vector of errors.
- FileResult (nsbaci::Error error)

  Constructs a failed result from a single error.
- **FileResult** (FileResult &&) noexcept=default
- FileResult & **operator=** (FileResult &&) noexcept=default
- **FileResult** (const FileResult &)=default
- FileResult & **operator=** (const FileResult &)=default

Public Member Functions inherited from nsbaci::BaseResult

- **BaseResult** ()

  Default constructor creates a successful result.
- BaseResult (std::vector< nsbaci::Error > errs)

  Constructs a result from a vector of errors.
- BaseResult (nsbaci::Error error)

  Constructs a failed result from a single error.
- **BaseResult** (BaseResult &&) noexcept=default
- BaseResult & **operator=** (BaseResult &&) noexcept=default
- **BaseResult** (const BaseResult &)=default
- BaseResult & **operator=** (const BaseResult &)=default

Additional Inherited Members

Public Attributes inherited from nsbaci::BaseResult

- bool **ok**

  True if the operation succeeded.
- std::vector< nsbaci::Error > **errors**

  Errors encountered (empty if ok is true).

6.40.1 Detailed Description

Result type for file save operations.

Contains only success/failure status and error information since save operations do not return additional data on success.

6.40.2 Constructor & Destructor Documentation

6.40.2.1 saveResult() [1/2]

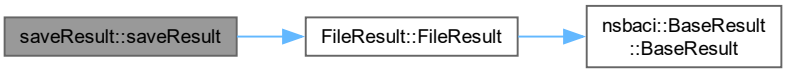
```
saveResult::saveResult (
            std::vector< nsbaci::Error > errs)  [inline], [explicit]
```

Constructs a result from a vector of errors.

Parameters

errs	Vector of errors encountered during the save.

Here is the call graph for this function:



6.40.2.2 saveResult() [2/2]

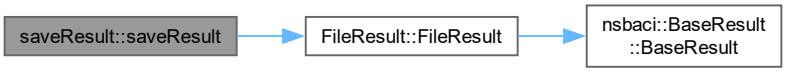
```
saveResult::saveResult (
            nsbaci::Error error)  [inline], [explicit]
```

Constructs a failed result from a single error.

Parameters

error	The error that caused the save to fail.

Here is the call graph for this function:



The documentation for this struct was generated from the following file:

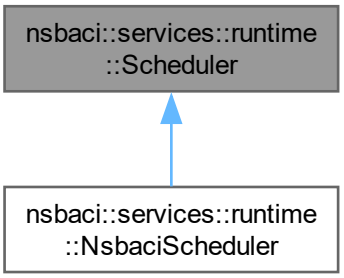
- fileService.h

6.41 nsbaci::services::runtime::Scheduler Class Reference

Manages thread scheduling and state transitions.

```
#include <scheduler.h>
```

Inheritance diagram for nsbaci::services::runtime::Scheduler:



Public Member Functions

- virtual Thread * pickNext ()=0

  Pick the next thread to run.
- virtual void addThread (Thread thread)=0

  Add a new thread to the scheduler.
- virtual void blockCurrent ()=0

  Block the currently running thread.
- virtual void unblock (nsbaci::types::ThreadID threadId)=0

  Move a thread from blocked to ready state.
- virtual void yield ()=0

  Yield the current thread (move to back of ready queue).
- virtual void terminateCurrent ()=0

  Terminate the currently running thread.
- virtual bool hasThreads () const =0

  Check if there are any threads left to run.
- virtual Thread * current ()=0

  Get the currently running thread.
- virtual void clear ()=0

  Clear all threads and reset scheduler state.
- virtual void unblockIO ()=0

  Move all I/O waiting threads back to ready queue.
- virtual const std::vector< Thread > & getThreads () const =0

  Get all threads managed by the scheduler.

Protected Attributes

- std::vector< Thread > **threads**

  All threads owned by scheduler.
- std::vector< size_t > **readyQueue**

  Indices of ready threads.
- std::vector< size_t > **blockedQueue**

  Indices of blocked threads.
- std::vector< size_t > **ioQueue**

  Indices of I/O waiting threads.
- std::optional< size_t > **runningIndex**

  Index of currently running thread.

6.41.1 Detailed Description

Manages thread scheduling and state transitions.

The Scheduler is responsible for determining which thread runs next, managing thread queues for different states, and handling thread state transitions.

6.41.2 Member Function Documentation

6.41.2.1 addThread()


```
virtual void nsbaci::services::runtime::Scheduler::addThread (
            Thread thread)  [pure virtual]
```

Add a new thread to the scheduler

Parameters

thread	The thread to add.

Implemented in nsbaci::services::runtime::NsbaciScheduler.

6.41.2.2 blockCurrent()

```
virtual void nsbaci::services::runtime::Scheduler::blockCurrent ()  [pure virtual]
```

Block the currently running thread.

Implemented in nsbaci::services::runtime::NsbaciScheduler.

6.41.2.3 clear()

```
virtual void nsbaci::services::runtime::Scheduler::clear ()  [pure virtual]
```

Clear all threads and reset scheduler state.

Implemented in nsbaci::services::runtime::NsbaciScheduler.

6.41.2.4 current()

```
virtual Thread * nsbaci::services::runtime::Scheduler::current ()  [pure virtual]
```

Get the currently running thread.

Returns

Pointer to current thread, or nullptr if none.

Implemented in nsbaci::services::runtime::NsbaciScheduler.

6.41.2.5 getThreads()

```
virtual const std::vector< Thread > & nsbaci::services::runtime::Scheduler::getThreads ()
const  [pure virtual]
```

Get all threads managed by the scheduler.

Returns

Const reference to the threads vector.

Implemented in nsbaci::services::runtime::NsbaciScheduler.

6.41.2.6 hasThreads()

```
virtual bool nsbaci::services::runtime::Scheduler::hasThreads () const  [pure virtual]
```

Check if there are any threads left to run.

Returns

True if there are threads in any queue.

Implemented in nsbaci::services::runtime::NsbaciScheduler.

6.41.2.7 pickNext()

```
virtual Thread * nsbaci::services::runtime::Scheduler::pickNext ()  [pure virtual]
```

Pick the next thread to run.

Returns

Pointer to the next thread, or nullptr if no threads are ready.

Implemented in nsbaci::services::runtime::NsbaciScheduler.

6.41.2.8 terminateCurrent()

```
virtual void nsbaci::services::runtime::Scheduler::terminateCurrent ()  [pure virtual]
```

Terminate the currently running thread.

Implemented in nsbaci::services::runtime::NsbaciScheduler.

6.41.2.9 unblock()

```
virtual void nsbaci::services::runtime::Scheduler::unblock (
            nsbaci::types::ThreadID threadId)  [pure virtual]
```

Move a thread from blocked to ready state.

Parameters

threadId	The ID of the thread to unblock.

Implemented in nsbaci::services::runtime::NsbaciScheduler.

6.41.2.10 unblockIO()

```
virtual void nsbaci::services::runtime::Scheduler::unblockIO ()  [pure virtual]
```

Move all I/O waiting threads back to ready queue.

Called when input becomes available.

Implemented in nsbaci::services::runtime::NsbaciScheduler.

6.41.2.11 yield()

```
virtual void nsbaci::services::runtime::Scheduler::yield ()  [pure virtual]
```

Yield the current thread (move to back of ready queue).

Implemented in nsbaci::services::runtime::NsbaciScheduler.

The documentation for this class was generated from the following file:

- scheduler.h

6.42 nsbaci::types::SymbolInfo Struct Reference

Information about a variable/symbol.

```
#include <compilerTypes.h>
```

Public Attributes

- VarName **name**
- MemoryAddr **address**
- std::string **type**
  "int", "bool", "char", "void", etc.
- bool **isGlobal**

6.42.1 Detailed Description

Information about a variable/symbol.

The documentation for this struct was generated from the following file:

- compilerTypes.h

6.43 nsbaci::services::runtime::Thread Class Reference

Represents a thread in the runtime service.

`#include <thread.h>`

Public Member Functions

- nsbaci::types::ThreadID getId () const
  Gets the thread ID.
- nsbaci::types::ThreadState getState () const
  Gets the current state of the thread.
- void setState (nsbaci::types::ThreadState newState)
  Sets the state of the thread.
- nsbaci::types::Priority getPriority () const
  Gets the priority of the thread.
- void setPriority (nsbaci::types::Priority newPriority)
  Sets the priority of the thread.
- void **push** (int32_t value)
  Push a value onto the thread's stack.
- int32_t **pop** ()
  Pop a value from the thread's stack.
- int32_t **top** () const
  Peek at the top of the stack without removing.
- uint32_t **getPC** () const
  Get the program counter.
- void **setPC** (uint32_t addr)
  Set the program counter.
- void **advancePC** ()
  Increment the program counter.
- uint32_t **getBP** () const
- void **setBP** (uint32_t addr)
- uint32_t **getSP** () const
- void **setSP** (uint32_t addr)

6.43.1 Detailed Description

Represents a thread in the runtime service.

Each thread has its own stack, program counter, and execution state.

6.43.2 Member Function Documentation

6.43.2.1 getId()

```
ThreadID nsbaci::services::runtime::Thread::getId () const
```

Gets the thread ID.

Returns

The unique identifier of this thread.

6.43.2.2 getPriority()

```
Priority nsbaci::services::runtime::Thread::getPriority () const
```

Gets the priority of the thread.

Returns

The current Priority level.

6.43.2.3 getState()

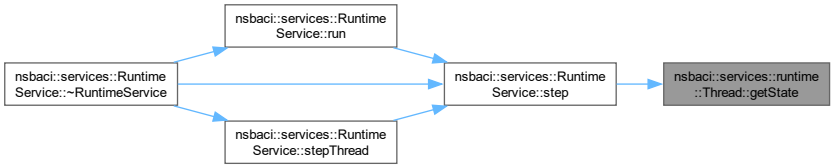
```
ThreadState nsbaci::services::runtime::Thread::getState () const
```

Gets the current state of the thread.

Returns

The current ThreadState.

Here is the caller graph for this function:



6.43.2.4 setPriority()

```
void nsbaci::services::runtime::Thread::setPriority (
            nsbaci::types::Priority newPriority)
```

Sets the priority of the thread.

Parameters

newPriority	The new priority level to set.

6.43.2.5 setState()

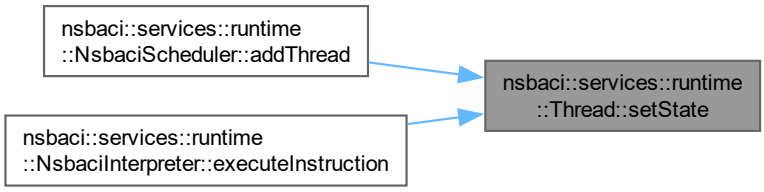
```
void nsbaci::services::runtime::Thread::setState (
            nsbaci::types::ThreadState newState)
```

Sets the state of the thread.

Parameters

newState	The new state to set.

Here is the caller graph for this function:



The documentation for this class was generated from the following files:

- thread.h
- thread.cpp

6.44 nsbaci::ui::ThreadInfo Struct Reference

Information about a thread for display.

`#include <runtimeView.h>`

Public Attributes

- nsbaci::types::ThreadID **id**
- nsbaci::types::ThreadState **state**
- size_t **pc**
- QString **currentInstruction**

6.44.1 Detailed Description

Information about a thread for display.

The documentation for this struct was generated from the following file:

- runtimeView.h

6.45 nsbaci::UIError Struct Reference

UI-ready error representation for display in dialogs.

`#include <uiError.h>`

Static Public Member Functions

- static std::vector< UIError > fromBackendErrors (const std::vector< Error > &errors)

  Converts backend errors to UI-ready errors.

Public Attributes

- QString **title**
- QString **body**
- nsbaci::types::ErrSeverity **severity**

6.45.1 Detailed Description

UI-ready error representation for display in dialogs.

Contains all the information needed to render an error dialog: title, body text, and severity (which maps to an icon).

6.45.2 Member Function Documentation

6.45.2.1 fromBackendErrors()

```
std::vector< UIError > nsbaci::UIError::fromBackendErrors (
            const std::vector< Error > & errors)  [static]
```

Converts backend errors to UI-ready errors.

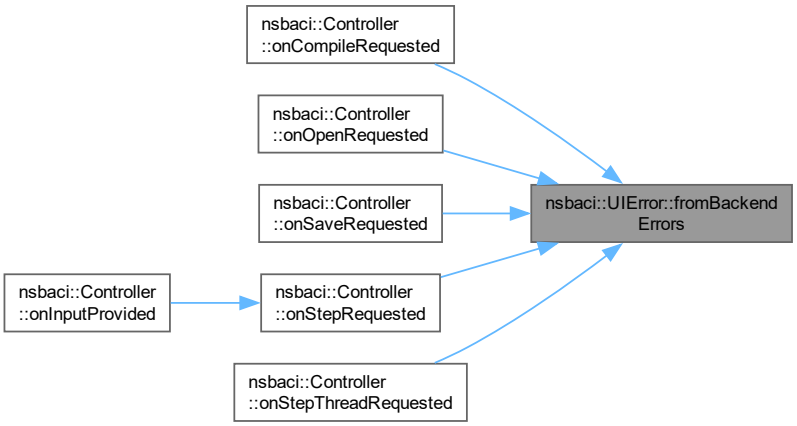
Parameters

errors	Vector of backend Error objects.

Returns

Vector of UIError objects ready for display.

Here is the caller graph for this function:



The documentation for this struct was generated from the following files:

- uiError.h
- uiError.cpp

6.46 nsbaci::ui::VariableInfo Struct Reference

Information about a variable for display.

`#include <runtimeView.h>`

Public Attributes

- QString **name**
- QString **type**
- QString **value**
- size_t **address**

6.46.1 Detailed Description

Information about a variable for display.

The documentation for this struct was generated from the following file:

- runtimeView.h

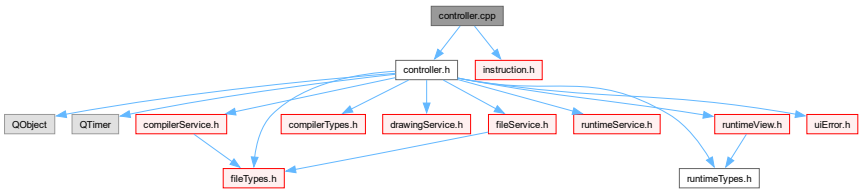
Chapter 7

File Documentation

7.1 controller.cpp File Reference

Implementation of the Controller class for nsbaci.

Include dependency graph for controller.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.

7.1.1 Detailed Description

Implementation of the Controller class for nsbaci.

This file contains the implementation of all Controller methods that coordinate between the user interface and backend services. It handles the complete workflow from file operations through compilation to runtime execution and monitoring.

The implementation uses Qt's signal-slot mechanism for asynchronous communication with the UI, and a QTimer-based approach for continuous program execution that maintains UI responsiveness.

Author

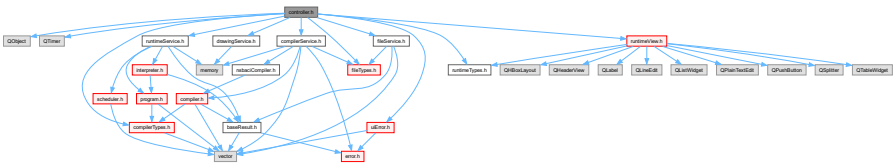
Nicolás Serrano García

Copyright

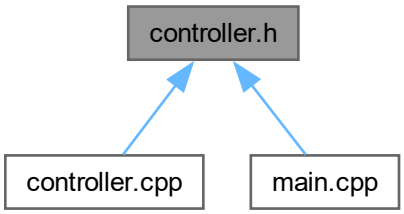
7.2 controller.h File Reference

Controller class declaration for nsbaci.

Include dependency graph for controller.h:



This graph shows which files directly or indirectly include this file:



Classes

- class nsbaci::Controller

  Central coordinator between UI and backend services.

Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.

7.2.1 Detailed Description

Controller class declaration for nsbaci.

This module defines the central Controller class that serves as the main coordination layer between the user interface and the backend services. The Controller implements the Model-View-Controller (MVC) pattern, acting as the mediator that handles user actions from the UI, delegates work to appropriate services, and communicates results back to the view layer.

The Controller manages the complete lifecycle of program execution including:

- File operations (save/load source files)
- Compilation workflow (source code to p-code instructions)
- Runtime execution control (run, step, pause, reset)
- Thread scheduling and monitoring
- Variable state tracking and display updates
- Input/output handling between runtime and UI

Author

Nicolás Serrano García

Copyright

7.3 controller.h

Go to the documentation of this file.

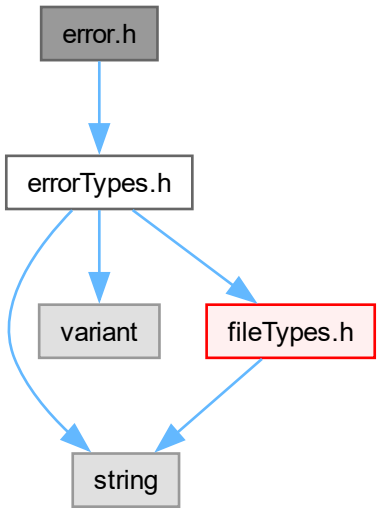
```
00001
00023
00024 #ifndef NSBACI_CONTROLLER_H
00025 #define NSBACI_CONTROLLER_H
00026
00027 #include <QObject>
00028 #include <QTimer>
00029
00030 #include "compilerService.h"
00031 #include "compilerTypes.h"
00032 #include "drawingService.h"
00033 #include "fileService.h"
00034 #include "fileTypes.h"
00035 #include "runtimeService.h"
00036 #include "runtimeTypes.h"
00037 #include "runtimeView.h"
00038 #include "uiError.h"
00039
00047 namespace nsbaci {
00048
00073 class Controller : public QObject {
00074   Q_OBJECT
00075
00076  public:
00089   explicit Controller(nsbaci::services::FileService&& f,
00090                       nsbaci::services::CompilerService&& c,
00091                       nsbaci::services::RuntimeService&& r,
00092                       nsbaci::services::DrawingService&& d,
00093                       QObject* parent = nullptr);
00094
00100   ~Controller() = default;
00101
00102  signals:
00107   void saveFailed(std::vector<UIError> errors);
00108
00112   void saveSucceeded();
00113
00118   void loadFailed(std::vector<UIError> errors);
00119
00124   void loadSucceeded(const QString& contents);
00125
00130   void compileFailed(std::vector<UIError> errors);
00131
00138   void compileSucceeded();
00139
00144   void runStarted(const QString& programName);
00145
00152   void runtimeStateChanged(bool running, bool halted);
00153
```

```
00159   void threadsUpdated(const std::vector<nsbaci::ui::ThreadInfo>& threads);
00160
00165   void variablesUpdated(const std::vector<nsbaci::ui::VariableInfo>& variables);
00166
00171   void outputReceived(const QString& output);
00172
00177   void inputRequested(const QString& prompt);
00178
00179  public slots:
00185   void onSaveRequested(nsbaci::types::File file, nsbaci::types::Text contents);
00186
00191   void onOpenRequested(nsbaci::types::File file);
00192
00202   void onCompileRequested(nsbaci::types::Text contents);
00203
00212   void onRunRequested();
00213
00220   void onStepRequested();
00221
00226   void onStepThreadRequested(nsbaci::types::ThreadID threadId);
00227
00234   void onRunContinueRequested();
00235
00241   void onPauseRequested();
00242
00249   void onResetRequested();
00250
00256   void onStopRequested();
00257
00267   void onInputProvided(const QString& input);
00268
00269  private:
00276   void updateRuntimeDisplay();
00277
00282   std::vector<nsbaci::ui::ThreadInfo> gatherThreadInfo();
00283
00288   std::vector<nsbaci::ui::VariableInfo> gatherVariableInfo();
00289
00297   void runBatch();
00298
00299   nsbaci::services::FileService
00300       fileService;
00301   nsbaci::services::CompilerService
00302       compilerService;
00303   nsbaci::services::RuntimeService
00304       runtimeService;
00305   nsbaci::services::DrawingService
00306       drawingService;
00307
00308   QString currentProgramName;
00309   bool programLoaded = false;
00310   bool isRunning = false;
00311   bool wasRunningBeforeInput =
00312       false;
00313   QTimer* runTimer = nullptr;
00314 };
00315
00316 }  // namespace nsbaci
00317
00318 #endif  // NSBACI_CONTROLLER_H
```

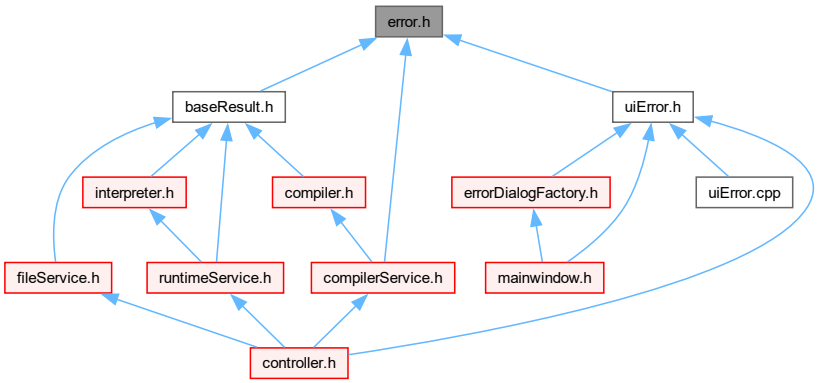
7.4 error.h File Reference

Error class declaration for nsbaci.

