LibSledDebugger API Reference

This document provides information about the API elements that are used in the successful implementation of LibSledDebugger.

Note:
The specifications contained in this document are subject to change without prior notice.

Version	Revision Date	Author(s)	Comments
2.0	01-August-08	Sage Knowles, Patrick O'Leary	
3.5	August-2010	Risa Galant, Patrick O'Leary	
5.0.0	Apr-14	Gary Staas	Updates for 5.0.0. Update format.
5.1.0	Oct-14	Gary Staas	Updates for 5.1.0.
5.1.2	Feb-15	Gary Staas	Open source version.

Table of Contents

About the LibSledDebugger API Reference

This document provides information about the API elements that are used in implementing LibSledDebugger. Use this document as a reference to the specifics of the API, such as its functions, data structures, type definitions, and defined symbols. This is a C API.

All of the information in this document is taken from comments in the header (`*.h`) files that are located in the `components\sce_sled\src\ sleddebugger` directory.

SledDebugger Class

`SledDebugger` is the class for the SLED debugger object that allows debugging scripts during run time. To debug an application with SLED, the application must create a `SledDebugger` instance when it runs. The SLED GUI communicates with `SledDebugger` instances. The LibSledDebugger library handles `SledDebugger` instance creation.

What Is and Is Not Included in this Reference

This reference document contains information that is most useful for using components of the LibSledDebugger API. Header files that contain information that is mostly for internal use, or platform-specific information that is defined elsewhere, are not included in this reference. If you are digging into the code in more depth, you can look in the header files for additional information.

The following LibSledLuaPlugin header files in `components\sce_sled\src\sleddebugger` are included in this reference document and contain the public API:

- `assert.h`
- `buffer.h`
- `params.h`
- `plugin.h`
- `scmp.h`
- `sequentialallocator.h`
- `sleddebugger.h`

- `stringarray.h`
- `timer.h`
- `utilities.h`

Most of the information from items tagged with the following identifiers is not included in this reference document:

- `private`
- `protected`

Look in the header files if you want more information about the `private` and `protected` class members, or any of the files that are not included in this reference.

Related Documentation

The following documentation, available on SHIP, contains useful information about using the SLED and Lua Toolset.

- *Getting Started with SLED*
- *SLED User's Guide*
- *SLED Plugin Guide*

Introduction

Library Summary

Library Contents

Item	Description
sce	sce namespace.
sce::Sled	SLED namespace.
sce::Sled::Assert	Assert namespace.
sce::Sled::DebuggerMode	DebuggerMode enum namespace.
sce::Sled::Protocol	Protocol enum namespace.
sce::Sled::SCMP	SLED Control Message Protocol namespace.
sce::Sled::SCMP::TypeCodes	Scoping TypeCodes enumeration namespace.
sce::Sled::Utilities	Utilities namespace.
sce::Sled::SledDebugger	Class describing a SLED debugger instance.
sce::Sled::SledDebuggerPlugin	Language plugin abstract base class.
sce::Sled::Timer	Multi-platform timer.
sce::Sled::BreakpointParams	Breakpoint params struct.
sce::Sled::NetworkParams	Struct that describes details of network configuration structure.
sce::Sled::SCMP::Base	SLED Control Message Protocol base network message structure.
sce::Sled::SledDebuggerConfig	Structure describing details of SledDebugger instance.
sce::Sled::Version	Version detail.

Defines

Define Summary

Define	Value	Description
`SCE_LIBSLEDDEBUGGER_VER_MAJOR`	5	LibSledDebugger version details - major version number.
`SCE_LIBSLEDDEBUGGER_VER_MINOR`	1	LibSledDebugger version details - minor version number.
`SCE_LIBSLEDDEBUGGER_VER_REVISION`	1	LibSledDebugger version details - revision version number.
`SCE_LIBSLEDDEBUGGER_VER_OTHER`	0	LibSledDebugger version details - extra version number.

sce namespace

Summary

sce

sce namespace.

Definition

```
namespace sce {}
```

Description

Namespace for sce classes and functions.

Inner Classes, Structures, and Namespaces

Item	Description
sce::Sled	SLED namespace.

sce::Sled namespace

Summary

sce::Sled

SLED namespace.

Definition

```
namespace Sled {}
```

Description

Namespace for Sled classes and functions.