Include dependency graph for error.h:



This graph shows which files directly or indirectly include this file:



Classes

- class nsbaci::Error

  Represents an error with a message and optional code.

Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.

7.4.1 Detailed Description

Error class declaration for nsbaci.

This module defines the error handling structures and classes used throughout the application for consistent error reporting.

Author

Nicolás Serrano García

Copyright

7.5 error.h

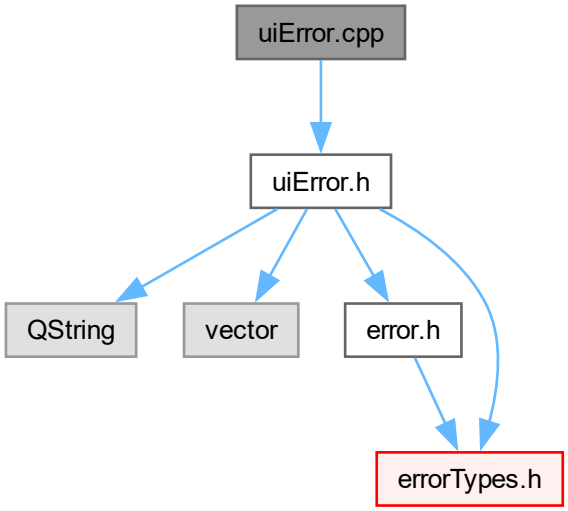
Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_ERROR_H
00014 #define NSBACI_ERROR_H
00015
00016 #include "errorTypes.h"
00017
00022 namespace nsbaci {
00023
00028 class Error {
00029  public:
00030   Error() = default;
00031   ~Error() = default;
00032   nsbaci::types::ErrorBase basic;
00033   nsbaci::types::ErrorPayload payload;
00034 };
00035
00036 }  // namespace nsbaci
00037
00038 #endif  // NSBACI_ERROR_H
```

7.6 uiError.cpp File Reference

Implementation of UIError utilities.

Include dependency graph for uiError.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.

7.6.1 Detailed Description

Implementation of UIError utilities.

Author

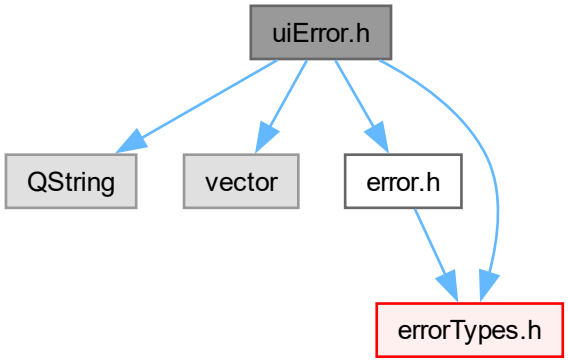
Nicolás Serrano García

Copyright

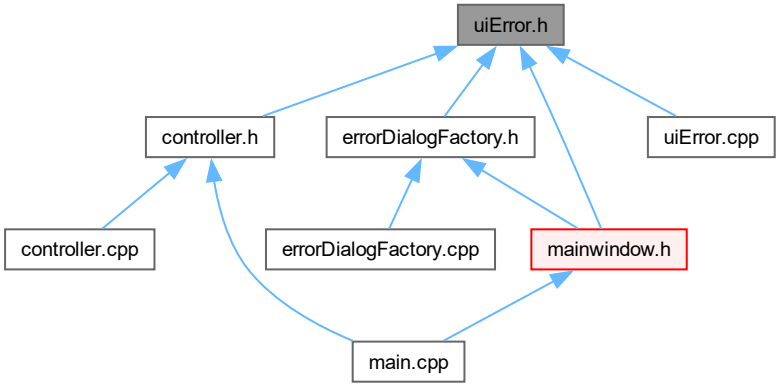
7.7 uiError.h File Reference

UI Error type definitions for nsbaci.

Include dependency graph for uiError.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct nsbaci::UIError

  UI-ready error representation for display in dialogs.

Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.

7.7.1 Detailed Description

UI Error type definitions for nsbaci.

This header provides types used to communicate between the controller and the view layer, particularly for error display.

Author

Nicolás Serrano García

Copyright

7.8 uiError.h

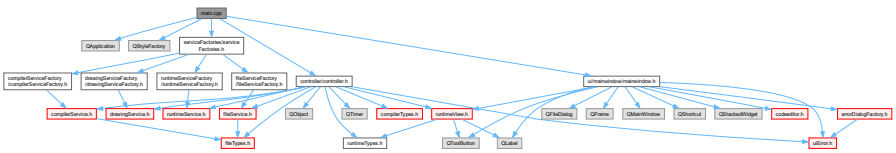
Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_UIERROR_H
00014 #define NSBACI_UIERROR_H
00015
00016 #include <QString>
00017 #include <vector>
00018
00019 #include "error.h"
00020 #include "errorTypes.h"
00021
00022 namespace nsbaci {
00023
00031 struct UIError {
00032   QString title;  // Dialog title (e.g., "Save Error", "Load Error")
00033   QString body;   // Main message body with details
00034   nsbaci::types::ErrSeverity severity;  // Maps to icon (Warning, Error, Fatal)
00035
00041   static std::vector<UIError> fromBackendErrors(
00042       const std::vector<Error>& errors);
00043
00044  private:
00048   static QString reasonFromType(nsbaci::types::ErrType type);
00049
00053   static QString titleFromPayload(const nsbaci::types::ErrorPayload& payload);
00054
00058   static QString contextFromPayload(const nsbaci::types::ErrorPayload& payload);
00059 };
00060
00061 }  // namespace nsbaci
00062
00063 #endif  // NSBACI_UIERROR_H
```

7.9 main.cpp File Reference

Application entry point for nsbaci.

Include dependency graph for main.cpp:



Functions

- void **setupViewController** (nsbaci::Controller ∗c, MainWindow ∗w)
- int **main** (int argc, char ∗argv[ ])

7.9.1 Detailed Description

Application entry point for nsbaci.

This file contains the main function that initializes the Qt application and displays the main window.

Author

Nicolás Serrano García

Copyright

Copyright (c) 2025 Nicolás Serrano García. Licensed under the MIT License.

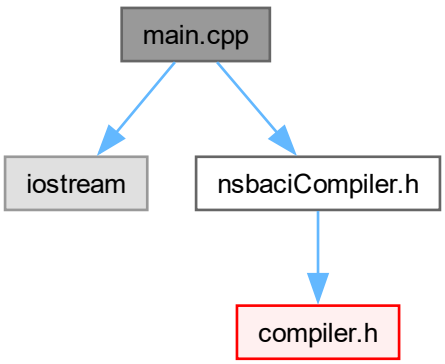
See also

MainWindow

7.10 services/compilerService/compiler/nsbaci/tools/main.cpp File Reference

CLI tool for testing the nsbaci compiler.

Include dependency graph for services/compilerService/compiler/nsbaci/tools/main.cpp:



Functions

- int **main** ()

7.10.1 Detailed Description

CLI tool for testing the nsbaci compiler.

This is a standalone command-line tool for testing the compiler independently of the main application.

Author

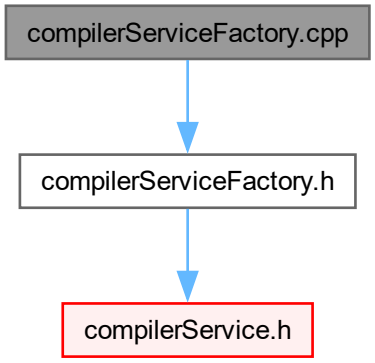
Nicolás Serrano García

Copyright

7.11 compilerServiceFactory.cpp File Reference

Implementation unit for the CompilerServiceFactory.

Include dependency graph for compilerServiceFactory.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::factories

  Factories namespace for nsbaci.

7.11.1 Detailed Description

Implementation unit for the CompilerServiceFactory.

Author

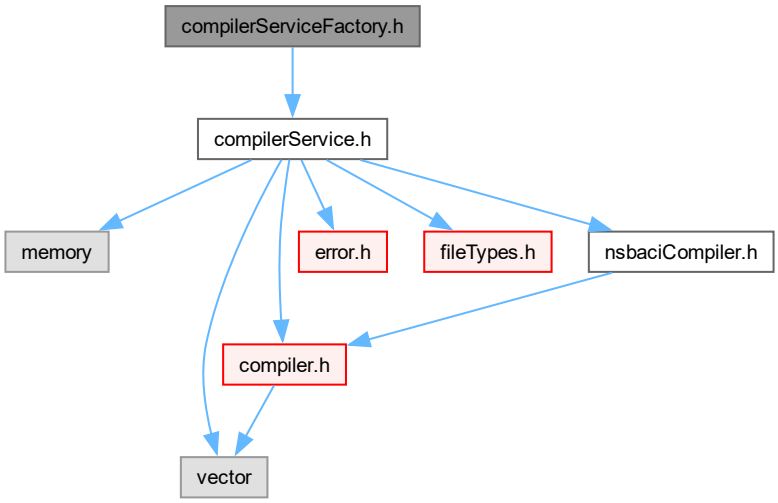
Nicolás Serrano García

Copyright

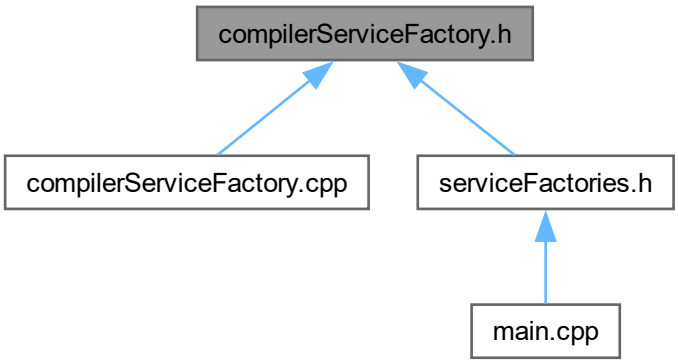
7.12 compilerServiceFactory.h File Reference

CompilerServiceFactory class declaration for nsbaci.

Include dependency graph for compilerServiceFactory.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct nsbaci::factories::NsbaciCompiler
- class nsbaci::factories::CompilerServiceFactory

  Factory for creating CompilerService instances.

Namespaces

- namespace nsbaci::factories

  Factories namespace for nsbaci.
- namespace nsbaci

  Root namespace for the nsbaci application.

Variables

- constexpr NsbaciCompiler **nsbaci::factories::nsbaciCompiler** {}

7.12.1 Detailed Description

CompilerServiceFactory class declaration for nsbaci.

This factory is responsible for creating CompilerService instances based on configuration or runtime parameters.

Author

Nicolás Serrano García

Copyright

7.13 compilerServiceFactory.h

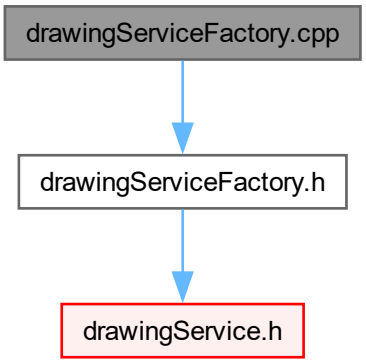
Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_COMPILERSERVICEFACTORY_H
00014 #define NSBACI_COMPILERSERVICEFACTORY_H
00015
00016 #include "compilerService.h"
00017
00022 namespace nsbaci::factories {
00023
00024 struct NsbaciCompiler {
00025   explicit NsbaciCompiler() = default;
00026 };
00027
00028 // Tag used to generate a service with a nsbaci compiler
00029 constexpr inline NsbaciCompiler nsbaciCompiler{};
00030
00035 class CompilerServiceFactory {
00036  private:
00037   CompilerServiceFactory() = default;
00038
00039  public:
00040   static nsbaci::services::CompilerService createService(NsbaciCompiler t);
00041   // static nsbaci::services::CompilerService createService(OtherCompilerOrTest
00042   // t);
00043   ~CompilerServiceFactory() = default;
00044 };
00045
00046 }  // namespace nsbaci::factories
00047
00048 #endif  // NSBACI_COMPILERSERVICEFACTORY_H
```

7.14 drawingServiceFactory.cpp File Reference

Implementation unit for the DrawingServiceFactory.

Include dependency graph for drawingServiceFactory.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::factories

  Factories namespace for nsbaci.

7.14.1 Detailed Description

Implementation unit for the DrawingServiceFactory.

Author

Nicolás Serrano García

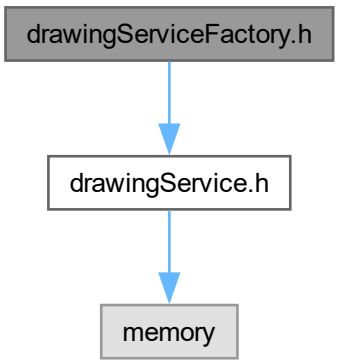
Copyright

Copyright (c) 2025 Nicolás Serrano García. Licensed under the MIT License.

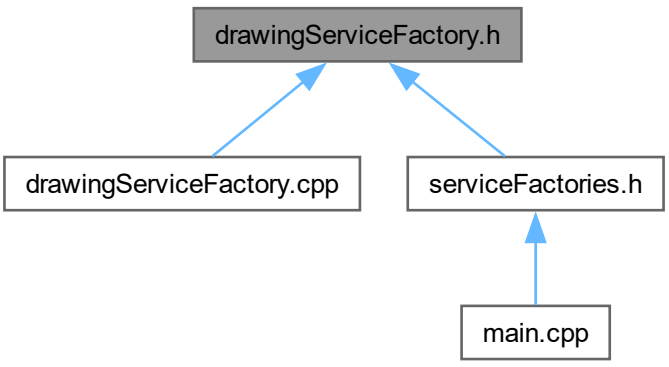
7.15 drawingServiceFactory.h File Reference

DrawingServiceFactory class declaration for nsbaci.

Include dependency graph for drawingServiceFactory.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct nsbaci::factories::DefaultDrawingBackend
- class nsbaci::factories::DrawingServiceFactory

  Factory for creating DrawingService instances.

Namespaces

- namespace nsbaci::factories

  Factories namespace for nsbaci.
- namespace nsbaci

  Root namespace for the nsbaci application.

Variables

- constexpr DefaultDrawingBackend **nsbaci::factories::defaultDrawingBackend** {}

7.15.1 Detailed Description

DrawingServiceFactory class declaration for nsbaci.

This factory is responsible for creating DrawingService instances based on configuration or runtime parameters.

Author

Nicolás Serrano García

Copyright

7.16 drawingServiceFactory.h

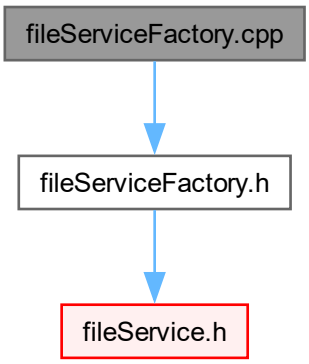
Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_DRAWINGSERVICEFACTORY_H
00014 #define NSBACI_DRAWINGSERVICEFACTORY_H
00015
00016 #include "drawingService.h"
00017
00022 namespace nsbaci::factories {
00023
00024 struct DefaultDrawingBackend {
00025   explicit DefaultDrawingBackend() = default;
00026 };
00027
00028 // tag used to generate a service with the default drawing backend
00029 constexpr inline DefaultDrawingBackend defaultDrawingBackend{};
00030
00035 class DrawingServiceFactory {
00036  private:
00037   DrawingServiceFactory() = default;
00038
00039  public:
00040   static nsbaci::services::DrawingService createService(
00041       DefaultDrawingBackend t);
00042   // static nsbaci::services::DrawingService createService(OtherBackendOrTest
00043   // t);
00044   ~DrawingServiceFactory() = default;
00045 };
00046
00047 }  // namespace nsbaci::factories
00048
00049 #endif  // NSBACI_DRAWINGSERVICEFACTORY_H
```

7.17 fileServiceFactory.cpp File Reference

Implementation unit for the FileServiceFactory.

Include dependency graph for fileServiceFactory.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::factories

  Factories namespace for nsbaci.

7.17.1 Detailed Description

Implementation unit for the FileServiceFactory.

Author

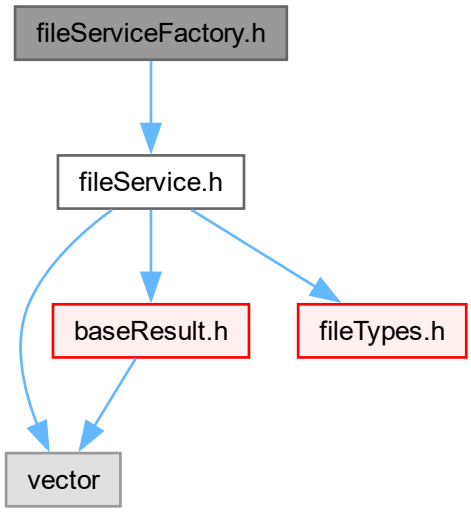
Nicolás Serrano García

Copyright

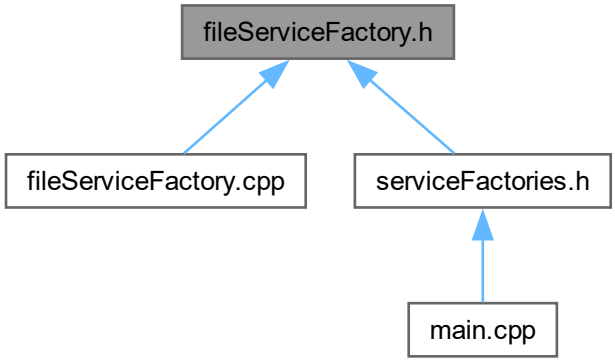
7.18 fileServiceFactory.h File Reference

FileServiceFactory class declaration for nsbaci.

Include dependency graph for fileServiceFactory.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct nsbaci::factories::DefaultFileSystem
- class nsbaci::factories::FileServiceFactory

  Factory for creating FileService instances.

Namespaces

- namespace nsbaci::factories

  Factories namespace for nsbaci.
- namespace nsbaci

  Root namespace for the nsbaci application.

Variables

- constexpr DefaultFileSystem **nsbaci::factories::defaultFileSystem** {}

7.18.1 Detailed Description

FileServiceFactory class declaration for nsbaci.

This factory is responsible for creating FileService instances based on configuration or runtime parameters.