Function Summary

Function	Description
debuggerAddPlugin	Add plugin to SledDebugger.
debuggerBreakpointReached	Plugins call this function when they encounter a breakpoint.
debuggerCreate	Create SledDebugger instance.
debuggerGenerateHash	Generate simple hash from string and line number.
debuggerGetDebuggerMode	Get current debugger mode.
debuggerGetVersion	Get SledDebugger version information.
debuggerIsConnected	Determine whether or not SLED client connected.
debuggerIsNetworking	Determine whether or not networking is enabled.
debuggerRemovePlugin	Remove plugin from SledDebugger.
debuggerRequiredMemory	Calculate size in bytes required for SledDebugger instance based on configuration structure.
debuggerScriptCacheAdd	Add a script file to internal list of scripts so that when SLED connects it knows which scripts are being debugged.
debuggerScriptCacheClear	Clear internal list of scripts being debugged.
debuggerScriptCacheRemove	Remove script file from internal list of scripts so that when SLED connects it knows which scripts are being debugged.
debuggerShutdown	Shut down SledDebugger instance.
debuggerStartNetworking	Initialize networking and optionally block execution until connection is made.
debuggerStopNetworking	Stop networking (disconnect SLED if connected).
debuggerTtyNotify	Send message to SLED's TTY window.
debuggerUpdate	Poll sockets and process any incoming messages.

Inner Classes, Structures, and Namespaces

Item	Description
sce::Sled::Assert	Assert namespace.
sce::Sled::BreakpointParams	Breakpoint params struct.
sce::Sled::DebuggerMode	DebuggerMode enum namespace.
sce::Sled::NetworkParams	Struct that describes details of network configuration structure.
sce::Sled::Protocol	Protocol enum namespace.
sce::Sled::SCMP	SLED Control Message Protocol namespace.
sce::Sled::SledDebugger	Class describing a SLED debugger instance.

Item	Description
sce::Sled::SledDebuggerConfig	Structure describing details of `SledDebugger` instance.
sce::Sled::SledDebuggerPlugin	Language plugin abstract base class.
sce::Sled::Timer	Multi-platform timer.
sce::Sled::Utilities	`Utilities` namespace.
sce::Sled::Version	`Version` detail.

Functions

debuggerAddPlugin

Add plugin to SledDebugger.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerAddPlugin(
            SledDebugger *debugger,
            SledDebuggerPlugin *plugin
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

debugger	SledDebugger to use
plugin	Language plugin to add

Return Values

Value	Description
SCE_SLED_ERROR_OK	Success
SCE_SLED_ERROR_NULLPARAMETER	Null debugger or plugin
SCE_SLED_ERROR_INVALIDPLUGIN	Invalid plugin
SCE_SLED_ERROR_MAXPLUGINSREACHED	Maximum number of plugins reached
SCE_SLED_ERROR_PLUGINALREADYADDED	Plugin already added

Description

Add a plugin to SledDebugger.

See Also

debuggerRemovePlugin, debuggerScriptCacheAdd

debuggerBreakpointReached

Plugins call this function when they encounter a breakpoint.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerBreakpointReached(
            SledDebugger *debugger,
            const BreakpointParams *params
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

debugger	SledDebugger to use
params	Breakpoint parameters, including file, line number, and the plugin that hit the breakpoint

Return Values

Value	Description
SCE_SLED_ERROR_OK	Success
SCE_SLED_ERROR_NULLPARAMETER	Null debugger or params
SCE_SLED_ERROR_NOTNETWORKING	Not networking
SCE_SLED_ERROR_NOCLIENTCONNECTED	SLED is not connected

Description

Plugins call the `debuggerBreakpointReached()` function when they encounter a breakpoint. `debuggerBreakpointReached()` notifies other plugins that a breakpoint has been reached, and then handles breakpoint synchronization and communication with SLED. `debuggerBreakpointReached()` can be commandeered to forcibly halt execution and break in SLED if needed.

debuggerCreate

Create SledDebugger instance.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerCreate(
            const SledDebuggerConfig *config,
            void *location,
            SledDebugger **outDebugger
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

config	Configuration structure that details the settings to use
location	Location in memory in which to place the SledDebugger instance. It needs to be as big as the value returned by debuggerRequiredMemory().
outDebugger	SledDebugger instance that is created

Return Values

Value	Description
SCE_SLED_ERROR_OK	Success
SCE_SLED_ERROR_NULLPARAMETER	Configuration structure is null
SCE_SLED_ERROR_INVALIDCONFIGURATION	Invalid value in the configuration structure

Description

Create a SledDebugger instance.

See Also

debuggerRequiredMemory, debuggerShutdown

debuggerGenerateHash

Generate simple hash from string and line number.

Definition

```
#include <sleddebugger.h>
namespace sce {
   namespace Sled {
      int32_t debuggerGenerateHash(
         const char *pszString,
         int32_t line,
         int32_t *outHash
      );
   }
}
```

Calling Conditions

Multithread safe.

Arguments

pszString	String to hash
line	Line number that gets used in the hash
outHash	Hash if the function was successful

Return Values

Value	Description
SCE_SLED_ERROR_OK	Success
SCE_SLED_ERROR_NULLPARAMETER	pszString or outHash is null
SCE_SLED_ERROR_INVALIDPARAMETER	pszString is empty

Description

Generate a simple hash from a string and a line number.

debuggerGetDebuggerMode

Get current debugger mode.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerGetDebuggerMode(
            const SledDebugger *debugger,
            DebuggerMode::Enum *outDebuggerMode
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

debugger	`SledDebugger` to use
outDebuggerMode	Current debugger mode

Return Values

Value	Description
`SCE_SLED_ERROR_OK`	Success
`SCE_SLED_ERROR_NULLPARAMETER`	Null debugger or `outDebuggerMode`

Description

Get the current debugger mode. `debuggerGetDebuggerMode()` is used primarily by language plugins.

debuggerGetVersion

Get SledDebugger version information.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerGetVersion(
            const SledDebugger *debugger,
            Version *outVersion
        );
    }
}
```

Calling Conditions

Multithread safe.

Arguments

debugger	SledDebugger to use
outVersion	Version information for SledDebugger

Return Values

Value	Description
SCE_SLED_ERROR_OK	Success
SCE_SLED_ERROR_NULLPARAMETER	Null debugger or outVersion

Description

Get version information for SledDebugger.

debuggerIsConnected

Determine whether or not SLED client connected.

Definition

```
#include <sleddebugger.h>
namespace sce {
   namespace Sled {
      int32_t debuggerIsConnected(
         const SledDebugger *debugger,
         bool *outResult
      );
   }
}
```

Calling Conditions

Not multithread safe.

Arguments

debugger	SledDebugger to use
outResult	True if a client is connected, false if a client is not connected

Return Values

Value	Description
SCE_SLED_ERROR_OK	Success
SCE_SLED_ERROR_NULLPARAMETER	Null debugger or outResult

Description

Determine whether or not a SLED client is connected.

See Also

debuggerStartNetworking, debuggerStopNetworking, debuggerIsNetworking

debuggerIsNetworking

Determine whether or not networking is enabled.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerIsNetworking(
            const SledDebugger *debugger,
            bool *outResult
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

debugger	SledDebugger to use
outResult	True if debuggerStartNetworking() has been called but debuggerStopNetworking() has not been called yet, false otherwise

Return Values

Value	Description
SCE_SLED_ERROR_OK	Success
SCE_SLED_ERROR_NULLPARAMETER	Null debugger or outResult

Description

Determine whether or not networking is enabled. LibSledDebugger can only accept connections between the time that debuggerStartNetworking() has been called and the time that debuggerStopNetworking() has been called. During that period of time, debuggerIsNetworking() returns true. Outside of that period of time, debuggerIsNetworking() returns false.

See Also

debuggerStartNetworking, debuggerStopNetworking, debuggerIsConnected

debuggerRemovePlugin

Remove plugin from SledDebugger.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerRemovePlugin(
            SledDebugger *debugger,
            SledDebuggerPlugin *plugin
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

debugger	SledDebugger to use
plugin	Language plugin to remove

Return Values

Value	Description
SCE_SLED_ERROR_OK	Success
SCE_SLED_ERROR_NULLPARAMETER	Null debugger or plugin
SCE_SLED_ERROR_INVALIDPLUGIN	Invalid plugin
SCE_SLED_ERROR_SRCH	Plugin not found or doesn't exist

Description

Remove a plugin from SledDebugger.

See Also

debuggerAddPlugin, debuggerScriptCacheRemove

debuggerRequiredMemory

Calculate size in bytes required for `SledDebugger` instance based on configuration structure.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerRequiredMemory(
            const SledDebuggerConfig *config,
            std::size_t *outRequiredMemory
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

config	Configuration structure that details the settings to use
outRequiredMemory	The amount of memory that is needed for the `SledDebugger` instance

Return Values

Value	Description
SCE_SLED_ERROR_OK	Success
SCE_SLED_ERROR_NULLPARAMETER	Configuration structure is null
SCE_SLED_ERROR_INVALIDCONFIGURATION	Invalid value in the configuration structure

Description

Calculate the size in bytes required for a `SledDebugger` instance based on a configuration structure.

See Also

`debuggerCreate`

debuggerScriptCacheAdd

Add a script file to internal list of scripts so that when SLED connects it knows which scripts are being debugged.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerScriptCacheAdd(
            SledDebugger *debugger,
            const char *relativePathToScriptFile,
            bool *outResult
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

debugger	SledDebugger to use
relativePathToScriptFile	Relative path (from the asset directory) of the file
outResult	True if the file is added to internal list, false if the file is not added to internal list

Return Values

Value	Description
SCE_SLED_ERROR_OK	Success
SCE_SLED_ERROR_NULLPARAMETER	Null debugger or outResult

Description

Add a script file to the internal list of scripts, so that when SLED connects it knows which scripts are being debugged. Path should be relative to the asset directory. SLED will try to open the file from its asset directory.

See Also

debuggerAddPlugin, debuggerScriptCacheRemove, debuggerScriptCacheClear

debuggerScriptCacheClear

Clear internal list of scripts being debugged.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerScriptCacheClear(
            SledDebugger *debugger
        );
    }
}
```