Author

Nicolás Serrano García

Copyright

7.19 fileServiceFactory.h

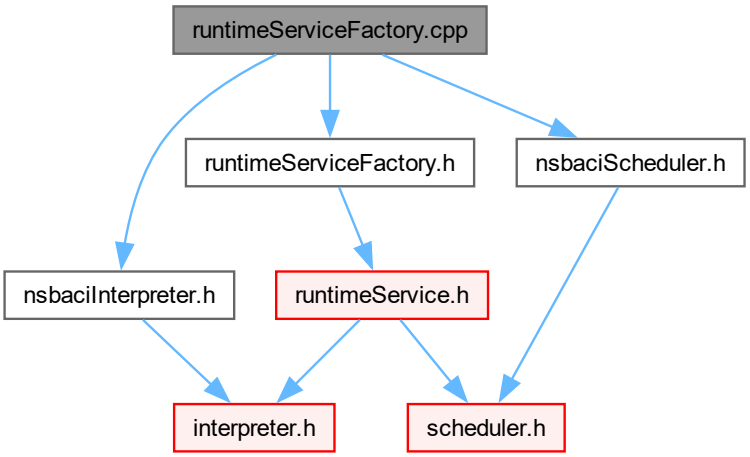
Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_FILESERVICEFACTORY_H
00014 #define NSBACI_FILESERVICEFACTORY_H
00015
00016 #include "fileService.h"
00017
00022 namespace nsbaci::factories {
00023
00024 struct DefaultFileSystem {
00025   explicit DefaultFileSystem() = default;
00026 };
00027
00028 // Tag used to generate a service with the default file system
00029 constexpr inline DefaultFileSystem defaultFileSystem{};
00030
00035 class FileServiceFactory {
00036  private:
00037   FileServiceFactory() = default;
00038
00039  public:
00040   static nsbaci::services::FileService createService(DefaultFileSystem t);
00041   // static nsbaci::services::FileService createService(OtherFileSystemOrTest
00042   // t);
00043   ~FileServiceFactory() = default;
00044 };
00045
00046 }  // namespace nsbaci::factories
00047
00048 #endif  // NSBACI_FILESERVICEFACTORY_H
```

7.20 runtimeServiceFactory.cpp File Reference

Implementation unit for the RuntimeServiceFactory.

Include dependency graph for runtimeServiceFactory.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::factories

  Factories namespace for nsbaci.

7.20.1 Detailed Description

Implementation unit for the RuntimeServiceFactory.

Author

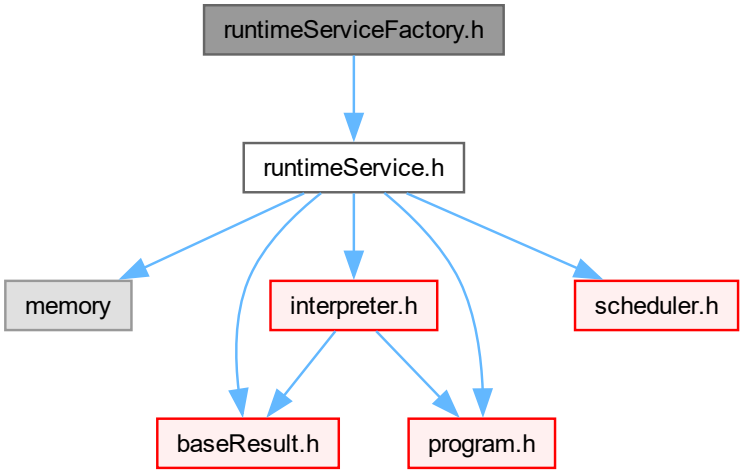
Nicolás Serrano García

Copyright

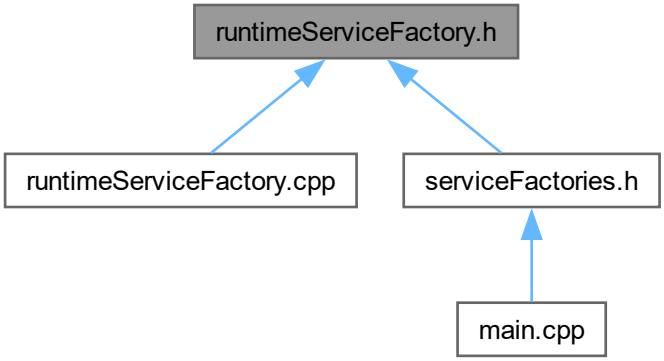
7.21 runtimeServiceFactory.h File Reference

RuntimeServiceFactory class declaration for nsbaci.

Include dependency graph for runtimeServiceFactory.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct nsbaci::factories::NsbaciRuntime
- class nsbaci::factories::RuntimeServiceFactory

  Factory for creating RuntimeService instances.

Namespaces

- namespace nsbaci::factories

  Factories namespace for nsbaci.
- namespace nsbaci

  Root namespace for the nsbaci application.

Variables

- constexpr NsbaciRuntime **nsbaci::factories::nsbaciRuntime** {}

7.21.1 Detailed Description

RuntimeServiceFactory class declaration for nsbaci.

This factory is responsible for creating RuntimeService instances based on configuration or runtime parameters.

Author

Nicolás Serrano García

Copyright

7.22 runtimeServiceFactory.h

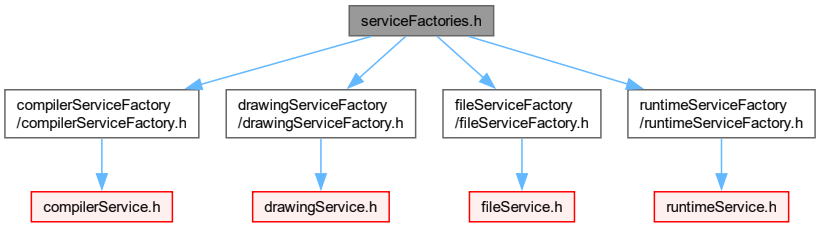
Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_RUNTIMESERVICEFACTORY_H
00014 #define NSBACI_RUNTIMESERVICEFACTORY_H
00015
00016 #include "runtimeService.h"
00017
00022 namespace nsbaci::factories {
00023
00024 struct NsbaciRuntime {
00025   explicit NsbaciRuntime() = default;
00026 };
00027
00028 // Tag used to generate a service with a nsbaci runtime
00029 constexpr inline NsbaciRuntime nsbaciRuntime{};
00030
00035 class RuntimeServiceFactory {
00036  private:
00037   RuntimeServiceFactory() = default;
00038
00039  public:
00040   static nsbaci::services::RuntimeService createService(NsbaciRuntime t);
00041   // static nsbaci::services::RuntimeService createService(OtherRuntimeOrTest
00042   // t);
00043   ~RuntimeServiceFactory() = default;
00044 };
00045
00046 }  // namespace nsbaci::factories
00047
00048 #endif  // NSBACI_RUNTIMESERVICEFACTORY_H
```

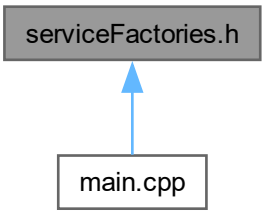
7.23 serviceFactories.h File Reference

Aggregate header for all service factories.

Include dependency graph for serviceFactories.h:



This graph shows which files directly or indirectly include this file:



7.23.1 Detailed Description

Aggregate header for all service factories.

This header includes all service factory headers for convenient access.

Author

Nicolás Serrano García

Copyright

7.24 serviceFactories.h

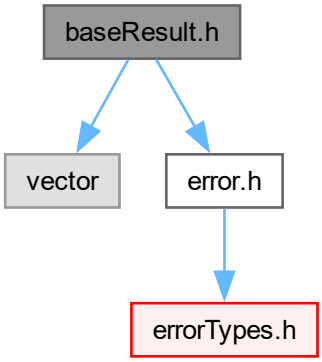
Go to the documentation of this file.

```
00001
00011
00012 #ifndef NSBACI_SERVICEFACTORIES_H
00013 #define NSBACI_SERVICEFACTORIES_H
00014
00015 #include "compilerServiceFactory/compilerServiceFactory.h"
00016 #include "drawingServiceFactory/drawingServiceFactory.h"
00017 #include "fileServiceFactory/fileServiceFactory.h"
00018 #include "runtimeServiceFactory/runtimeServiceFactory.h"
00019
00020 #endif  // NSBACI_SERVICEFACTORIES_H
```

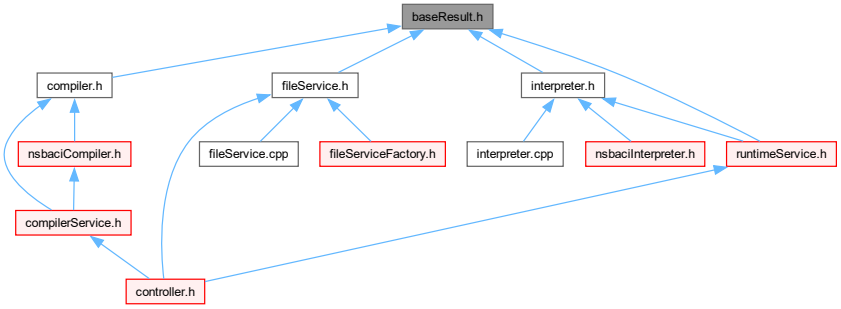
7.25 baseResult.h File Reference

Base result class declaration for nsbaci services.

Include dependency graph for baseResult.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct nsbaci::BaseResult

  Base result structure for all service operations.

Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.

7.25.1 Detailed Description

Base result class declaration for nsbaci services.

This module defines the BaseResult struct that serves as the foundational result type for all service operations in the nsbaci application. It implements a simple success/failure pattern with associated error information, providing a consistent interface for error handling across all services.

The BaseResult pattern ensures that:

- All service operations return a result that can be checked for success
- Failed operations always provide descriptive error information
- Results can be efficiently moved without copying error data

Author

Nicolás Serrano García

Copyright

7.26 baseResult.h

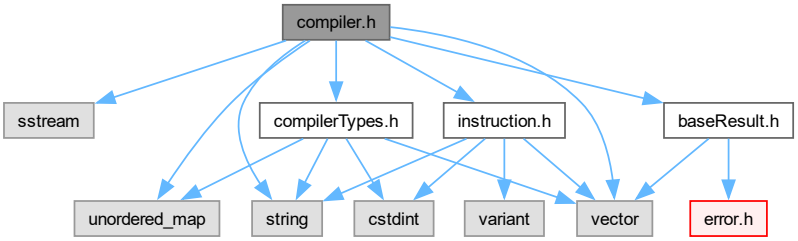
Go to the documentation of this file.

```
00001
00020
00021 #ifndef NSBACI_SERVICES_BASERESULT_H
00022 #define NSBACI_SERVICES_BASERESULT_H
00023
00024 #include <vector>
00025
00026 #include "error.h"
00027
00032 namespace nsbaci {
00033
00056 struct BaseResult {
00060   BaseResult() : ok(true) {}
00061
00069   explicit BaseResult(std::vector<nsbaci::Error> errs)
00070       : ok(errs.empty()), errors(std::move(errs)) {}
00071
00076   explicit BaseResult(nsbaci::Error error)
00077       : ok(false), errors({std::move(error)}) {}
00078
00079   BaseResult(BaseResult&&) noexcept = default;
00080   BaseResult& operator=(BaseResult&&) noexcept = default;
00081
00082   BaseResult(const BaseResult&) = default;
00083   BaseResult& operator=(const BaseResult&) = default;
00084
00085   bool ok;
00086   std::vector<nsbaci::Error>
00087       errors;
00088 };
00089
00090 }  // namespace nsbaci
00091
00092 #endif  // NSBACI_SERVICES_BASERESULT_H
```

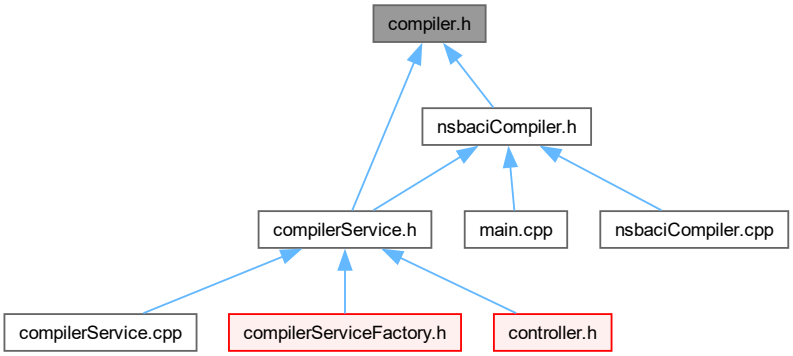
7.27 compiler.h File Reference

Abstract Compiler class declaration for nsbaci.

Include dependency graph for compiler.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct nsbaci::compiler::CompilerResult

  Result of a compilation operation.
- class nsbaci::compiler::Compiler

  Abstract base class for all compilers.

Namespaces

- namespace nsbaci::compiler

  Compiler namespace containing all compilation-related stuff.
- namespace nsbaci

  Root namespace for the nsbaci application.

7.27.1 Detailed Description

Abstract Compiler class declaration for nsbaci.

This module defines the abstract Compiler interface that all compiler implementations must follow. The Compiler is responsible for transforming nsbaci source code into an executable instruction stream (p-code) that can be run by the virtual machine interpreter.

The design follows the Strategy pattern, allowing different compiler implementations to be plugged in. Currently, NsbaciCompiler is the primary implementation using flex/bison for lexical analysis and parsing.

Author

Nicolás Serrano García

Copyright

7.28 compiler.h

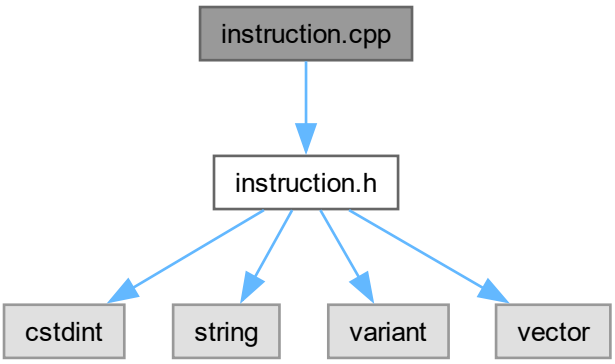
Go to the documentation of this file.

```
00001
00018
00019 #ifndef NSBACI_COMPILER_COMPILER_H
00020 #define NSBACI_COMPILER_COMPILER_H
00021
00022 #include <sstream>
00023 #include <string>
00024 #include <unordered_map>
00025 #include <vector>
00026
00027 #include "baseResult.h"
00028 #include "compilerTypes.h"
00029 #include "instruction.h"
00030
00035 namespace nsbaci::compiler {
00036
00047 struct CompilerResult : nsbaci::BaseResult {
00051   CompilerResult() : BaseResult() {}
00052
00057   explicit CompilerResult(std::vector<nsbaci::Error> errs)
00058       : BaseResult(std::move(errs)) {}
00059
00064   explicit CompilerResult(nsbaci::Error error) : BaseResult(std::move(error)) {}
00065
00066   CompilerResult(CompilerResult&&) noexcept = default;
00067   CompilerResult& operator=(CompilerResult&&) noexcept = default;
00068
00069   CompilerResult(const CompilerResult&) = default;
00070   CompilerResult& operator=(const CompilerResult&) = default;
00071
00072   InstructionStream instructions;
00073   nsbaci::types::SymbolTable symbols;
00074 };
00075
00092 class Compiler {
00093  public:
00097   Compiler() = default;
00098
00102   virtual ~Compiler() = default;
00103
00114   virtual CompilerResult compile(const std::string& source) = 0;
00115
00125   virtual CompilerResult compile(std::istream& input) = 0;
00126 };
00127
00128 }  // namespace nsbaci::compiler
00129
00130 #endif  // NSBACI_COMPILER_COMPILER_H
```

7.29 instruction.cpp File Reference

Instruction implementation for nsbaci compiler.

Include dependency graph for instruction.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::compiler

  Compiler namespace containing all compilation-related stuff.

Functions

- const char * nsbaci::compiler::opcodeName (Opcode op)

  Get the string name of an opcode (for debugging/display).

7.29.1 Detailed Description

Instruction implementation for nsbaci compiler.

Author

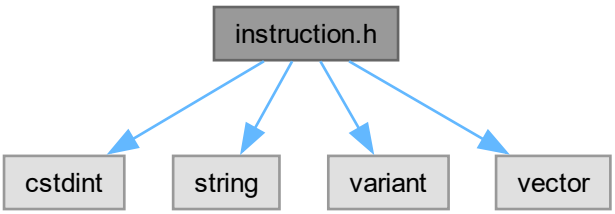
Nicolás Serrano García

Copyright

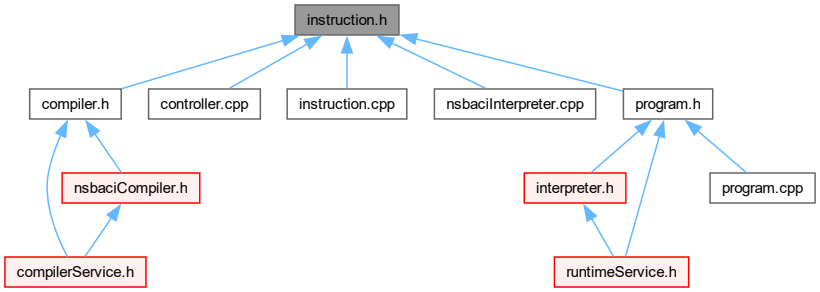
7.30 instruction.h File Reference

Instruction definitions for nsbaci compiler.

Include dependency graph for instruction.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct nsbaci::compiler::Instruction

  Represents a single instruction in the virtual machine.

Namespaces

- namespace nsbaci::compiler

  Compiler namespace containing all compilation-related stuff.
- namespace nsbaci

  Root namespace for the nsbaci application.

Typedefs

- using nsbaci::compiler::Operand

  Operand types that an instruction can have.
- using **nsbaci::compiler::InstructionStream** = std::vector<Instruction>

  Vector of instructions representing a compiled program.

Enumerations

- enum class nsbaci::compiler::Opcode : uint8_t {
  LoadValue , **LoadAddress** , **LoadIndirect** , **LoadBlock** ,
  Store , **StoreKeep** , **PushLiteral** , **Index** ,
  CopyBlock , **ValueAt** , **MarkStack** , **UpdateDisplay** ,
  Add , **Sub** , **Mult** , **Div** ,
  Mod , **Negate** , **Complement** , **And** ,
  Or , **TestEQ** , **TestNE** , **TestLT** ,
  TestLE , **TestGT** , **TestGE** , **TestEqualKeep** ,
  Jump , **JumpZero** , **Call** , **ShortCall** ,
  ShortReturn , **ExitProc** , **ExitFunction** , **Halt** ,
  BeginFor , **EndFor** , **Cobegin** , **Coend** ,
  Create , **Suspend** , **Revive** , **WhichProc** ,
  Wait , **Signal** , **StoreSemaphore** , **EnterMonitor** ,
  ExitMonitor , **CallMonitorInit** , **ReturnMonitorInit** , **WaitCondition** ,
  SignalCondition , **Empty** , **Read** , **ReadIn** ,
  Write , **WriteIn** , **WriteString** , **WriteRawString** ,
  EolEof , **Sprintf** , **Sscanf** , **CopyString** ,
  CopyRawString , **ConcatString** , **ConcatRawString** , **CompareString** ,
  CompareRawString , **LengthString** , **MoveTo** , **MoveBy** ,
  ChangeColor , **MakeVisible** , **Remove** , **Random** ,
  Test , **_Count** }

  Opcodes for the BACI virtual machine instruction set.

Functions

- const char * nsbaci::compiler::opcodeName (Opcode op)

  Get the string name of an opcode (for debugging/display).

7.30.1 Detailed Description

Instruction definitions for nsbaci compiler.

This module defines the instruction set for the BACI virtual machine, including opcodes and instruction representation.

Author

Nicolás Serrano García

Copyright

7.31 instruction.h

Go to the documentation of this file.