Arguments

debugger	SledDebugger to use

Return Values

Value	Description
SCE_SLED_ERROR_OK	Success
SCE_SLED_ERROR_NULLPARAMETER	Null debugger

Description

Clear the internal list of scripts being debugged.

See Also

debuggerScriptCacheAdd, debuggerScriptCacheRemove

debuggerScriptCacheRemove

Remove script file from internal list of scripts so that when SLED connects it knows which scripts are being debugged.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerScriptCacheRemove(
            SledDebugger *debugger,
            const char *relativePathToScriptFile,
            bool *outResult
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

debugger	`SledDebugger` to use
relativePathToScriptFile	Relative path (from the asset directory) of the file
outResult	True if the file is removed from the internal list, false if the file is not removed from the internal list

Return Values

Value	Description
`SCE_SLED_ERROR_OK`	Success
`SCE_SLED_ERROR_NULLPARAMETER`	Null debugger or `outResult`

Description

Remove a script file from the internal list of scripts, so that when SLED connects it knows which scripts are being debugged.

See Also

`debuggerScriptCacheAdd`, `debuggerScriptCacheClear`, `debuggerRemovePlugin`

debuggerShutdown

Shut down `SledDebugger` instance.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerShutdown(
            SledDebugger *debugger
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

debugger	`SledDebugger` instance to shut down

Return Values

Value	Description
`SCE_SLED_ERROR_OK`	Success
`SCE_SLED_ERROR_NULLPARAMETER`	Null debugger

Description

Shut down a `SledDebugger` instance.

See Also

`debuggerCreate`

debuggerStartNetworking

Initialize networking and optionally block execution until connection is made.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerStartNetworking(
            SledDebugger *debugger
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

debugger	SledDebugger to use

Return Values

Value	Description
SCE_SLED_ERROR_OK	Success
SCE_SLED_ERROR_NULLPARAMETER	Null debugger
SCE_SLED_ERROR_ALREADYNETWORKING	Already networking
SCE_SLED_ERROR_NOTINITIALIZED	Not initialized
SCE_SLED_ERROR_NETSUBSYSTEMFAIL	Network subsystem failed
SCE_SLED_ERROR_TCPSOCKETINITFAIL	Tcp socket initialization failed
SCE_SLED_ERROR_TCPNONBLOCKINGFAIL	Tcp socket set non-blocking mode failed
SCE_SLED_ERROR_TCPLISTENFAIL	Tcp socket failed to listen
SCE_SLED_ERROR_INVALIDPROTOCOL	Invalid network protocol

Description

Initialize networking and optionally block execution until a connection is made.

See Also

debuggerStopNetworking, debuggerIsConnected, debuggerUpdate, debuggerIsNetworking

debuggerStopNetworking

Stop networking (disconnect SLED if connected).

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerStopNetworking(
            SledDebugger *debugger
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

debugger	`SledDebugger` to use

Return Values

Value	Description
`SCE_SLED_ERROR_OK`	Success
`SCE_SLED_ERROR_NULLPARAMETER`	Null debugger
`SCE_SLED_ERROR_NOTNETWORKING`	Not networking

Description

Stop networking (disconnect SLED if connected). `debuggerStopNetworking()` expects `debuggerStartNetworking()` to have already been called.