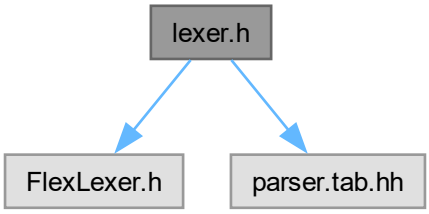
```
00001
00012
00013 #ifndef NSBACI_COMPILER_INSTRUCTION_H
00014 #define NSBACI_COMPILER_INSTRUCTION_H
00015
00016 #include <cstdint>
00017 #include <string>
00018 #include <variant>
00019 #include <vector>
00020
00025 namespace nsbaci::compiler {
00026
00031 enum class Opcode : uint8_t {
00032   // ============== Stack/Memory Operations ==============
00033   LoadValue,       // Load value from address onto stack
00034   LoadAddress,     // Load address onto stack
00035   LoadIndirect,    // Load value from address pointed to by top of stack
00036   LoadBlock,       // Load block of memory onto stack
00037   Store,           // Store top of stack to address
00038   StoreKeep,       // Store and keep value on stack
00039   PushLiteral,     // Push literal value onto stack
00040   Index,           // Array indexing
00041   CopyBlock,       // Copy block of memory
00042   ValueAt,         // Get value at address on stack
00043   MarkStack,       // Mark stack for procedure call
00044   UpdateDisplay,   // Update display register
00045
00046   // ============== Arithmetic Operations ==============
00047   Add,          // Addition
00048   Sub,          // Subtraction
00049   Mult,         // Multiplication
00050   Div,          // Integer division
00051   Mod,          // Modulo
00052   Negate,       // Unary negation
00053   Complement,   // Bitwise complement
00054
00055   // ============== Logical Operations ==============
00056   And,  // Logical AND
00057   Or,   // Logical OR
00058
00059   // ============== Comparison Operations ==============
00060   TestEQ,          // Test equal
00061   TestNE,          // Test not equal
00062   TestLT,          // Test less than
00063   TestLE,          // Test less or equal
00064   TestGT,          // Test greater than
00065   TestGE,          // Test greater or equal
00066   TestEqualKeep,   // Test equal, keep operands
00067
00068   // ============== Control Flow ==============
00069   Jump,           // Unconditional jump
00070   JumpZero,       // Jump if top of stack is zero
00071   Call,           // Call procedure
00072   ShortCall,      // Short call (no display update)
00073   ShortReturn,    // Short return
00074   ExitProc,       // Exit procedure
00075   ExitFunction,   // Exit function (with return value)
00076   Halt,           // Halt execution
00077
00078   // ============== Loop Control ==============
00079   BeginFor,   // Begin for loop
00080   EndFor,     // End for loop
00081
00082   // ============== Concurrency - Process ==============
00083   Cobegin,     // Begin concurrent block
00084   Coend,       // End concurrent block
00085   Create,      // Create new process
00086   Suspend,     // Suspend current process
00087   Revive,      // Revive suspended process
00088   WhichProc,   // Get current process ID
00089
00090   // ============== Concurrency - Semaphores ==============
00091   Wait,             // Wait on semaphore (P operation)
00092   Signal,           // Signal semaphore (V operation)
00093   StoreSemaphore,   // Initialize semaphore
00094
00095   // ============== Concurrency - Monitors ==============
00096   EnterMonitor,        // Enter monitor
00097   ExitMonitor,         // Exit monitor
00098   CallMonitorInit,     // Call monitor initialization
00099   ReturnMonitorInit,   // Return from monitor init
00100   WaitCondition,       // Wait on condition variable
```

```
00101   SignalCondition,    // Signal condition variable
00102   Empty,              // Check if condition queue is empty
00103
00104   // ============== I/O Operations ==============
00105   Read,            // Read integer
00106   Readln,          // Read line
00107   Write,           // Write value
00108   Writeln,         // Write newline
00109   WriteString,     // Write string
00110   WriteRawString,  // Write raw string literal
00111   EolEof,          // Check end of line/file
00112   Sprintf,         // Format string
00113   Sscanf,          // Scan string
00114
00115   // ============== String Operations ==============
00116   CopyString,        // Copy string
00117   CopyRawString,     // Copy raw string
00118   ConcatString,      // Concatenate strings
00119   ConcatRawString,   // Concatenate raw string
00120   CompareString,     // Compare strings
00121   CompareRawString,  // Compare raw strings
00122   LengthString,      // Get string length
00123
00124   // ============== Graphics Operations ==============
00125   MoveTo,       // Move to absolute position
00126   MoveBy,       // Move by relative offset
00127   ChangeColor,  // Change drawing color
00128   MakeVisible,  // Make object visible
00129   Remove,       // Remove object
00130
00131   // ============== Miscellaneous ==============
00132   Random,  // Generate random number
00133   Test,    // Generic test instruction
00134
00135   // ============== Total count ==============
00136   _Count  // Number of opcodes (keep last)
00137 };
00138
00142 using Operand = std::variant<std::monostate,  // No operand
00143                              int32_t,         // Integer literal or offset
00144                              uint32_t,        // Unsigned value or address
00145                              std::string      // String literal
00146                              >;
00147
00152 struct Instruction {
00153   Opcode opcode;
00154   Operand operand1;
00155   Operand operand2;
00156
00157   // Convenience constructors
00158   Instruction() : opcode(Opcode::Halt), operand1(), operand2() {}
00159
00160   explicit Instruction(Opcode op) : opcode(op), operand1(), operand2() {}
00161
00162   Instruction(Opcode op, int32_t op1) : opcode(op), operand1(op1), operand2() {}
00163
00164   Instruction(Opcode op, uint32_t op1)
00165       : opcode(op), operand1(op1), operand2() {}
00166
00167   Instruction(Opcode op, std::string op1)
00168       : opcode(op), operand1(std::move(op1)), operand2() {}
00169
00170   Instruction(Opcode op, int32_t op1, int32_t op2)
00171       : opcode(op), operand1(op1), operand2(op2) {}
00172 };
00173
00175 using InstructionStream = std::vector<Instruction>;
00176
00182 const char* opcodeName(Opcode op);
00183
00184 }  // namespace nsbaci::compiler
00185
00186 #endif  // NSBACI_COMPILER_INSTRUCTION_H
```

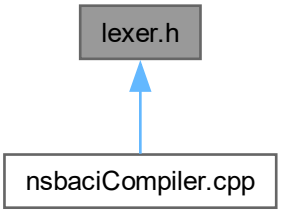
7.32 lexer.h File Reference

Lexer class declaration for nsbaci compiler.

Include dependency graph for lexer.h:



This graph shows which files directly or indirectly include this file:



Classes

- class nsbaci::compiler::Lexer

  Flex-based lexer for BACI source code.

Namespaces

- namespace nsbaci::compiler

  Compiler namespace containing all compilation-related stuff.
- namespace nsbaci

  Root namespace for the nsbaci application.

7.32.1 Detailed Description

Lexer class declaration for nsbaci compiler.

Author

Nicolás Serrano García

Copyright

7.33 lexer.h

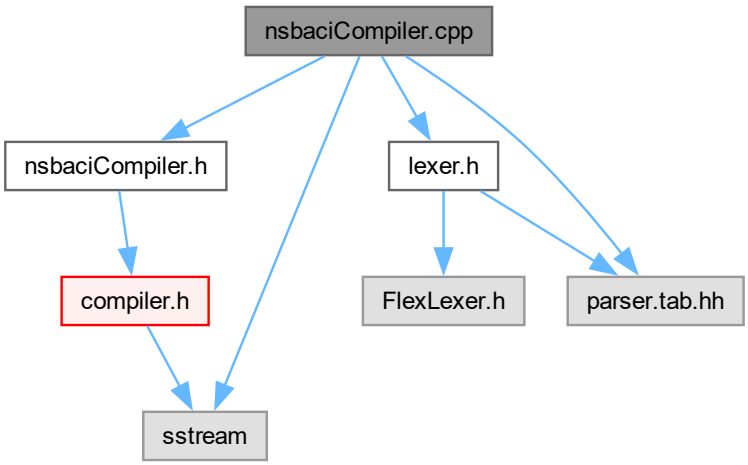
Go to the documentation of this file.

```
00001 #pragma once
00002
00011
00012 #if !defined(yyFlexLexerOnce)
00013 #include <FlexLexer.h>
00014 #endif
00015
00016 #include "parser.tab.hh"
00017
00022 namespace nsbaci::compiler {
00023
00028 class Lexer : public yyFlexLexer {
00029  public:
00030   Lexer(std::istream* in = nullptr) : yyFlexLexer(in) {}
00031   virtual ~Lexer() = default;
00032
00033   int yylex(Parser::semantic_type* yylval, Parser::location_type* yylloc);
00034 };
00035
00036 }  // namespace nsbaci::compiler
```

7.34 nsbaciCompiler.cpp File Reference

NsbaciCompiler class implementation for nsbaci.

Include dependency graph for nsbaciCompiler.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::compiler

  Compiler namespace containing all compilation-related stuff.

7.34.1 Detailed Description

NsbaciCompiler class implementation for nsbaci.

This file contains the implementation of the NsbaciCompiler class which uses flex-generated lexer and bison-generated parser to compile nsbaci source code into p-code instructions for the virtual machine.

Author

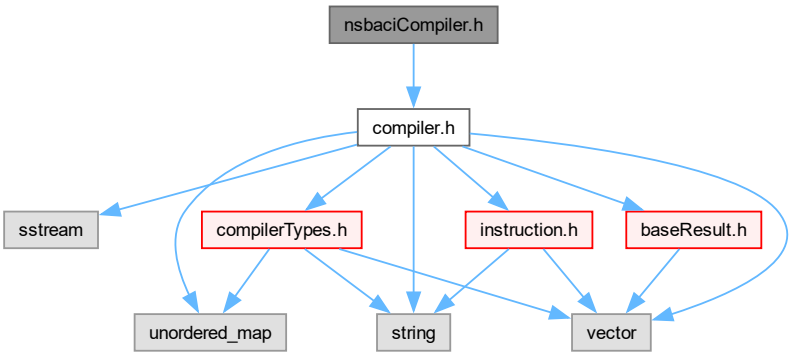
Nicolás Serrano García

Copyright

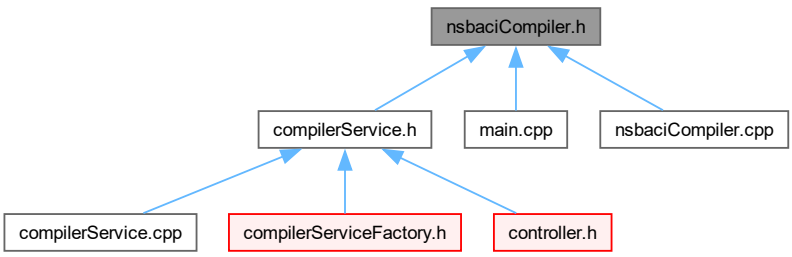
7.35 nsbaciCompiler.h File Reference

NsbaciCompiler class declaration for nsbaci.

Include dependency graph for nsbaciCompiler.h:



This graph shows which files directly or indirectly include this file:



Classes

- class nsbaci::compiler::NsbaciCompiler

  nsbaci compiler implementation using flex and bison.

Namespaces

- namespace nsbaci::compiler

  Compiler namespace containing all compilation-related stuff.
- namespace nsbaci

  Root namespace for the nsbaci application.

7.35.1 Detailed Description

NsbaciCompiler class declaration for nsbaci.

This module provides the concrete nsbaci implementation of the Compiler interface. NsbaciCompiler uses flex for lexical analysis and bison for parsing to compile BACI source code into p-code instructions.

The compiler supports:

- Basic types: int, bool, char
- Arithmetic and comparison operators
- Control flow: if/else, while, for loops
- C++ style I/O: cout << and cin >>
- Variable declarations with optional initialization
- Compound assignment operators (+=, -=, etc.)

Future features (not yet implemented):

- Functions and procedures
- Concurrency primitives (cobegin/coend, semaphores, monitors)
- Arrays and strings

Author

Nicolás Serrano García

Copyright

7.36 nsbaciCompiler.h

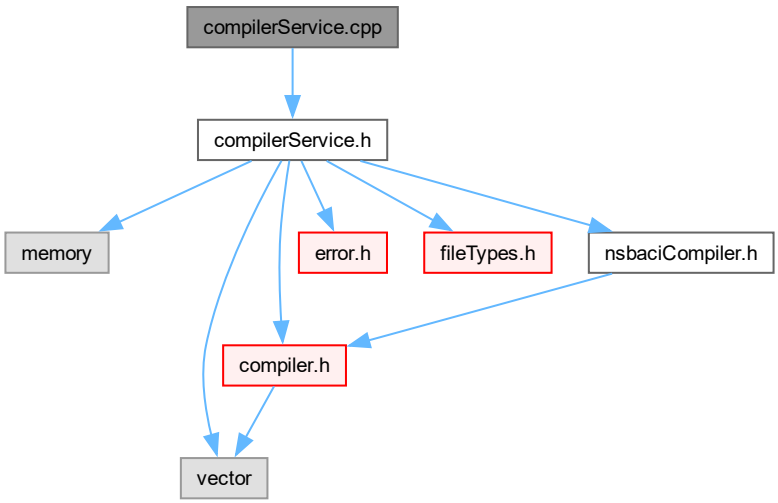
Go to the documentation of this file.

```
00001
00026
00027 #ifndef NSBACI_COMPILER_NSBACI_COMPILER_H
00028 #define NSBACI_COMPILER_NSBACI_COMPILER_H
00029
00030 #include "compiler.h"
00031
00036 namespace nsbaci::compiler {
00037
00054 class NsbaciCompiler final : public Compiler {
00055  public:
00059   NsbaciCompiler() = default;
00060
00064   ~NsbaciCompiler() override = default;
00065
00076   CompilerResult compile(const std::string& source) override;
00077
00091   CompilerResult compile(std::istream& input) override;
00092 };
00093
00094 }  // namespace nsbaci::compiler
00095
00096 #endif  // NSBACI_COMPILER_NSBACI_COMPILER_H
```

7.37 compilerService.cpp File Reference

Implementation of the CompilerService class for nsbaci.

Include dependency graph for compilerService.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.

7.37.1 Detailed Description

Implementation of the CompilerService class for nsbaci.

Author

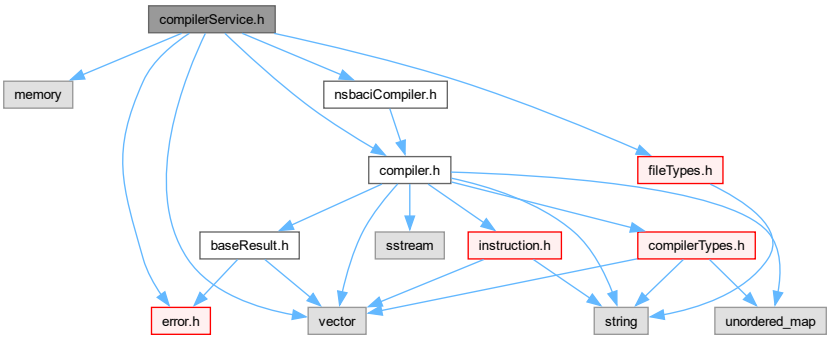
Nicolás Serrano García

Copyright

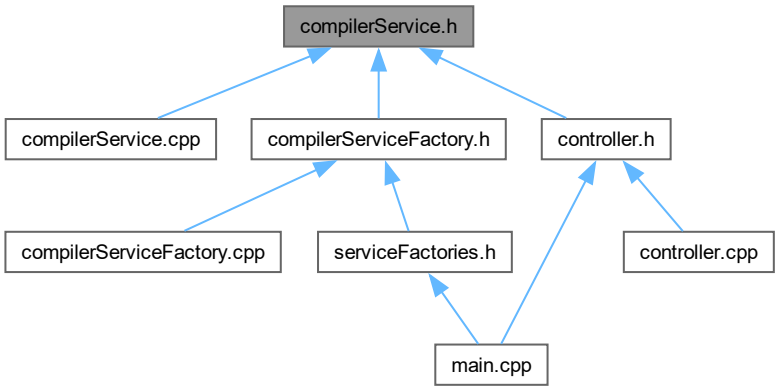
7.38 compilerService.h File Reference

CompilerService class declaration for nsbaci.

Include dependency graph for compilerService.h:



This graph shows which files directly or indirectly include this file:



Classes

- class nsbaci::services::CompilerService

  service for compiling nsbaci (or maybe other stuff in the future) source code.

Namespaces

- namespace nsbaci::services

  Services namespace containing all backend service implementations.
- namespace nsbaci

  Root namespace for the nsbaci application.

7.38.1 Detailed Description

CompilerService class declaration for nsbaci.

This module defines the CompilerService class that provides a high-level interface for compiling nsbaci source code into executable p-code instructions. The service wraps a Compiler implementation and manages the compiled program state, making it available for the runtime system.

The CompilerService acts as a bridge between the Controller and the actual compiler implementation. This means that the compilerService holds a specific compiler, and bubbles the interface of the compiler itself

Author

Nicolás Serrano García

Copyright

7.39 compilerService.h

Go to the documentation of this file.

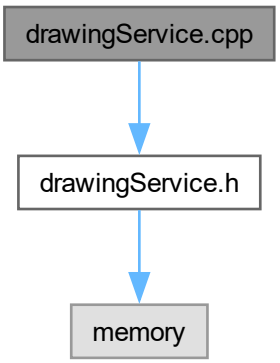
```
00001
00018
00019 #ifndef NSBACI_COMPILERSERVICE_H
00020 #define NSBACI_COMPILERSERVICE_H
00021
00022 #include <memory>
00023 #include <vector>
00024
00025 #include "compiler.h"
00026 #include "error.h"
00027 #include "fileTypes.h"
00028 #include "nsbaciCompiler.h"
00029
00034 namespace nsbaci::services {
00035
00061 class CompilerService {
00062  public:
00073   CompilerService(std::unique_ptr<nsbaci::compiler::Compiler> c =
00074                       std::make_unique<nsbaci::compiler::NsbaciCompiler>());
00075
00079   ~CompilerService() = default;
00080
00081   CompilerService(const CompilerService&) = delete;
00082   CompilerService& operator=(const CompilerService&) = delete;
00083
```

```
00084   CompilerService(CompilerService&&) = default;
00085   CompilerService& operator=(CompilerService&&) = default;
00086
00098   nsbaci::compiler::CompilerResult compile(nsbaci::types::Text raw);
00099
00109   bool hasProgramReady() const;
00110
00120   nsbaci::compiler::InstructionStream takeInstructions();
00121
00131   nsbaci::types::SymbolTable takeSymbols();
00132
00133  private:
00134   std::unique_ptr<nsbaci::compiler::Compiler>
00135       compiler;
00136   nsbaci::compiler::InstructionStream
00137       lastCompiledInstructions;
00138   nsbaci::types::SymbolTable
00139       lastCompiledSymbols;
00140   bool programReady = false;
00141 };
00142
00143 }  // namespace nsbaci::services
00144
00145 #endif  // NSBACI_COMPILERSERVICE_H
```

7.40 drawingService.cpp File Reference

DrawingService class implementation for nsbaci.

Include dependency graph for drawingService.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.

7.40.1 Detailed Description

DrawingService class implementation for nsbaci.

Author

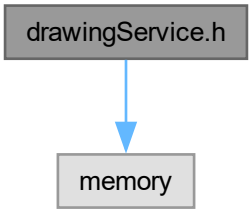
Nicolás Serrano García

Copyright

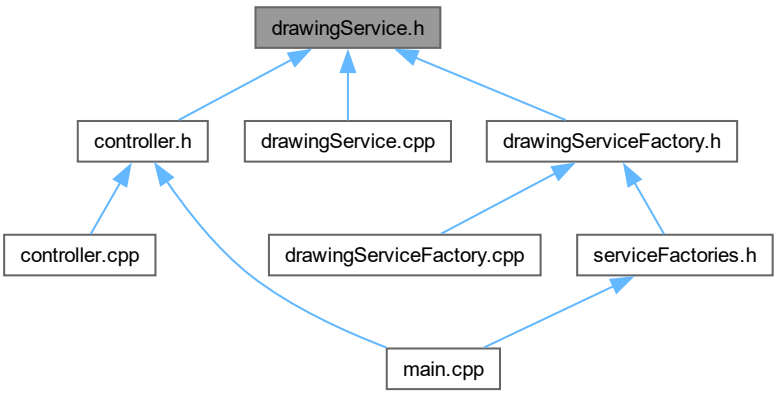
7.41 drawingService.h File Reference

DrawingService class declaration for nsbaci.

Include dependency graph for drawingService.h:



This graph shows which files directly or indirectly include this file:



Classes

- class nsbaci::services::DrawingService

  Adapter service for graphical output backends.

Namespaces

- namespace nsbaci::services

  Services namespace containing all backend service implementations.
- namespace nsbaci

  Root namespace for the nsbaci application.

7.41.1 Detailed Description

DrawingService class declaration for nsbaci.

This service acts as an adapter for graphical output backends. It communicates with the RuntimeService via Qt signals/slots to handle drawing operations triggered by program execution.

Author

Nicolás Serrano García

Copyright

7.42 drawingService.h

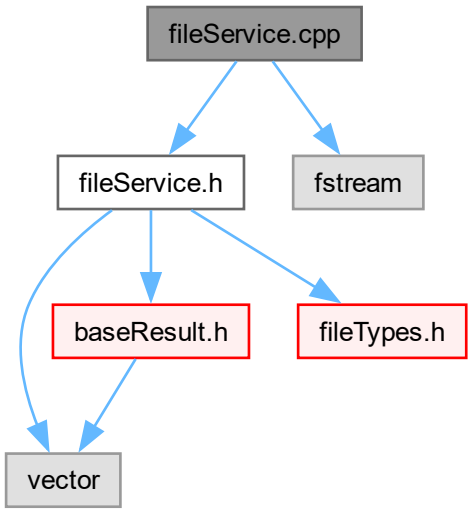
Go to the documentation of this file.

```
00001
00013
00014 #ifndef NSBACI_SERVICES_DRAWINGSERVICE_H
00015 #define NSBACI_SERVICES_DRAWINGSERVICE_H
00016
00017 #include <memory>
00018
00023 namespace nsbaci::services {
00024
00029 class DrawingService {
00030  public:
00031   DrawingService() = default;
00032   ~DrawingService() = default;
00033   // TODO: Add signals for notifying backends of drawing operations
00034   // Example: void drawFigure(std::shared_ptr<Figure> figure);
00035
00036   DrawingService(const DrawingService&) = delete;
00037   DrawingService& operator=(const DrawingService&) = delete;
00038
00039   DrawingService(DrawingService&&) = default;
00040   DrawingService& operator=(DrawingService&&) = default;
00041
00042  private:
00043   // TODO: Add backend reference/pointer when backends are implemented
00044 };
00045
00046 }  // namespace nsbaci::services
00047
00048 #endif  // NSBACI_SERVICES_DRAWINGSERVICE_H
```

7.43 fileService.cpp File Reference

Implementation of the FileService class for nsbaci.