See Also

`debuggerStartNetworking`, `debuggerIsConnected`, `debuggerUpdate`, `debuggerIsNetworking`

debuggerTtyNotify

Send message to SLED's TTY window.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerTtyNotify(
            SledDebugger *debugger,
            const char *pszMessage
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

debugger	`SledDebugger` to use
pszMessage	Data to send

Return Values

Value	Description
`SCE_SLED_ERROR_OK`	Success
`SCE_SLED_ERROR_NULLPARAMETER`	Null debugger or `pszMessage`
`SCE_SLED_ERROR_INVALIDPARAMETER`	`pszMessage` is empty
`SCE_SLED_ERROR_NOTNETWORKING`	Not networking
`SCE_SLED_ERROR_NOCLIENTCONNECTED`	SLED is not connected

Description

Send a message to SLED's TTY window.

debuggerUpdate

Poll sockets and process any incoming messages.

Definition

```
#include <sleddebugger.h>
namespace sce {
    namespace Sled {
        int32_t debuggerUpdate(
            SledDebugger *debugger
        );
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

debugger	`SledDebugger` to update

Return Values

Value	Description
`SCE_SLED_ERROR_OK`	Success
`SCE_SLED_ERROR_NULLPARAMETER`	Null debugger
`SCE_SLED_ERROR_NOTNETWORKING`	Not networking
`SCE_SLED_ERROR_RECURSIVEUPDATE`	Attempt to call `debuggerUpdate()` recursively
`SCE_SLED_ERROR_INVALIDPROTOCOL`	Invalid network protocol
`SCE_SLED_ERROR_TCPSOCKETINVALID`	Tcp socket is invalid
`SCE_SLED_ERROR_TCPSOCKETINITFAIL`	Tcp socket initialization failed
`SCE_SLED_ERROR_EVENTQUEUEESRCH`	Event queue invalid ID
`SCE_SLED_ERROR_EVENTQUEUECANCELED`	Event queue was forcibly destroyed
`SCE_SLED_ERROR_EVENTQUEUEINVAL`	Event queue invalid value specified
`SCE_SLED_ERROR_EVENTQUEUEABORT`	Event queue will be destroyed because the process terminated
`SCE_SLED_ERROR_NEGOTIATION`	Negotiation with SLED failed

Description

Poll sockets and process any incoming messages. `debuggerUpdate()` should be called from the main game loop every frame. Return an error if `debuggerStartNetworking()` has not been called or if `debuggerStopNetworking()` has been called.

See Also

`debuggerStartNetworking`, `debuggerStopNetworking`, `debuggerIsConnected`, `debuggerIsNetworking`

sce::Sled::Assert namespace

Summary

sce::Sled::Assert

`Assert` namespace.

Definition

```
namespace Assert {}
```

Description

Namespace for `Assert` classes and functions.

Function Summary

Function	Description
`assertHandler`	Get assert handler.
`reportFailure`	Report failure.
`setAssertHandler`	Set assert handler.

Enumerated Types

FailureBehavior

FailureBehavior enumeration.

Definition

```
#include <assert.h>
namespace sce {
    namespace Sled {
        namespace Assert {
            enum FailureBehavior {
                kHalt,
                kContinue
            };
        }
    }
}
```

Enumeration Values

Macro	Value	Description
`kHalt`	N/A	Halt execution.
`kContinue`	N/A	Continue execution.

Description

Failure behavior enumeration.

Type Definitions

Handler

Typedef for assert failure handler.

Definition

```
#include <assert.h>
namespace sce {
    namespace Sled {
        namespace Assert {
            typedef FailureBehavior (*Handler)(
                const char *condition,
                const char *file,
                const int & line,
                const char *message
            );
        }
    }
}
```

Arguments

condition	The assert condition
file	The file in which the assert triggered
line	The line number of the assert
message	The message to display when the assert triggers

Return Values

FailureBehavior.

Description

Typedef for the assert failure handler.

Functions

assertHandler

Get assert handler.

Definition

```
#include <assert.h>
namespace sce {
    namespace Sled {
        namespace Assert {
            Handler &assertHandler();
        }
    }
}
```

Calling Conditions

Not multithread safe.

Return Values

`Assert` failure handler

Description

Get the assert handler.

See Also

`setAssertHandler`

reportFailure

Report failure.

Definition