Include dependency graph for fileService.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.

7.43.1 Detailed Description

Implementation of the FileService class for nsbaci.

This file contains the implementation of file save and load operations for NsBaci source files (.nsb). It provides comprehensive validation and error handling for all file system operations.

Author

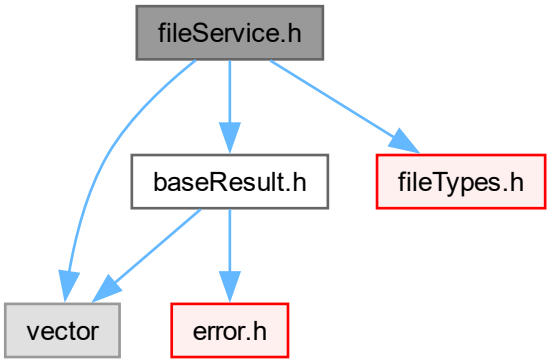
Nicolás Serrano García

Copyright

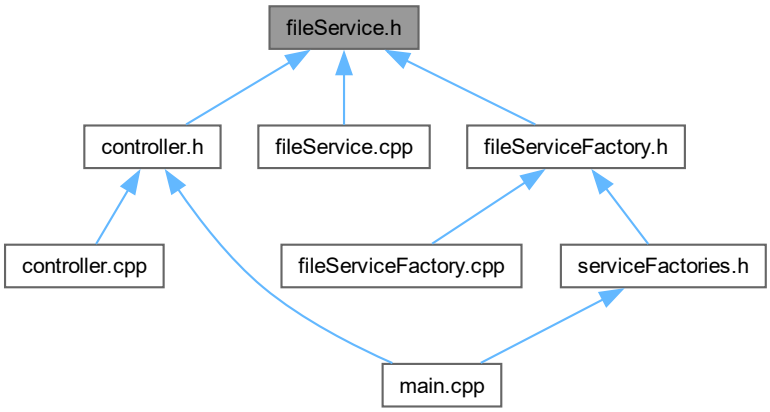
7.44 fileService.h File Reference

FileService class declaration for nsbaci.

Include dependency graph for fileService.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct FileResult

  Base result type for file operations.
- struct saveResult

  Result type for file save operations.
- struct LoadResult

  Result type for file load operations.
- class nsbaci::services::FileService

  Service for handling file system operations on BACI source files.

Namespaces

- namespace nsbaci::services

  Services namespace containing all backend service implementations.
- namespace nsbaci

  Root namespace for the nsbaci application.

7.44.1 Detailed Description

FileService class declaration for nsbaci.

This module defines the FileService class and its associated result types for handling file system operations. The FileService provides a clean abstraction over file I/O operations, specifically tailored for Nsbaci source files (.nsb extension).

The service handles:

- Saving source code to .nsb files with validation
- Loading source code from .nsb files with error checking
- Path validation

Author

Nicolás Serrano García

Copyright

7.45 fileService.h

Go to the documentation of this file.

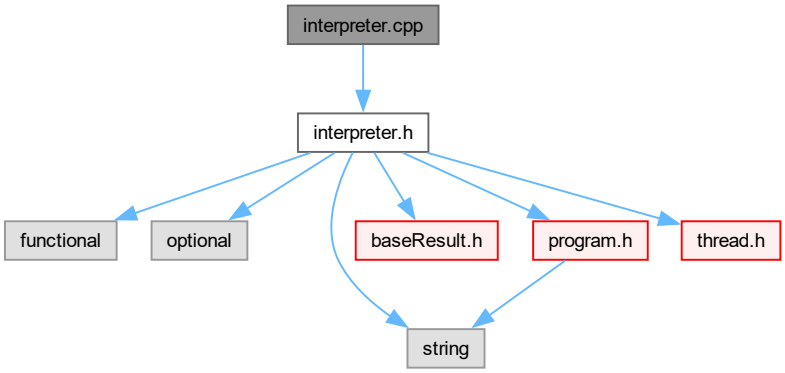
```
00001
00019
00020 #ifndef NSBACI_FILESERVICE_H
00021 #define NSBACI_FILESERVICE_H
00022
00023 #include <vector>
00024
00025 #include "baseResult.h"
00026 #include "fileTypes.h"
00027
00037 struct FileResult : nsbaci::BaseResult {
00041   FileResult() : BaseResult() {}
00042
00047   explicit FileResult(std::vector<nsbaci::Error> errs)
00048       : BaseResult(std::move(errs)) {}
00049
00054   explicit FileResult(nsbaci::Error error) : BaseResult(std::move(error)) {}
00055
00056   FileResult(FileResult&&) noexcept = default;
00057   FileResult& operator=(FileResult&&) noexcept = default;
00058
00059   FileResult(const FileResult&) = default;
00060   FileResult& operator=(const FileResult&) = default;
00061 };
00062
00070 struct saveResult : FileResult {
00074   saveResult() : FileResult() {}
```

```
00075
00080   explicit saveResult(std::vector<nsbaci::Error> errs)
00081       : FileResult(std::move(errs)) {}
00082
00087   explicit saveResult(nsbaci::Error error) : FileResult(std::move(error)) {}
00088
00089   saveResult(saveResult&&) noexcept = default;
00090   saveResult& operator=(saveResult&&) noexcept = default;
00091   saveResult(const saveResult&) = default;
00092   saveResult& operator=(const saveResult&) = default;
00093 };
00094
00103 struct LoadResult : FileResult {
00107   LoadResult() : FileResult() {}
00108
00114   LoadResult(nsbaci::types::Text conts, nsbaci::types::File name)
00115       : FileResult(), contents(std::move(conts)), fileName(std::move(name)) {}
00116
00121   explicit LoadResult(std::vector<nsbaci::Error> errs)
00122       : FileResult(std::move(errs)) {}
00123
00128   explicit LoadResult(nsbaci::Error error) : FileResult(std::move(error)) {}
00129
00130   LoadResult(LoadResult&&) noexcept = default;
00131   LoadResult& operator=(LoadResult&&) noexcept = default;
00132   LoadResult(const LoadResult&) = default;
00133   LoadResult& operator=(const LoadResult&) = default;
00134
00135   nsbaci::types::Text contents;
00136   nsbaci::types::File fileName;
00137 };
00138
00146 namespace nsbaci::services {
00147
00181 class FileService {
00182  public:
00196   saveResult save(nsbaci::types::Text contents, nsbaci::types::File file);
00197
00208   LoadResult load(nsbaci::types::File file);
00209
00213   FileService() = default;
00214
00215   FileService(const FileService&) = delete;
00216   FileService& operator=(const FileService&) = delete;
00217
00218   FileService(FileService&&) = default;
00219   FileService& operator=(FileService&&) = default;
00220
00221   ~FileService() = default;
00222 };
00223
00224 }  // namespace nsbaci::services
00225
00226 #endif  // NSBACI_FILESERVICE_H
```

7.46 interpreter.cpp File Reference

Interpreter class implementation for nsbaci runtime service.

Include dependency graph for interpreter.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.
- namespace nsbaci::services::runtime

  Runtime services namespace for nsbaci.

7.46.1 Detailed Description

Interpreter class implementation for nsbaci runtime service.

Author

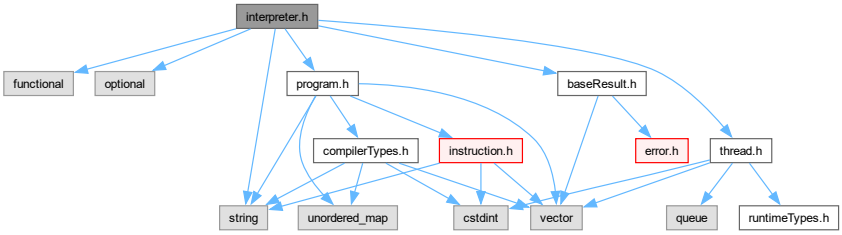
Nicolás Serrano García

Copyright

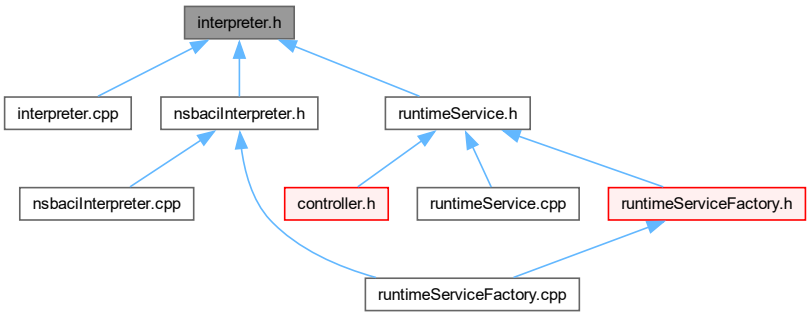
7.47 interpreter.h File Reference

Interpreter class declaration for nsbaci runtime service.

Include dependency graph for interpreter.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct InterpreterResult
- class nsbaci::services::runtime::Interpreter

  Executes instructions for threads within a program context.

Namespaces

- namespace nsbaci::services::runtime

  Runtime services namespace for nsbaci.
- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.

Typedefs

- using **nsbaci::services::runtime::OutputCallback** = std::function<void(const std::string&)>

  Callback type for output operations.
- using **nsbaci::services::runtime::InputRequestCallback** = std::function<void(const std::string&)>

  Callback type for input requests.

7.47.1 Detailed Description

Interpreter class declaration for nsbaci runtime service.

This module defines the Interpreter class responsible for executing instructions within a program context.

Author

Nicolás Serrano García

Copyright

7.48 interpreter.h

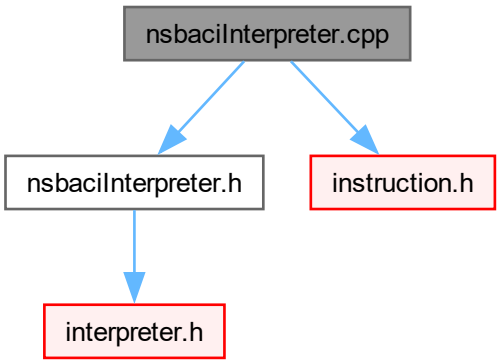
Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_SERVICES_RUNTIME_INTERPRETER_H
00014 #define NSBACI_SERVICES_RUNTIME_INTERPRETER_H
00015
00016 #include <functional>
00017 #include <optional>
00018 #include <string>
00019
00020 #include "baseResult.h"
00021 #include "program.h"
00022 #include "thread.h"
00023
00024 struct InterpreterResult : nsbaci::BaseResult {
00025   InterpreterResult() : BaseResult() {}
00026   explicit InterpreterResult(std::vector<nsbaci::Error> errs)
00027       : BaseResult(std::move(errs)) {}
00028   explicit InterpreterResult(nsbaci::Error error)
00029       : BaseResult(std::move(error)) {}
00030
00031   InterpreterResult(InterpreterResult&&) noexcept = default;
00032   InterpreterResult& operator=(InterpreterResult&&) noexcept = default;
00033
00034   InterpreterResult(const InterpreterResult&) = default;
00035   InterpreterResult& operator=(const InterpreterResult&) = default;
00036
00037   bool needsInput = false;
00038   std::string inputPrompt;
00039   std::string output;
00040 };
00041
00046 namespace nsbaci::services::runtime {
00047
00049 using OutputCallback = std::function<void(const std::string&)>;
00050
00052 using InputRequestCallback = std::function<void(const std::string&)>;
00053
00061 class Interpreter {
00062  public:
00063   Interpreter() = default;
00064   virtual ~Interpreter() = default;
00065
00072   virtual InterpreterResult executeInstruction(Thread& t, Program& program) = 0;
00073
00078   virtual void provideInput(const std::string& input) = 0;
00079
00084   virtual bool isWaitingForInput() const = 0;
00085
00090   virtual void setOutputCallback(OutputCallback callback) = 0;
00091 };
00092
00093 }  // namespace nsbaci::services::runtime
00094
00095 #endif  // NSBACI_SERVICES_RUNTIME_INTERPRETER_H
```

7.49 nsbaciInterpreter.cpp File Reference

NsbaciInterpreter class implementation for nsbaci runtime service.

Include dependency graph for nsbaciInterpreter.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.
- namespace nsbaci::services::runtime

  Runtime services namespace for nsbaci.

7.49.1 Detailed Description

NsbaciInterpreter class implementation for nsbaci runtime service.

Author

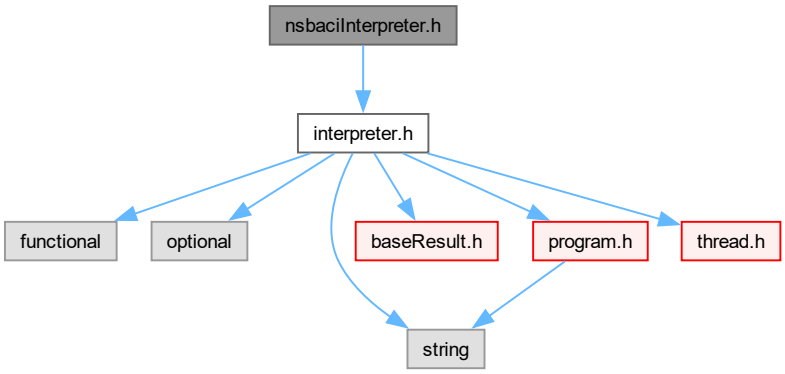
Nicolás Serrano García

Copyright

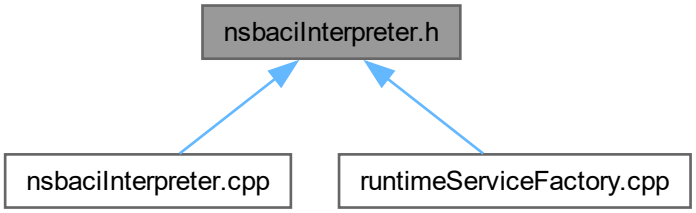
7.50 nsbaciInterpreter.h File Reference

NsbaciInterpreter class declaration for nsbaci runtime service.

Include dependency graph for nsbaciInterpreter.h:



This graph shows which files directly or indirectly include this file:



Classes

- class nsbaci::services::runtime::NsbaciInterpreter

  BACI-specific implementation of the Interpreter.

Namespaces

- namespace nsbaci::services::runtime

  Runtime services namespace for nsbaci.
- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.

7.50.1 Detailed Description

NsbaciInterpreter class declaration for nsbaci runtime service.

This module provides the concrete BACI-specific implementation of the Interpreter interface for executing BACI instructions.

Author

Nicolás Serrano García

Copyright

7.51 nsbaciInterpreter.h

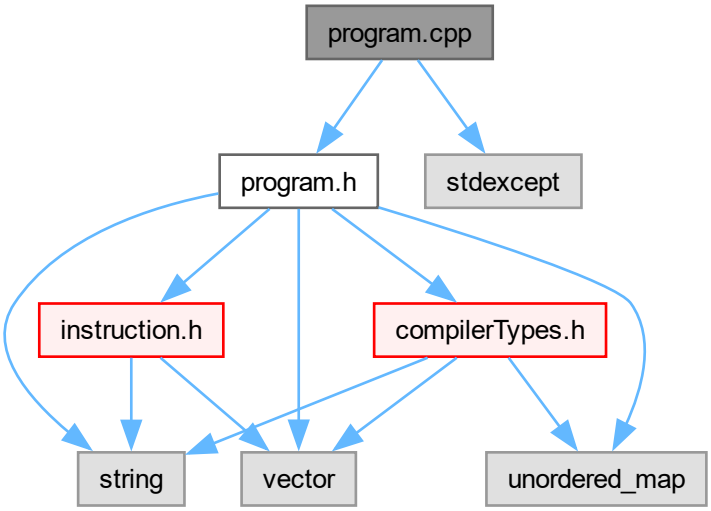
Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_SERVICES_RUNTIME_NSBACI_INTERPRETER_H
00014 #define NSBACI_SERVICES_RUNTIME_NSBACI_INTERPRETER_H
00015
00016 #include "interpreter.h"
00017
00022 namespace nsbaci::services::runtime {
00023
00031 class NsbaciInterpreter final : public Interpreter {
00032  public:
00033   NsbaciInterpreter() = default;
00034   ~NsbaciInterpreter() override = default;
00035
00043   InterpreterResult executeInstruction(Thread& t, Program& program) override;
00044
00045   void provideInput(const std::string& input) override;
00046   bool isWaitingForInput() const override;
00047   void setOutputCallback(OutputCallback callback) override;
00048
00049  private:
00050   OutputCallback outputCallback;
00051   bool waitingForInput = false;
00052   std::string pendingInput;
00053   bool hasInput = false;
00054 };
00055
00056 }  // namespace nsbaci::services::runtime
00057
00058 #endif  // NSBACI_SERVICES_RUNTIME_NSBACI_INTERPRETER_H
```

7.52 program.cpp File Reference

Program class implementation for nsbaci runtime service.

Include dependency graph for program.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.
- namespace nsbaci::services::runtime

  Runtime services namespace for nsbaci.

7.52.1 Detailed Description

Program class implementation for nsbaci runtime service.

Author

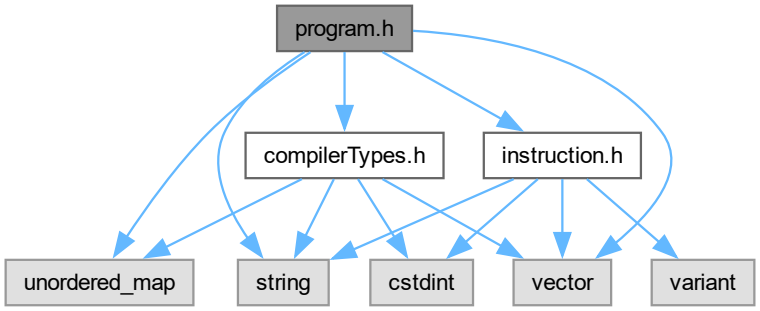
Nicolás Serrano García

Copyright

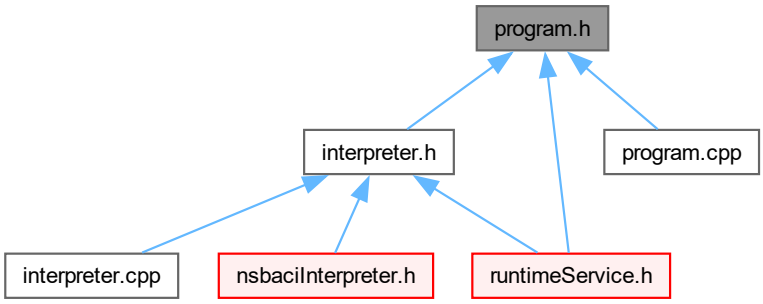
7.53 program.h File Reference

Program class declaration for nsbaci runtime service.

Include dependency graph for program.h:



This graph shows which files directly or indirectly include this file:



Classes

- class nsbaci::services::runtime::Program

  Represents a compiled program ready for execution.

Namespaces

- namespace nsbaci::types

  Type definitions namespace for nsbaci (runtime-specific).
- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services::runtime

  Runtime services namespace for nsbaci.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.

Typedefs

- using **nsbaci::types::StackValue** = int32_t

  Stack value type (can hold int or address).
- using **nsbaci::types::Stack** = std::vector<StackValue>

  Runtime stack.
- using **nsbaci::types::Memory** = std::vector<int32_t>

  Memory block for runtime data.

7.53.1 Detailed Description

Program class declaration for nsbaci runtime service.

This module defines the Program class which holds the compiled program's instruction vector, memory tables, and other runtime data.

Author

Nicolás Serrano García

Copyright

7.54 program.h

Go to the documentation of this file.

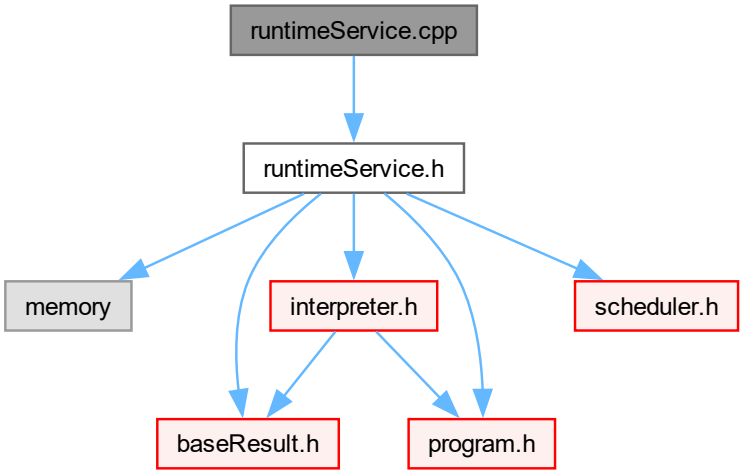
```
00001
00012
00013 #ifndef NSBACI_SERVICES_RUNTIME_PROGRAM_H
00014 #define NSBACI_SERVICES_RUNTIME_PROGRAM_H
00015
00016 #include <string>
00017 #include <unordered_map>
00018 #include <vector>
00019
00020 #include "compilerTypes.h"
00021 #include "instruction.h"
00022
00027 namespace nsbaci::types {
00028
00030 using StackValue = int32_t;
00031
00033 using Stack = std::vector<StackValue>;
00034
00036 using Memory = std::vector<int32_t>;
00037
00038 }  // namespace nsbaci::types
00039
00044 namespace nsbaci::services::runtime {
00045
00053 class Program {
00054  public:
00055   Program() = default;
00056   explicit Program(nsbaci::compiler::InstructionStream i);
00057   Program(nsbaci::compiler::InstructionStream i, nsbaci::types::SymbolTable s);
00058   ~Program() = default;
00059
00060   Program(const Program&) = delete;
00061   Program& operator=(const Program&) = delete;
00062
00063   Program(Program&&) = default;
00064   Program& operator=(Program&&) = default;
00065
00071   const nsbaci::compiler::Instruction& getInstruction(uint32_t addr) const;
```

```
00072
00077   size_t instructionCount() const;
00078
00083   nsbaci::types::Memory& memory();
00084   const nsbaci::types::Memory& memory() const;
00085
00090   const nsbaci::types::SymbolTable& symbols() const;
00091
00096   void addSymbol(nsbaci::types::SymbolInfo info);
00097
00103   int32_t readMemory(nsbaci::types::MemoryAddr addr) const;
00104
00110   void writeMemory(nsbaci::types::MemoryAddr addr, int32_t value);
00111
00112  private:
00113   // Instruction stream - read-only after construction
00114   nsbaci::compiler::InstructionStream instructions;
00115   // Global symbol table
00116   nsbaci::types::SymbolTable symbolTable;
00117   // Global memory
00118   nsbaci::types::Memory globalMemory;
00119 };
00120
00121 }  // namespace nsbaci::services::runtime
00122
00123 #endif  // NSBACI_SERVICES_RUNTIME_PROGRAM_H
```

7.55 runtimeService.cpp File Reference

Implementation unit for the RuntimeService.

Include dependency graph for runtimeService.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.

7.55.1 Detailed Description

Implementation unit for the RuntimeService.

Author

Nicolás Serrano García

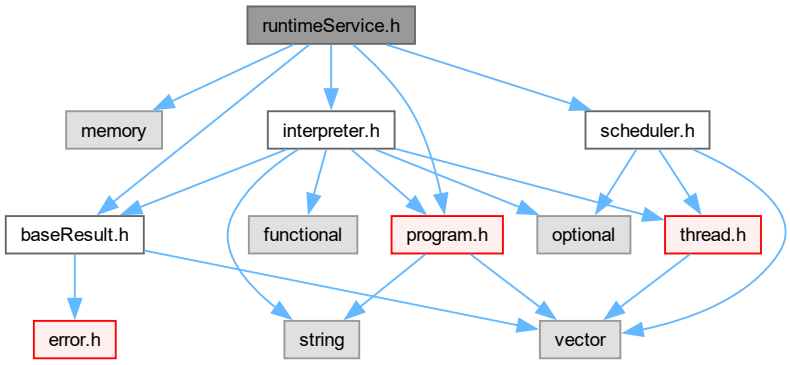
Copyright

Copyright (c) 2025 Nicolás Serrano García. Licensed under the MIT License.

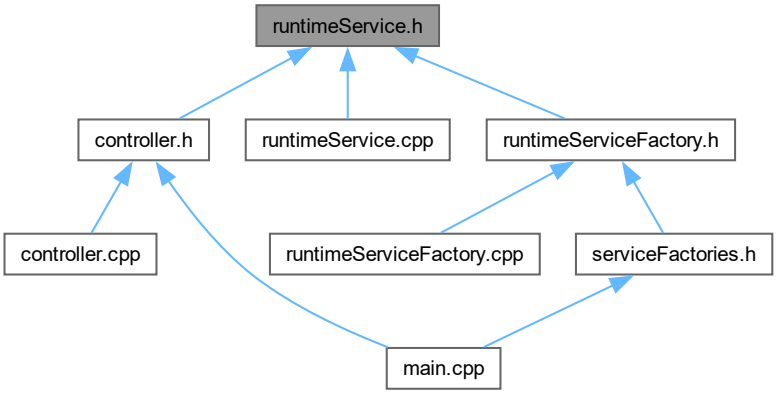
7.56 runtimeService.h File Reference

RuntimeService class declaration for nsbaci.

Include dependency graph for runtimeService.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct nsbaci::services::RuntimeResult

  Result of a runtime operation (step, run, etc.).
- class nsbaci::services::RuntimeService

  Service that manages program execution.

Namespaces

- namespace nsbaci::services

  Services namespace containing all backend service implementations.
- namespace nsbaci

  Root namespace for the nsbaci application.

Enumerations

- enum class nsbaci::services::RuntimeState { nsbaci::services::Idle , nsbaci::services::Running , nsbaci::services::Paused , nsbaci::services::Halted }

  Possible states of the runtime service.

7.56.1 Detailed Description

RuntimeService class declaration for nsbaci.

This module defines the RuntimeService class which serves as the main interface for executing compiled nsbaci programs. It controls the interpreter and scheduler components to provide program execution with support for stepping, continuous running, and status info

Author

Nicolás Serrano García

Copyright

7.57 runtimeService.h

Go to the documentation of this file.

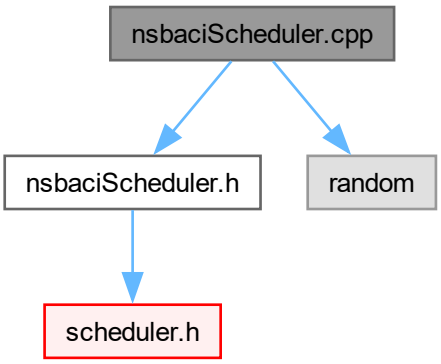
```
00001
00014
00015 #ifndef NSBACI_RUNTIMESERVICE_H
00016 #define NSBACI_RUNTIMESERVICE_H
00017
00018 #include <memory>
00019
00020 #include "baseResult.h"
00021 #include "interpreter.h"
00022 #include "program.h"
00023 #include "scheduler.h"
00024
00029 namespace nsbaci::services {
00030
00035 struct RuntimeResult : nsbaci::BaseResult {
```

```
00039   RuntimeResult() : BaseResult() {}
00040
00045   explicit RuntimeResult(std::vector<nsbaci::Error> errs)
00046       : BaseResult(std::move(errs)) {}
00047
00052   explicit RuntimeResult(nsbaci::Error error) : BaseResult(std::move(error)) {}
00053
00054   RuntimeResult(RuntimeResult&&) noexcept = default;
00055   RuntimeResult& operator=(RuntimeResult&&) noexcept = default;
00056   RuntimeResult(const RuntimeResult&) = default;
00057   RuntimeResult& operator=(const RuntimeResult&) = default;
00058
00059   bool halted = false;
00060   bool needsInput = false;
00061   std::string inputPrompt;
00062   std::string output;
00063 };
00064
00075 enum class RuntimeState {
00076   Idle,
00077   Running,
00078   Paused,
00079   Halted
00080 };
00081
00104 class RuntimeService {
00105  public:
00113   RuntimeService() = default;
00114
00120   RuntimeService(std::unique_ptr<runtime::Interpreter> i,
00121                  std::unique_ptr<runtime::Scheduler> s);
00122
00126   ~RuntimeService() = default;
00127
00128   RuntimeService(const RuntimeService&) = delete;
00129   RuntimeService& operator=(const RuntimeService&) = delete;
00130
00131   RuntimeService(RuntimeService&&) = default;
00132   RuntimeService& operator=(RuntimeService&&) = default;
00133
00144   void loadProgram(runtime::Program&& p);
00145
00152   void reset();
00153
00162   RuntimeResult step();
00163
00172   RuntimeResult stepThread(nsbaci::types::ThreadID threadId);
00173
00186   RuntimeResult run(size_t maxSteps = 0);
00187
00193   void pause();
00194
00199   RuntimeState getState() const;
00200
00205   bool isHalted() const;
00206
00211   size_t threadCount() const;
00212
00217   const std::vector<runtime::Thread>& getThreads() const;
00218
00224   const runtime::Program& getProgram() const;
00225
00234   void provideInput(const std::string& input);
00235
00240   bool isWaitingForInput() const;
00241
00250   void setOutputCallback(runtime::OutputCallback callback);
00251
00252  private:
00253   runtime::Program
00254       program;
00255   std::unique_ptr<runtime::Interpreter>
00256       interpreter;
00257   std::unique_ptr<runtime::Scheduler>
00258       scheduler;
00259   RuntimeState state = RuntimeState::Idle;
00260 };
00261
00262 }  // namespace nsbaci::services
00263
00264 #endif  // NSBACI_RUNTIMESERVICE_H
```

7.58 nsbaciScheduler.cpp File Reference

NsbaciScheduler class implementation for nsbaci runtime service.

Include dependency graph for nsbaciScheduler.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.
- namespace nsbaci::services::runtime

  Runtime services namespace for nsbaci.

7.58.1 Detailed Description

NsbaciScheduler class implementation for nsbaci runtime service.

Author

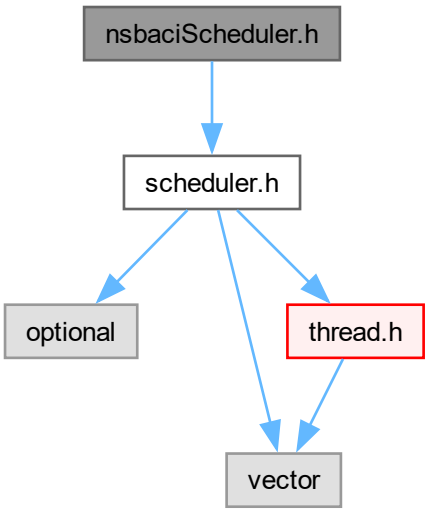
Nicolás Serrano García

Copyright

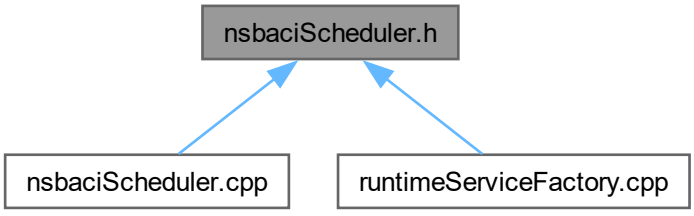
7.59 nsbaciScheduler.h File Reference

NsbaciScheduler class declaration for nsbaci runtime service.

Include dependency graph for nsbaciScheduler.h:



This graph shows which files directly or indirectly include this file:



Classes

- class nsbaci::services::runtime::NsbaciScheduler

  BACI-specific implementation of the Scheduler.

Namespaces

- namespace nsbaci::services::runtime

  Runtime services namespace for nsbaci.
- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.

7.59.1 Detailed Description

NsbaciScheduler class declaration for nsbaci runtime service.

This module provides the concrete BACI-specific implementation of the Scheduler interface for managing thread scheduling in the nsbaci runtime.

Author

Nicolás Serrano García

Copyright

7.60 nsbaciScheduler.h

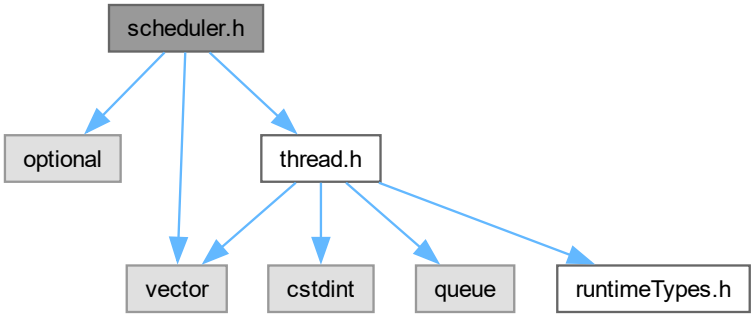
Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_SERVICES_RUNTIME_NSBACI_SCHEDULER_H
00014 #define NSBACI_SERVICES_RUNTIME_NSBACI_SCHEDULER_H
00015
00016 #include "scheduler.h"
00017
00022 namespace nsbaci::services::runtime {
00023
00033 class NsbaciScheduler final : public Scheduler {
00034  public:
00035   NsbaciScheduler() = default;
00036   ~NsbaciScheduler() override = default;
00037
00038   Thread* pickNext() override;
00039   void addThread(Thread thread) override;
00040   void blockCurrent() override;
00041   void unblock(nsbaci::types::ThreadID threadId) override;
00042   void yield() override;
00043   void terminateCurrent() override;
00044   bool hasThreads() const override;
00045   Thread* current() override;
00046   void clear() override;
00047   void unblockIO() override;
00048   const std::vector<Thread>& getThreads() const override;
00049
00050  private:
00056   std::optional<size_t> findThreadIndex(nsbaci::types::ThreadID threadId) const;
00057 };
00058
00059 }  // namespace nsbaci::services::runtime
00060
00061 #endif  // NSBACI_SERVICES_RUNTIME_NSBACI_SCHEDULER_H
```

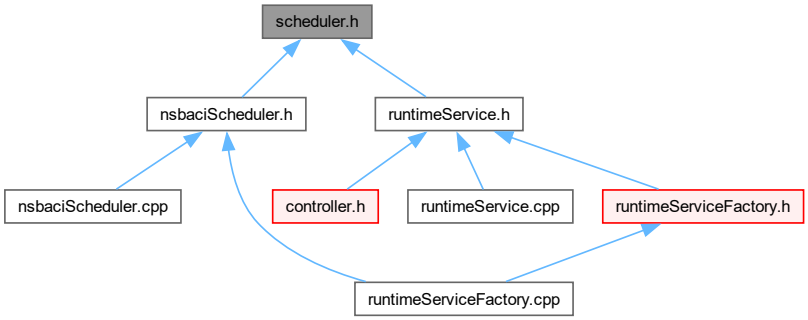
7.61 scheduler.h File Reference

Scheduler class declaration for nsbaci runtime service.

Include dependency graph for scheduler.h:



This graph shows which files directly or indirectly include this file:



Classes

- class nsbaci::services::runtime::Scheduler

  Manages thread scheduling and state transitions.

Namespaces

- namespace nsbaci::services::runtime

  Runtime services namespace for nsbaci.
- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.

7.61.1 Detailed Description

Scheduler class declaration for nsbaci runtime service.

This module defines the thread scheduler responsible for managing thread execution order and state transitions.

Author

Nicolás Serrano García

Copyright

7.62 scheduler.h

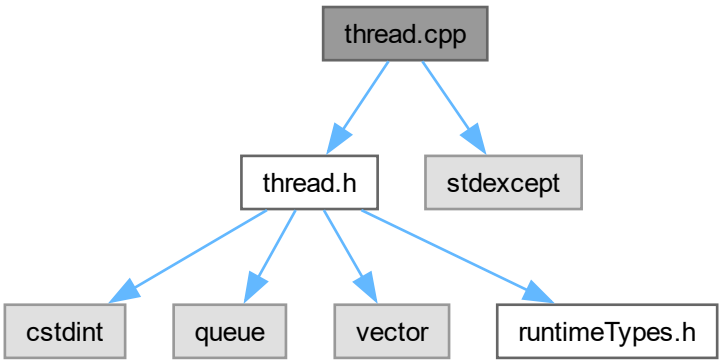
Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_SERVICES_RUNTIME_SCHEDULER_H
00014 #define NSBACI_SERVICES_RUNTIME_SCHEDULER_H
00015
00016 #include <optional>
00017 #include <vector>
00018
00019 #include "thread.h"
00020
00025 namespace nsbaci::services::runtime {
00026
00035 class Scheduler {
00036  public:
00037   Scheduler() = default;
00038   virtual ~Scheduler() = default;
00039
00044   virtual Thread* pickNext() = 0;
00045
00050   virtual void addThread(Thread thread) = 0;
00051
00055   virtual void blockCurrent() = 0;
00056
00061   virtual void unblock(nsbaci::types::ThreadID threadId) = 0;
00062
00066   virtual void yield() = 0;
00067
00071   virtual void terminateCurrent() = 0;
00072
00077   virtual bool hasThreads() const = 0;
00078
00083   virtual Thread* current() = 0;
00084
00088   virtual void clear() = 0;
00089
00094   virtual void unblockIO() = 0;
00095
00100   virtual const std::vector<Thread>& getThreads() const = 0;
00101
00102  protected:
00103   std::vector<Thread> threads;
00104   std::vector<size_t> readyQueue;
00105   std::vector<size_t> blockedQueue;
00106   std::vector<size_t> ioQueue;
00107   std::optional<size_t> runningIndex;
00108 };
00109
00110 }  // namespace nsbaci::services::runtime
00111
00112 #endif  // NSBACI_SERVICES_RUNTIME_SCHEDULER_H
```

7.63 thread.cpp File Reference

Thread class implementation for nsbaci runtime service.

Include dependency graph for thread.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.
- namespace nsbaci::services::runtime

  Runtime services namespace for nsbaci.

7.63.1 Detailed Description

Thread class implementation for nsbaci runtime service.

Author

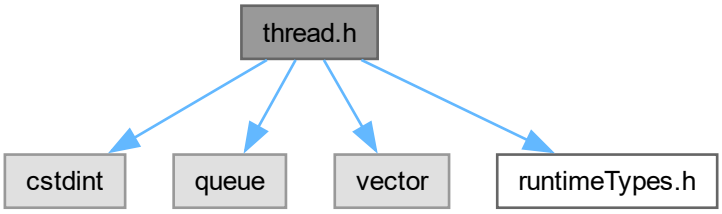
Nicolás Serrano García

Copyright

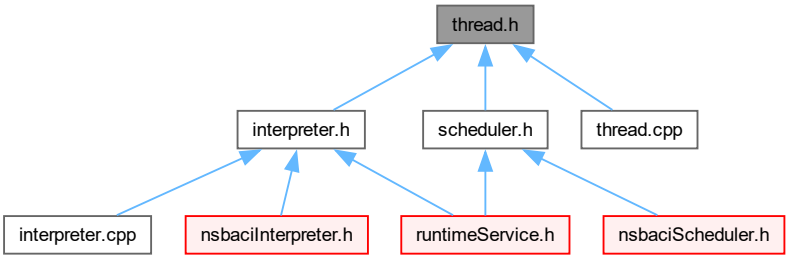
7.64 thread.h File Reference

Thread class declaration for nsbaci runtime service.

Include dependency graph for thread.h:



This graph shows which files directly or indirectly include this file:



Classes

- class nsbaci::services::runtime::Thread

  Represents a thread in the runtime service.

Namespaces

- namespace nsbaci::services::runtime

  Runtime services namespace for nsbaci.
- namespace nsbaci

  Root namespace for the nsbaci application.
- namespace nsbaci::services

  Services namespace containing all backend service implementations.
- namespace nsbaci::types

  Type definitions namespace for nsbaci (runtime-specific).

Typedefs

- using **nsbaci::types::ThreadQueue** = std::queue<nsbaci::services::runtime::Thread>

  Queue of threads for scheduler operations.

7.64.1 Detailed Description

Thread class declaration for nsbaci runtime service.

Author

Nicolás Serrano García

Copyright

7.65 thread.h

Go to the documentation of this file.