```
#include <assert.h>
namespace sce {
    namespace Sled {
        namespace Assert {
            FailureBehavior reportFailure(
                const char *condition,
                const char *file,
                const int &line,
                const char *message,
                ...
            );
        }
    }
}
```

Arguments

condition	The assert condition
file	The file in which the assert triggered
line	The line number of the assert
message	Description of failure data format
...	Variable parameter list containing failure data

Return Values

FailureBehavior describing failure

Description

Report a failure.

See Also

`setAssertHandler`

setAssertHandler

Set assert handler.

Definition

```
#include <assert.h>
namespace sce {
    namespace Sled {
        namespace Assert {
            void setAssertHandler(
                Handler assertHandler
            );
        }
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

assertHandler	The assert handler to set

Return Values

None

Description

Set the assert handler.

See Also

`assertHandler`

sce::Sled::DebuggerMode namespace

Summary

sce::Sled::DebuggerMode

`DebuggerMode` enum namespace.

Definition

```
namespace DebuggerMode {}
```

Description

Namespace for scoping `DebuggerMode` enumeration.

Enumerated Types

Enum

Debugger mode enumeration.

Definition

```
#include <params.h>
namespace sce {
   namespace Sled {
      namespace DebuggerMode {
         enum Enum {
            kNormal,
            kStepInto,
            kStepOver,
            kStepOut,
            kStop
         };
      }
   }
}
```

Enumeration Values

Macro	Value	Description
`kNormal`	N/A	Normal.
`kStepInto`	N/A	Step into.
`kStepOver`	N/A	Step over.
`kStepOut`	N/A	Step out.
`kStop`	N/A	Stop.

Description

Debugger mode enum.

sce::Sled::Protocol namespace

Summary

sce::Sled::Protocol

`Protocol` enum namespace.

Definition

```
namespace Protocol {}
```

Description

Namespace for scoping `Protocol` enumeration.

Enumerated Types

Enum

Protocol enum.

Definition

```
#include <params.h>
namespace sce {
    namespace Sled {
        namespace Protocol {
            enum Enum {
                kTcp
            };
        }
    }
}
```

Enumeration Values

Macro	Value	Description
kTcp	N/A	Tcp.

Description

Protocol enumeration.

sce::Sled::SCMP namespace

Summary

sce::Sled::SCMP

SLED Control Message `Protocol` namespace.

Definition

```
namespace SCMP {}
```

Description

Namespace for SLED Control Message `Protocol` classes and functions.

Inner Classes, Structures, and Namespaces

Item	Description
sce::Sled::SCMP::Base	SLED Control Message `Protocol` base network message structure.
sce::Sled::SCMP::TypeCodes	Scoping `TypeCodes` enumeration namespace.

sce::Sled::SCMP::TypeCodes namespace

Summary

sce::Sled::SCMP::TypeCodes

Scoping TypeCodes enumeration namespace.

Definition

```
namespace TypeCodes {}
```

Description

Namespace for scoping TypeCodes enumeration.

Enumerated Types

Enum

Network messages type codes.

Definition

```
#include <scmp.h>
namespace sce {
    namespace Sled {
        namespace SCMP {
            namespace TypeCodes {
                enum Enum {
                    kBase = 0,
                    kBreakpointDetails = 1,
                    kBreakpointBegin = 2,
                    kBreakpointSync = 3,
                    kBreakpointEnd = 4,
                    kBreakpointContinue = 5,
                    kDisconnect = 6,
                    kHeartbeat = 8,
                    kSuccess = 9,
                    kFailure = 10,
                    kVersion = 11,
                    kDebugStart = 12,
                    kDebugStepInto = 13,
                    kDebugStepOver = 14,
                    kDebugStepOut = 15,
                    kDebugStop = 16,
                    kScriptCache = 17,
                    kAuthenticated = 18,
                    kReady = 20,
                    kPluginsReady = 21,
                    kFunctionInfo = 22,
                    kTTYBegin = 23,
                    kTTY = 24,
                    kTTYEnd = 25,
                    kDevCmd = 26,
                    kEditAndContinue = 27,
                    kEndianness = 28,
                    kProtocolDebugMark = 29
                };
            }
        }
    }
}
```

Enumeration Values

Macro	Value	Description
`kBase`	0	`Base` message.
`kBreakpointDetails`	1	Breakpoint details message.
`kBreakpointBegin`	2	Breakpoint begin message.
`kBreakpointSync`	3	Breakpoint sync message.
`kBreakpointEnd`	4	Breakpoint end message.