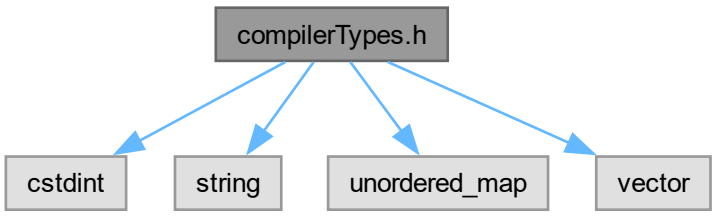
```
00001
00009
00010 #ifndef NSBACI_SERVICES_RUNTIME_THREAD_H
00011 #define NSBACI_SERVICES_RUNTIME_THREAD_H
00012
00013 #include <cstdint>
00014 #include <queue>
00015 #include <vector>
00016
00017 #include "runtimeTypes.h"
00018
00023 namespace nsbaci::services::runtime {
00024
00031 class Thread {
00032  public:
00033   Thread()
00034       : id(nextThreadId++),
00035         state(nsbaci::types::ThreadState::Ready),
00036         priority(0),
00037         pc(0),
00038         bp(0),
00039         sp(0) {}
00040   ~Thread() = default;
00041
00046   nsbaci::types::ThreadID getId() const;
00047
00052   nsbaci::types::ThreadState getState() const;
00053
00058   void setState(nsbaci::types::ThreadState newState);
00059
00064   nsbaci::types::Priority getPriority() const;
00065
00070   void setPriority(nsbaci::types::Priority newPriority);
00071
00072   // ============== Stack Operations ==============
00073
00077   void push(int32_t value);
00078
00082   int32_t pop();
00083
00087   int32_t top() const;
00088
00089   // ============== Program Counter ==============
00090
00094   uint32_t getPC() const { return pc; }
00095
00099   void setPC(uint32_t addr) { pc = addr; }
00100
00104   void advancePC() { ++pc; }
```

```
00105
00106   // ============== Base/Stack Pointers ==============
00107
00108   uint32_t getBP() const { return bp; }
00109   void setBP(uint32_t addr) { bp = addr; }
00110
00111   uint32_t getSP() const { return sp; }
00112   void setSP(uint32_t addr) { sp = addr; }
00113
00114  private:
00115   nsbaci::types::ThreadID id;
00116   nsbaci::types::ThreadState state;
00117   nsbaci::types::Priority priority;
00118
00119   // Program counter - index into instruction stream
00120   uint32_t pc;
00121   // Base pointer - for stack frames
00122   uint32_t bp;
00123   // Stack pointer
00124   uint32_t sp;
00125
00126   // Thread-local stack
00127   std::vector<int32_t> stack;
00128
00129   static nsbaci::types::ThreadID nextThreadId;
00130
00131   // friend the scheduler
00132 };
00133
00134 }  // namespace nsbaci::services::runtime
00135
00140 namespace nsbaci::types {
00141
00143 using ThreadQueue = std::queue<nsbaci::services::runtime::Thread>;
00144
00145 }  // namespace nsbaci::types
00146
00147 #endif  // NSBACI_SERVICES_RUNTIME_THREAD_H
```

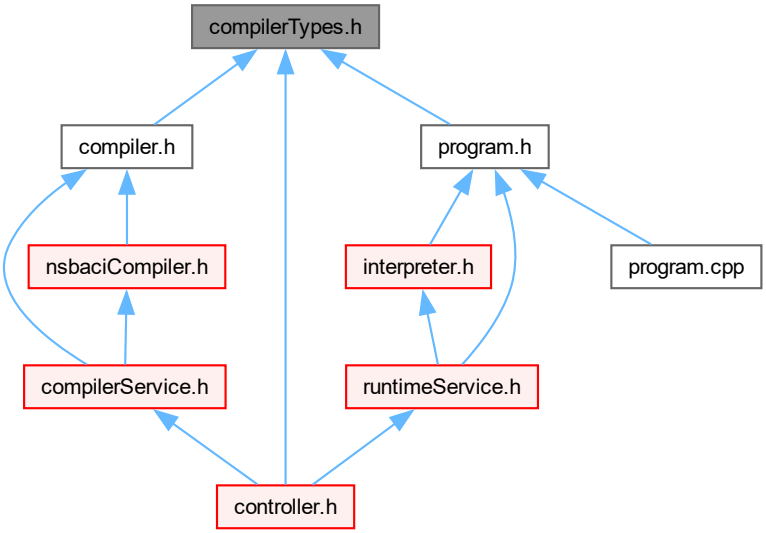
7.66 compilerTypes.h File Reference

Type definitions for compiler-related operations.

Include dependency graph for compilerTypes.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct nsbaci::types::SymbolInfo

  Information about a variable/symbol.

Namespaces

- namespace nsbaci::types

  Type definitions namespace for nsbaci (runtime-specific).
- namespace nsbaci

  Root namespace for the nsbaci application.

Typedefs

- using **nsbaci::types::VarName** = std::string

  Type alias for variable names.
- using **nsbaci::types::MemoryAddr** = uint32_t

  Type alias for memory addresses.
- using **nsbaci::types::SymbolTable** = std::unordered_map<VarName, SymbolInfo>

  Lookup table mapping variable names to their symbol info.

7.66.1 Detailed Description

Type definitions for compiler-related operations.

This header provides type aliases used by the CompilerService and other components that work with compilation.

Author

Nicolás Serrano García

Copyright

7.67 compilerTypes.h

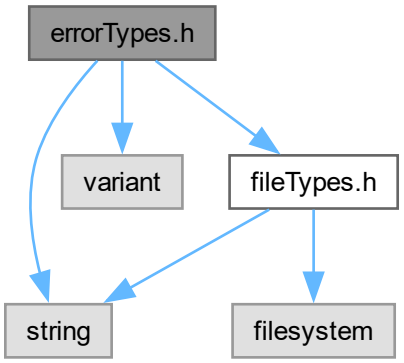
Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_TYPES_COMPILERTYPES_H
00014 #define NSBACI_TYPES_COMPILERTYPES_H
00015
00016 #include <cstdint>
00017 #include <string>
00018 #include <unordered_map>
00019 #include <vector>
00020
00025 namespace nsbaci::types {
00026
00028 using VarName = std::string;
00029
00031 using MemoryAddr = uint32_t;
00032
00034 struct SymbolInfo {
00035   VarName name;
00036   MemoryAddr address;
00037   std::string type;
00038   bool isGlobal;
00039 };
00040
00042 using SymbolTable = std::unordered_map<VarName, SymbolInfo>;
00043
00044 }  // namespace nsbaci::types
00045
00046 #endif  // NSBACI_TYPES_COMPILERTYPES_H
```

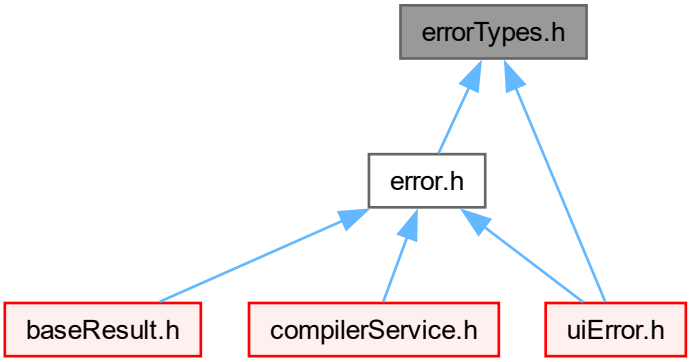
7.68 errorTypes.h File Reference

Type definitions for error-related structures.

Include dependency graph for errorTypes.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct nsbaci::types::ErrorBase

  Base structure containing common error properties.
- struct nsbaci::types::CompileError

  Error payload for compilation errors.
- struct nsbaci::types::SaveError

  Error payload for file save errors.
- struct nsbaci::types::LoadError

  Error payload for file load errors.
- struct nsbaci::types::RuntimeError

  Error payload for runtime errors.

Namespaces

- namespace nsbaci::types

  Type definitions namespace for nsbaci (runtime-specific).
- namespace nsbaci

  Root namespace for the nsbaci application.

Typedefs

- using **nsbaci::types::ErrMessage** = std::string
- using nsbaci::types::ErrorPayload

  Variant type for all possible error payloads.

Enumerations

- enum class nsbaci::types::ErrSeverity { **Warning** , **Error** , **Fatal** }

  Severity levels for errors.
- enum class nsbaci::types::ErrType {
  emptyPath , **invalidPath** , **invalidExtension** , **directoryNotFound** ,
  fileNotFound , **notARegularFile** , **permissionDenied** , **openFailed** ,
  readFailed , **writeFailed** , **compilationError** , **unknown** }

  Types of errors that can occur in the application.

7.68.1 Detailed Description

Type definitions for error-related structures.

This header provides type aliases and enums used by the Error module and other components that handle errors.

Author

Nicolás Serrano García

Copyright

7.69 errorTypes.h

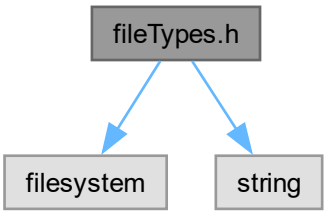
Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_TYPES_ERRORTYPES_H
00014 #define NSBACI_TYPES_ERRORTYPES_H
00015
00016 #include <string>
00017 #include <variant>
00018
00019 #include "fileTypes.h"
00020
00025 namespace nsbaci::types {
00026
00031 enum class ErrSeverity { Warning, Error, Fatal };
00032
00037 enum class ErrType {
00038   // File path errors
00039   emptyPath,          // Path string is empty
00040   invalidPath,        // Path is malformed or invalid
00041   invalidExtension,   // File does not have .nsb extension
00042   directoryNotFound,  // Parent directory doesn't exist
00043   fileNotFound,       // File doesn't exist
00044   notARegularFile,    // Path points to directory, symlink, etc.
00045
00046   // Permission errors
00047   permissionDenied,  // No read/write access
00048
00049   // I/O errors
00050   openFailed,   // Could not open file
00051   readFailed,   // Error while reading
00052   writeFailed,  // Error while writing
00053
00054   // Compilation errors
00055   compilationError,  // Syntax or semantic error during compilation
00056
00057   // Generic
00058   unknown  // Unspecified error
00059 };
00060
00061 using ErrMessage = std::string;
00062
00067 struct ErrorBase {
00068   // this severity serves as an indicator to what icon to use in the error, the
00069   // string that appears in the top of the dialog, additional buttons for things
00070   // like restart when the error is fatal...
00071   ErrSeverity severity;
00072   ErrMessage message;
00073   // useful to show the reason for things. For example, if when trying to save a
00074   // file to a sensible location, if the app doesn't have privileges, the ui can
00075   // show a message of type: "/Error/ X could not be saved in Y. Reason:
00076   // Permission Denied. Try starting the app in admin mode. [Ok]"
00077   ErrType type;
00078 };
00079
00084 struct CompileError {
00085   int line = 0;
00086   int column = 0;
00087 };
00088
00093 struct SaveError {
00094   File associatedFile;
00095 };
00096
00101 struct LoadError {
00102   File associatedFile;
00103 };
00104
00109 struct RuntimeError {};
00110
00117 using ErrorPayload =
00118     std::variant<SaveError, LoadError, CompileError, RuntimeError>;
00119
00120 }  // namespace nsbaci::types
00121
00122 #endif  // NSBACI_TYPES_ERRORTYPES_H
```

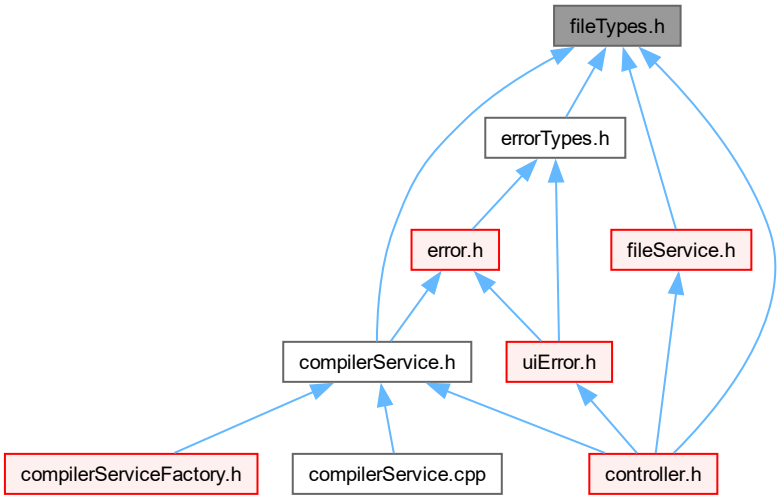
7.70 fileTypes.h File Reference

Type definitions for file-related operations.

Include dependency graph for fileTypes.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace nsbaci::types

  Type definitions namespace for nsbaci (runtime-specific).
- namespace nsbaci

  Root namespace for the nsbaci application.

Typedefs

- using **nsbaci::types::Text** = std::string

  Alias for text content (file contents, source code, etc.).
- using **nsbaci::types::File** = fs::path

  Alias for file system paths.

7.70.1 Detailed Description

Type definitions for file-related operations.

This header provides type aliases used by the FileService and other components that work with files.

Author

Nicolás Serrano García

Copyright

7.71 fileTypes.h

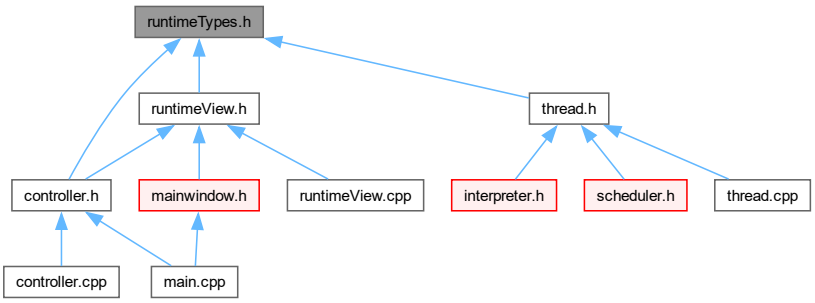
Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_TYPES_FILETYPES_H
00014 #define NSBACI_TYPES_FILETYPES_H
00015
00016 #include <filesystem>
00017 #include <string>
00018
00023 namespace nsbaci::types {
00024
00025 namespace fs = std::filesystem;
00026
00030 using Text = std::string;
00031
00035 using File = fs::path;
00036
00037 }  // namespace nsbaci::types
00038
00039 #endif  // NSBACI_TYPES_FILETYPES_H
```

7.72 runtimeTypes.h File Reference

Type definitions for runtime-related operations.

This graph shows which files directly or indirectly include this file:



Classes

- struct nsbaci::types::Address

  Represents a memory address in the runtime.

Namespaces

- namespace nsbaci::types

  Type definitions namespace for nsbaci (runtime-specific).
- namespace nsbaci

  Root namespace for the nsbaci application.

Typedefs

- using **nsbaci::types::ThreadID** = unsigned long long int
- using **nsbaci::types::Priority** = unsigned long int

Enumerations

- enum class nsbaci::types::ThreadState {
  nsbaci::types::Ready , nsbaci::types::Running , nsbaci::types::Blocked , nsbaci::types::Waiting ,
  nsbaci::types::IO , nsbaci::types::Terminated }

7.72.1 Detailed Description

Type definitions for runtime-related operations.

This header provides type aliases used by the RuntimeService and other components that work with runtime execution.

Author

Nicolás Serrano García

Copyright

7.73 runtimeTypes.h

Go to the documentation of this file.

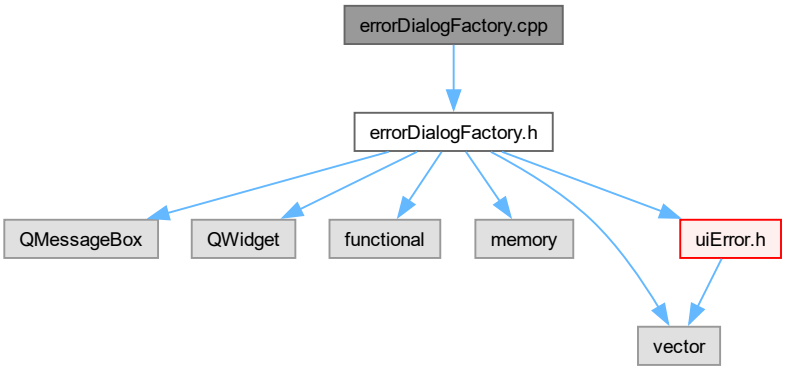
```
00001
00012
00013 #ifndef NSBACI_TYPES_RUNTIMETYPES_H
00014 #define NSBACI_TYPES_RUNTIMETYPES_H
00015
00020 namespace nsbaci::types {
00021
00022 using ThreadID = unsigned long long int;
00023
00024 using Priority = unsigned long int;
00025
00026 enum class ThreadState {
00027   Ready,
00028   Running,
00029   Blocked,
00030   Waiting,
00031   IO,
00032   Terminated
00033 };
00034
00041 struct Address {
00042   unsigned long long int value;
00043
00044   // Default constructor
00045   Address() : value(0) {}
00046
00047   // Value constructor
00048   explicit Address(unsigned long long int val) : value(val) {}
00049
00050   // Comparison operators
00051   bool operator==(const Address& other) const { return value == other.value; }
00052   bool operator!=(const Address& other) const { return value != other.value; }
00053   bool operator<(const Address& other) const { return value < other.value; }
00054   bool operator>(const Address& other) const { return value > other.value; }
00055   bool operator<=(const Address& other) const { return value <= other.value; }
00056   bool operator>=(const Address& other) const { return value >= other.value; }
00057
00058   // Arithmetic operators with Address
00059   Address operator+(const Address& other) const {
00060     return Address(value + other.value);
00061   }
00062   Address operator-(const Address& other) const {
00063     return Address(value - other.value);
00064   }
00065
00066   // Arithmetic operators with offset
00067   Address operator+(unsigned long long int offset) const {
00068     return Address(value + offset);
00069   }
00070   Address operator-(unsigned long long int offset) const {
00071     return Address(value - offset);
00072   }
00073
00074   // Compound assignment operators
00075   Address& operator+=(const Address& other) {
00076     value += other.value;
00077     return *this;
00078   }
00079   Address& operator-=(const Address& other) {
00080     value -= other.value;
00081     return *this;
00082   }
00083   Address& operator+=(unsigned long long int offset) {
00084     value += offset;
00085     return *this;
00086   }
00087   Address& operator-=(unsigned long long int offset) {
00088     value -= offset;
00089     return *this;
00090   }
00091
00092   // Increment/decrement operators
00093   Address& operator++() {  // Pre-increment
00094     ++value;
00095     return *this;
00096   }
00097   Address operator++(int) {  // Post-increment
00098     Address temp(*this);
00099     ++value;
00100     return temp;
00101   }
00102   Address& operator--() {  // Pre-decrement
```

```
00103     --value;
00104     return *this;
00105   }
00106   Address operator--(int) {  // Post-decrement
00107     Address temp(*this);
00108     --value;
00109     return temp;
00110   }
00111
00112   // Conversion operator (explicit to avoid implicit conversions)
00113   explicit operator unsigned long long int() const { return value; }
00114 };
00115
00116 }  // namespace nsbaci::types
00117
00118 #endif  // NSBACI_TYPES_RUNTIMETYPES_H
```

7.74 errorDialogFactory.cpp File Reference

Implementation of ErrorDialogFactory.

Include dependency graph for errorDialogFactory.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.

7.74.1 Detailed Description

Implementation of ErrorDialogFactory.

Author

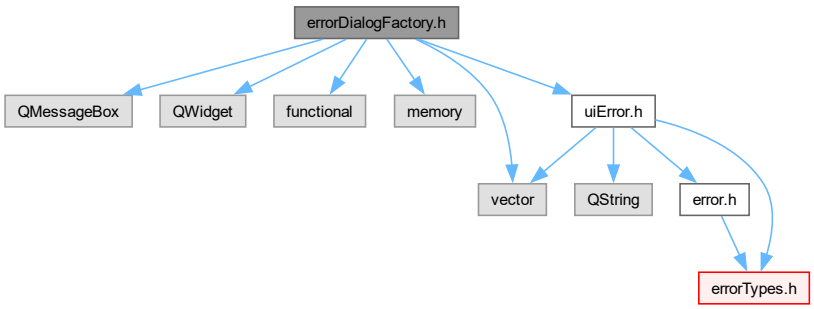
Nicolás Serrano García

Copyright

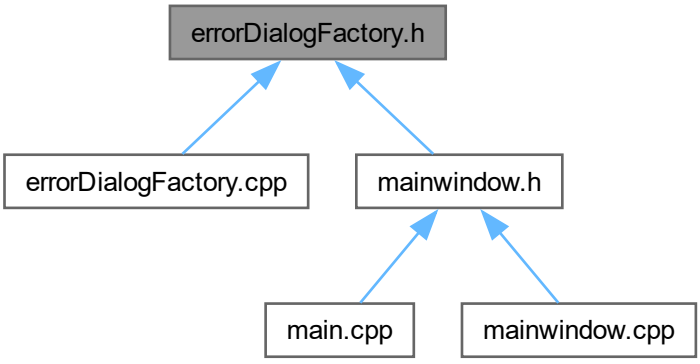
7.75 errorDialogFactory.h File Reference

Factory for creating error dialogs from UIError objects.

Include dependency graph for errorDialogFactory.h:



This graph shows which files directly or indirectly include this file:



Classes

- class nsbaci::ui::ErrorDialogFactory

  Factory for creating error dialogs from UIError objects.

Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.

7.75.1 Detailed Description

Factory for creating error dialogs from UIError objects.

This module provides a factory that converts UIError objects into ready-to-show QMessageBox dialogs. It is private to the UI layer.

7.75.1.1 Overview

The ErrorDialogFactory operates in two modes:

7.75.1.1.1 1. Deferred Mode (Factory Pattern)

Returns a **DialogInvoker** - a callable (`std::function`) that encapsulates all dialog data but does NOT show the dialog immediately. The caller decides when to invoke it. This is similar to a "lazy evaluation" or "thunk" pattern.

```
// Create the invoker (dialog is NOT shown yet)
auto dialogInvoker = ErrorDialogFactory::getDialogFromUIError(error, this);

// ... do other work, validation, logging, etc.

// Show the dialog when ready (blocks until user clicks)
QMessageBox::StandardButton clicked = dialogInvoker();

// React to user choice
if (clicked == QMessageBox::Close) {
    QApplication::quit();
}
```

7.75.1.1.2 2. Immediate Mode (Convenience)

Shows the dialog immediately and returns the result. Internally, this creates an invoker and calls it right away. Use this when you don't need to defer the dialog display.

```
// Show immediately - blocks until user clicks
ErrorDialogFactory::showError(error, this);

// Or capture the result
auto clicked = ErrorDialogFactory::showError(error, this);
```

7.75.1.2 Why Return Callables?

The DialogInvoker pattern provides:

- **Separation of concerns**: Factory knows HOW to build, caller knows WHEN to show
- **Deferred execution**: Prepare dialogs ahead of time, show when appropriate
- **User response handling**: The return value indicates which button was clicked
- **Flexibility**: Batch prepare multiple dialogs, show conditionally, etc.

The callable captures all necessary data by value, so the original UIError can be destroyed before the dialog is shown.

Author

Nicolás Serrano García

Copyright

7.76 errorDialogFactory.h

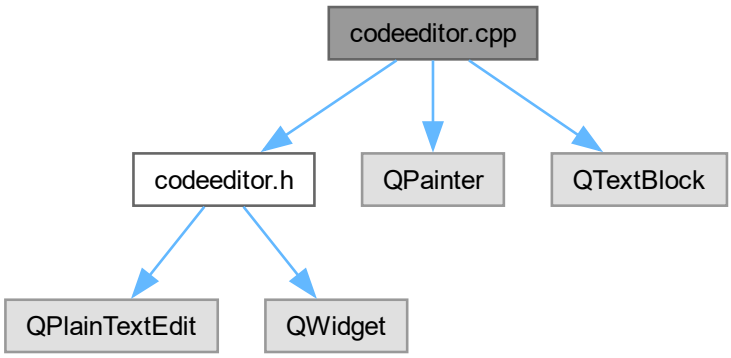
Go to the documentation of this file.

```
00001
00065
00066 #ifndef NSBACI_ERRORDIALOGFACTORY_H
00067 #define NSBACI_ERRORDIALOGFACTORY_H
00068
00069 #include <QMessageBox>
00070 #include <QWidget>
00071 #include <functional>
00072 #include <memory>
00073 #include <vector>
00074
00075 #include "uiError.h"
00076
00077 namespace nsbaci::ui {
00078
00090 class ErrorDialogFactory {
00091  public:
00102   using DialogInvoker = std::function<QMessageBox::StandardButton()>;
00103
00114   static DialogInvoker getDialogFromUIError(const UIError& error,
00115                                             QWidget* parent = nullptr);
00116
00124   static std::vector<DialogInvoker> getDialogsFromUIErrors(
00125       const std::vector<UIError>& errors, QWidget* parent = nullptr);
00126
00135   static DialogInvoker getSuccessDialog(const QString& title,
00136                                         const QString& message,
00137                                         QWidget* parent = nullptr);
00138
00148   static void showErrors(const std::vector<UIError>& errors,
00149                          QWidget* parent = nullptr);
00150
00158   static QMessageBox::StandardButton showError(const UIError& error,
00159                                                QWidget* parent = nullptr);
00160
00168   static void showSuccess(const QString& title, const QString& message,
00169                           QWidget* parent = nullptr);
00170
00171  private:
00175   static QMessageBox::Icon iconFromSeverity(
00176       nsbaci::types::ErrSeverity severity);
00177 };
00178
00179 }  // namespace nsbaci::ui
00180
00181 #endif  // NSBACI_ERRORDIALOGFACTORY_H
```

7.77 codeeditor.cpp File Reference

Implementation of the CodeEditor class with line numbers.

Include dependency graph for codeeditor.cpp:



7.77.1 Detailed Description

Implementation of the CodeEditor class with line numbers.

Author

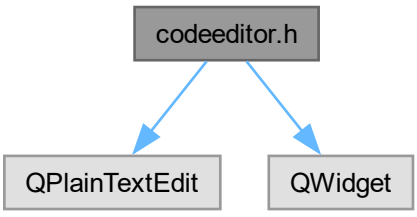
Nicolás Serrano García

Copyright

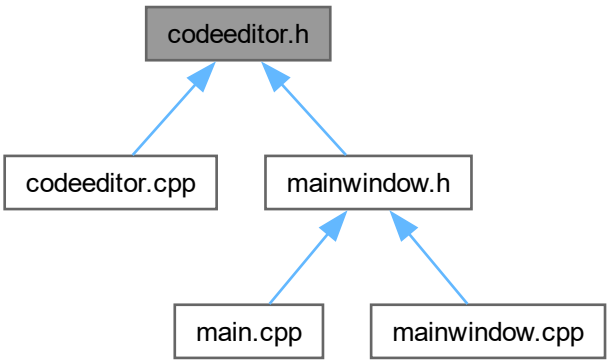
7.78 codeeditor.h File Reference

CodeEditor class with line numbers for nsbaci.

Include dependency graph for codeeditor.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CodeEditor
- class LineNumberArea

7.78.1 Detailed Description

CodeEditor class with line numbers for nsbaci.

Author

Nicolás Serrano García

Copyright

7.79 codeeditor.h

Go to the documentation of this file.

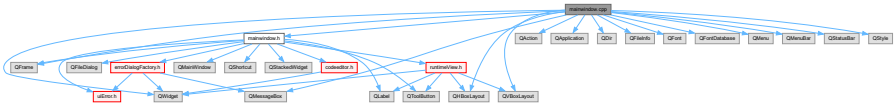
```
00001
00009
00010 #ifndef CODEEDITOR_H
00011 #define CODEEDITOR_H
00012
00013 #include <QPlainTextEdit>
00014 #include <QWidget>
00015
00016 class LineNumberArea;
00017
00018 class CodeEditor : public QPlainTextEdit {
00019   Q_OBJECT
00020
00021  public:
00022   explicit CodeEditor(QWidget* parent = nullptr);
```

```
00023
00024   void lineNumberAreaPaintEvent(QPaintEvent* event);
00025   int lineNumberAreaWidth();
00026
00027  protected:
00028   void resizeEvent(QResizeEvent* event) override;
00029
00030  private slots:
00031   void updateLineNumberAreaWidth(int newBlockCount);
00032   void highlightCurrentLine();
00033   void updateLineNumberArea(const QRect& rect, int dy);
00034
00035  private:
00036   LineNumberArea* lineNumberArea;
00037 };
00038
00039 class LineNumberArea : public QWidget {
00040  public:
00041   explicit LineNumberArea(CodeEditor* editor)
00042       : QWidget(editor), codeEditor(editor) {}
00043
00044   QSize sizeHint() const override {
00045     return QSize(codeEditor->lineNumberAreaWidth(), 0);
00046   }
00047
00048  protected:
00049   void paintEvent(QPaintEvent* event) override {
00050     codeEditor->lineNumberAreaPaintEvent(event);
00051   }
00052
00053  private:
00054   CodeEditor* codeEditor;
00055 };
00056
00057 #endif  // CODEEDITOR_H
```

7.80 mainwindow.cpp File Reference

Implementation of the MainWindow class.

Include dependency graph for mainwindow.cpp:



7.80.1 Detailed Description

Implementation of the MainWindow class.

Author

Nicolás Serrano García

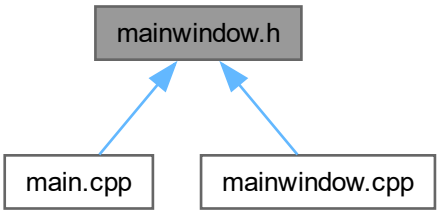
Copyright

7.81 mainwindow.h File Reference

Main window class declaration for nsbaci.

Include dependency graph for mainwindow.h:

This graph shows which files directly or indirectly include this file:



Classes

- class MainWindow

7.81.1 Detailed Description

Main window class declaration for nsbaci.

Author

Nicolás Serrano García

Copyright

7.82 mainwindow.h

Go to the documentation of this file.

```
00001
00009
00010 #ifndef MAINWINDOW_H
00011 #define MAINWINDOW_H
00012
00013 #include <QFileDialog>
00014 #include <QFrame>
00015 #include <QLabel>
00016 #include <QMainWindow>
00017 #include <QShortcut>
00018 #include <QStackedWidget>
00019 #include <QToolButton>
00020
00021 #include "codeeditor.h"
00022 #include "errorDialogFactory.h"
00023 #include "runtimeView.h"
00024 #include "uiError.h"
00025
00026 QT_BEGIN_NAMESPACE
00027 namespace Ui {
00028 class MainWindow;
00029 }
00030 QT_END_NAMESPACE
00031
00032 class MainWindow : public QMainWindow {
00033   Q_OBJECT
00034
00035  public:
00036   explicit MainWindow(QWidget* parent = nullptr);
00037   ~MainWindow() override = default;
00038
00039  signals:
00040   void saveRequested(const QString& filePath, const QString& contents);
00041   void openRequested(const QString& filePath);
00042   void compileRequested(const QString& contents);
00043   void runRequested();
00044
00045   // Runtime control signals
00046   void stepRequested();
00047   void stepThreadRequested(nsbaci::types::ThreadID threadId);
00048   void runContinueRequested();
00049   void pauseRequested();
00050   void resetRequested();
00051   void stopRequested();
00052   void inputProvided(const QString& input);
00053
00054  public slots:
00055   void setEditorContents(const QString& contents);
00056   void setStatusMessage(const QString& message);
00057   void setCurrentFile(const QString& fileName, bool modified = false);
00058
00059   // Controller response slots
00060   void onSaveSucceeded();
00061   void onSaveFailed(std::vector<nsbaci::UIError> errors);
00062   void onLoadSucceeded(const QString& contents);
00063   void onLoadFailed(std::vector<nsbaci::UIError> errors);
00064   void onCompileSucceeded();
00065   void onCompileFailed(std::vector<nsbaci::UIError> errors);
00066
00067   // Runtime slots
00068   void onRunStarted(const QString& programName);
00069   void onRuntimeStateChanged(bool running, bool halted);
00070   void onThreadsUpdated(const std::vector<nsbaci::ui::ThreadInfo>& threads);
00071   void onVariablesUpdated(
00072       const std::vector<nsbaci::ui::VariableInfo>& variables);
00073   void onOutputReceived(const QString& output);
00074   void onInputRequested(const QString& prompt);
00075
00076  private slots:
00077   // File menu
00078   void onNew();
00079   void onSave();
00080   void onSaveAs();
00081   void onOpen();
00082   void onExit();
00083   void onCompile();
00084   void onRun();
00085
00086   // Edit menu
00087   void onUndo();
00088   void onRedo();
00089   void onCut();
```

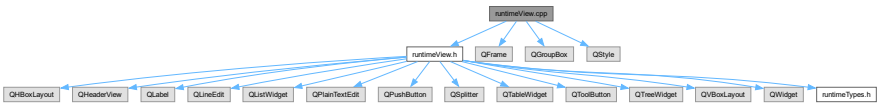
```
00090   void onCopy();
00091   void onPaste();
00092   void onSelectAll();
00093
00094   // View menu
00095   void onToggleSidebar();
00096   void onToggleFullscreen();
00097
00098   // Help menu
00099   void onAbout();
00100
00101   // Editor
00102   void onTextChanged();
00103
00104   // Runtime view
00105   void onStopRuntime();
00106
00107  private:
00108   // Stacked widget for switching views
00109   QStackedWidget* centralStack = nullptr;
00110
00111   // Editor view container
00112   QWidget* editorView = nullptr;
00113
00114   // File info bar
00115   QFrame* fileInfoBar = nullptr;
00116   QLabel* fileNameLabel = nullptr;
00117   QLabel* fileModifiedIndicator = nullptr;
00118   QLabel* compileStatusIndicator = nullptr;
00119
00120   // Central widget
00121   CodeEditor* codeEditor = nullptr;
00122
00123   // Sidebar
00124   QFrame* sideBar = nullptr;
00125   QToolButton* compileButton = nullptr;
00126   QToolButton* runButton = nullptr;
00127
00128   // Runtime view
00129   nsbaci::ui::RuntimeView* runtimeView = nullptr;
00130
00131   // File actions
00132   QAction* actionNew = nullptr;
00133   QAction* actionSave = nullptr;
00134   QAction* actionSaveAs = nullptr;
00135   QAction* actionOpen = nullptr;
00136   QAction* actionExit = nullptr;
00137
00138   // Edit actions
00139   QAction* actionUndo = nullptr;
00140   QAction* actionRedo = nullptr;
00141   QAction* actionCut = nullptr;
00142   QAction* actionCopy = nullptr;
00143   QAction* actionPaste = nullptr;
00144   QAction* actionSelectAll = nullptr;
00145
00146   // View actions
00147   QAction* actionToggleSidebar = nullptr;
00148   QAction* actionFullscreen = nullptr;
00149
00150   // Build actions
00151   QAction* actionCompile = nullptr;
00152   QAction* actionRun = nullptr;
00153
00154   // Help actions
00155   QAction* actionAbout = nullptr;
00156
00157   // State
00158   QString currentFileName;  // Just the filename for display
00159   QString currentFilePath;  // Full path for saving
00160   bool isModified = false;
00161   bool hasName = false;
00162   bool isCompiled =
00163       false;  // True when program is compiled and code hasn't changed
00164
00165  private:
00166   void createCentralWidget();
00167   void createEditorView();
00168   void createRuntimeView();
00169   void createMenuBar();
00170   void createStatusBar();
00171   void setupShortcuts();
00172   void applyStyleSheet();
00173
00174   void switchToEditor();
00175   void switchToRuntime();
00176 };
```

```
00177 #endif  // MAINWINDOW_H
```

7.83 runtimeView.cpp File Reference

Implementation of the RuntimeView widget.

Include dependency graph for runtimeView.cpp:



Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.

7.83.1 Detailed Description

Implementation of the RuntimeView widget.

Author

Nicolás Serrano García

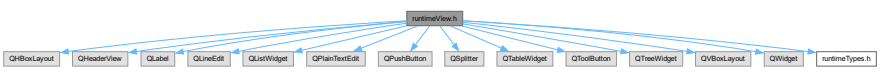
Copyright

Copyright (c) 2025 Nicolás Serrano García. Licensed under the MIT License.

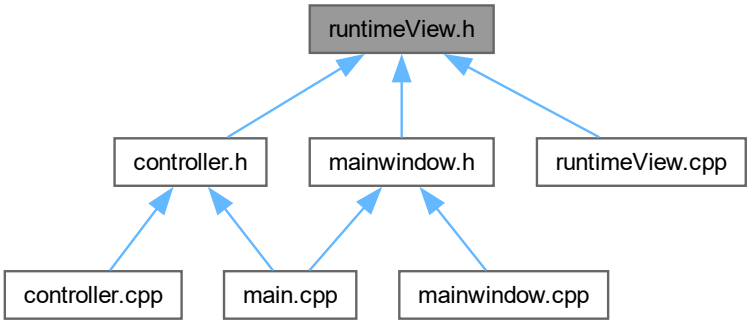
7.84 runtimeView.h File Reference

RuntimeView widget declaration for nsbaci.

Include dependency graph for runtimeView.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct nsbaci::ui::ThreadInfo

  Information about a thread for display.
- struct nsbaci::ui::VariableInfo

  Information about a variable for display.
- class nsbaci::ui::RuntimeView

  Widget displaying runtime execution state.

Namespaces

- namespace nsbaci

  Root namespace for the nsbaci application.

7.84.1 Detailed Description

RuntimeView widget declaration for nsbaci.

This widget displays runtime information when executing a program: variables, threads, I/O console, and execution controls.

Author

Nicolás Serrano García

Copyright

7.85 runtimeView.h

Go to the documentation of this file.

```
00001
00012
00013 #ifndef NSBACI_RUNTIMEVIEW_H
00014 #define NSBACI_RUNTIMEVIEW_H
00015
00016 #include <QHBoxLayout>
00017 #include <QHeaderView>
00018 #include <QLabel>
00019 #include <QLineEdit>
00020 #include <QListWidget>
00021 #include <QPlainTextEdit>
00022 #include <QPushButton>
00023 #include <QSplitter>
00024 #include <QTableWidget>
00025 #include <QToolButton>
00026 #include <QTreeWidget>
00027 #include <QVBoxLayout>
00028 #include <QWidget>
00029
00030 #include "runtimeTypes.h"
00031
00032 namespace nsbaci::ui {
00033
00038 struct ThreadInfo {
00039   nsbaci::types::ThreadID id;
00040   nsbaci::types::ThreadState state;
00041   size_t pc;
00042   QString currentInstruction;
00043 };
00044
00049 struct VariableInfo {
00050   QString name;
00051   QString type;
00052   QString value;
00053   size_t address;
00054 };
00055
00066 class RuntimeView : public QWidget {
00067   Q_OBJECT
00068
00069  public:
00070   explicit RuntimeView(QWidget* parent = nullptr);
00071   ~RuntimeView() override = default;
00072
00073  signals:
00074   // Execution control signals
00075   void stepRequested();
00076   void stepThreadRequested(nsbaci::types::ThreadID threadId);
00077   void runRequested();
00078   void pauseRequested();
00079   void resetRequested();
00080   void stopRequested();
00081
00082   // I/O signals
00083   void inputProvided(const QString& input);
00084
00085  public slots:
00086   // Update display
00087   void updateThreads(const std::vector<ThreadInfo>& threads);
00088   void updateVariables(const std::vector<VariableInfo>& variables);
00089   void updateCurrentInstruction(const QString& instruction);
00090   void updateExecutionState(bool running, bool halted);
00091
00092   // I/O
00093   void appendOutput(const QString& text);
00094   void requestInput(const QString& prompt);
00095   void clearConsole();
00096
00097   // State
00098   void onProgramLoaded(const QString& programName);
00099   void onProgramHalted();
00100
00101  private slots:
00102   void onStepClicked();
00103   void onRunClicked();
00104   void onPauseClicked();
00105   void onResetClicked();
00106   void onStopClicked();
00107   void onInputSubmitted();
00108   void onThreadSelected(QTreeWidgetItem* item, int column);
00109
00110  private:
```

```
00111     void createUI();
00112     void createToolbar();
00113     void createThreadPanel();
00114     void createVariablePanel();
00115     void createConsolePanel();
00116     void applyStyleSheet();
00117 
00118     // Toolbar
00119     QWidget* toolbar = nullptr;
00120     QToolButton* stepButton = nullptr;
00121     QToolButton* runButton = nullptr;
00122     QToolButton* pauseButton = nullptr;
00123     QToolButton* resetButton = nullptr;
00124     QToolButton* stopButton = nullptr;
00125     QLabel* statusLabel = nullptr;
00126 
00127     // Thread panel
00128     QTreeWidget* threadTree = nullptr;
00129 
00130     // Variable panel
00131     QTableWidget* variableTable = nullptr;
00132 
00133     // Console panel
00134     QPlainTextEdit* consoleOutput = nullptr;
00135     QLineEdit* consoleInput = nullptr;
00136     QPushButton* inputSubmitButton = nullptr;
00137     QLabel* inputPromptLabel = nullptr;
00138 
00139     // State
00140     bool isRunning = false;
00141     bool isHalted = false;
00142     bool waitingForInput = false;
00143     nsbaci::types::ThreadID selectedThread = 0;
00144 };
00145 
00146 }  // namespace nsbaci::ui
00147 
00148 #endif  // NSBACI_RUNTIMEVIEW_H
```