Macro	Value	Description
`kBreakpointContinue`	`5`	Breakpoint continue message.
`kDisconnect`	`6`	Disconnect message.
`kHeartbeat`	`8`	Heartbeat message.
`kSuccess`	`9`	Success message.
`kFailure`	`10`	Failure message.
`kVersion`	`11`	`Version` message.
`kDebugStart`	`12`	Debug start message.
`kDebugStepInto`	`13`	Debug step into message.
`kDebugStepOver`	`14`	Debug step over message.
`kDebugStepOut`	`15`	Debug step out message.
`kDebugStop`	`16`	Debug stop message.
`kScriptCache`	`17`	Script cache message.
`kAuthenticated`	`18`	Authenticated message.
`kReady`	`20`	Ready message.
`kPluginsReady`	`21`	Plugins ready message.
`kFunctionInfo`	`22`	Function information message.
`kTTYBegin`	`23`	TTY Begin message.
`kTTY`	`24`	TTY message.
`kTTYEnd`	`25`	TTY End message.
`kDevCmd`	`26`	Developer entered command message.
`kEditAndContinue`	`27`	Edit & Continue message.
`kEndianness`	`28`	Endianness message.
`kProtocolDebugMark`	`29`	`Protocol` Debug Mark message.

Description

Type codes for network messages.

sce::Sled::Utilities namespace

Summary

sce::Sled::Utilities

`Utilities` namespace.

Definition

```
namespace Utilities {}
```

Description

Namespace for `Utilities` classes and functions.

Function Summary

Function	Description
`appendString`	Append one string to another string.
`areStringsEqual`	Check whether or not strings equal.
`copyString`	Copy string to another string.
`copySubstring`	Copy string to another string.
`findFirstOf`	Find first occurrence of character in target string, starting from specified position in target string.
`findFirstOf`	Find first occurrence of string in target string, starting from specified position in target string.
`openFileCallback`	Get or set open file callback to use.
`openFileFinishCallback`	Get or set open file finish callback to use.

Type Definitions

FileCallback

Typedef used to signal when library needs file opened by client code.

Definition

```
#include <utilities.h>
namespace sce {
    namespace Sled {
        namespace Utilities {
            typedef const char * (*FileCallback)(
                const char *pszFilePath,
                void *pUserData
            );
        }
    }
}
```

Arguments

pszFilePath	Path to the file that the client code needs to open
pUserData	Optional user provided data

Return Values

File contents

Description

Typedef used to signal when the library needs a file opened by client code.

See Also

`FileFinishCallback`, `openFileCallback`, `openFileFinishCallback`

FileFinishCallback

Typedef used to signal when library is done using file contents that client code provided.

Definition

```
#include <utilities.h>
namespace sce {
   namespace Sled {
      namespace Utilities {
         typedef void (*FileFinishCallback)(
            const char *pszFilePath,
            void *pUserData
         );
      }
   }
}
```

Arguments

pszFilePath	Path to file that client code opened
pUserData	Optional user provided data

Return Values

None

Description

Typedef used to signal when the library is done using the file contents that the client code provided.

See Also

FileCallback, openFileCallback, openFileFinishCallback

Functions

appendString

Append one string to another string.

Definition

```
#include <utilities.h>
namespace sce {
    namespace Sled {
        namespace Utilities {
            void appendString(
                char *pszAppendTo,
                std::size_t len,
                const char *pszAppendFrom
            );
        }
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

pszAppendTo	Target string. Cannot be NULL and should already be initialized.
len	Maximum size of the target string buffer
pszAppendFrom	Source string. Can be NULL.

Return Values

None

Description

Append one string to another existing string.

See Also

`copyString`, `areStringsEqual`, `findFirstOf`, `copySubstring`

areStringsEqual

Check whether or not strings equal.

Definition

```
#include <utilities.h>
namespace sce {
    namespace Sled {
        namespace Utilities {
            bool areStringsEqual(
                const char *pszString1,
                const char *pszString2
            );
        }
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

pszString1	String to use in comparison
pszString2	String to use in comparison

Return Values

True if strings are equal; false if they are not

Description

Check whether or not two strings are equal.

See Also

`copyString`, `appendString`, `findFirstOf`, `copySubstring`

copyString

Copy string to another string.

Definition

```
#include <utilities.h>
namespace sce {
    namespace Sled {
        namespace Utilities {
            void copyString(
                char *pszCopyTo,
                std::size_t len,
                const char *pszCopyFrom
            );
        }
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

pszCopyTo	Target string. Cannot be NULL.
len	Maximum size of the target string buffer
pszCopyFrom	Source string. Can be NULL.

Return Values

None

Description

Copy one string to another string.

See Also

`appendString`, `areStringsEqual`, `findFirstOf`, `copySubstring`

copySubstring

Copy string to another string.

Definition

```
#include <utilities.h>
namespace sce {
    namespace Sled {
        namespace Utilities {
            void copySubstring(
                char *pszCopyTo,
                std::size_t len,
                const char *pszCopyFrom,
                const std::size_t &iStartPos,
                const std::size_t &iCopyLen
            );
        }
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

pszCopyTo	Target string. Cannot be NULL.
len	Maximum size of the target string buffer
pszCopyFrom	Source string. Cannot be NULL.
iStartPos	Starting position in the source string where the copy starts
iCopyLen	Number of characters to copy

Return Values

None

Description

Copy one string to another string,

See Also

copyString, appendString, areStringsEqual, findFirstOf

findFirstOf

Find first occurrence of character in target string, starting from specified position in target string.

Definition

```
#include <utilities.h>
namespace sce {
   namespace Sled {
      namespace Utilities {
         int findFirstOf(
            const char *pszSearch,
            char chWhat,
            int iStartPos
         );
      }
   }
}
```

Calling Conditions

Not multithread safe.

Arguments

pszSearch	The string to look in
chWhat	The character to look for
iStartPos	The position in `pszSearch` to start looking

Return Values

Position in `pszSearch` where searched-for character exists, or -1 if character was not found or if starting position is invalid

Description

Find the first occurrence of a character in a target string, starting from a specified position in that target string.

See Also

`copyString`, `appendString`, `areStringsEqual`, `copySubstring`

findFirstOf

Find first occurrence of string in target string, starting from specified position in target string.

Definition

```
#include <utilities.h>
namespace sce {
    namespace Sled {
        namespace Utilities {
            int findFirstOf(
                const char *pszSearch,
                const char *pszWhat,
                int iStartPos
            );
        }
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

pszSearch	The string to look in
pszWhat	The string to look for
iStartPos	The position in `pszSearch` to start looking

Return Values

Position in `pszSearch` where searched-for string starts, or -1 if string was not found or starting position is invalid.

Description

Find the first occurrence of a string in a target string, starting from a specified position in that target string.

See Also

`copyString`, `appendString`, `areStringsEqual`, `copySubstring`

openFileCallback

Get or set open file callback to use.

Definition

```
#include <utilities.h>
namespace sce {
    namespace Sled {
        namespace Utilities {
            FileCallback &openFileCallback();
        }
    }
}
```

Return Values

Open file callback to use

Description

Get or set the open file callback to use.

See Also

`FileCallback`, `FileFinishCallback`, `openFileFinishCallback`

openFileFinishCallback

Get or set open file finish callback to use.

Definition

```
#include <utilities.h>
namespace sce {
   namespace Sled {
      namespace Utilities {
         FileFinishCallback &openFileFinishCallback();
      }
   }
}
```

Return Values

Open file finish callback to use

Description

Get or set the open file finish callback to use.

See Also

`FileCallback`, `FileFinishCallback`, `openFileCallback`

sce::Sled::SledDebugger class

Summary

sce::Sled::SledDebugger

Class describing a SLED debugger instance.

Definition

```
#include <sleddebugger_class.h>
class SledDebugger {};
```

Description

Widely used class encapsulating the internals of a SLED debugger instance. Instantiate a `SledDebugger` from a `SledDebuggerConfig`.

This class is closed, and its internal data is not accessible.

The following are the main functions handling `SledDebugger`:

`sce::Sled::debuggerCreate()`: Create a `SledDebugger` instance.

`sce::Sled::debuggerRequiredMemory()`: Calculate the size in bytes required for a `SledDebugger` instance.

`sce::Sled::debuggerShutdown()`: Shut down a `SledDebugger` instance.

`sce::Sled::debuggerStartNetworking()`: Initialize networking and optionally block execution until a connection is made.

`sce::Sled::debuggerStopNetworking()`: Stop networking (disconnect SLED if connected).

`sce::Sled::debuggerUpdate()`: Poll sockets and process any incoming messages.

`sce::Sled::debuggerAddPlugin()`: Add plugin to `SledDebugger`.

`sce::Sled::debuggerRemovePlugin()`: Remove plugin from `SledDebugger`.

`sce::Sled::debuggerBreakpointReached`: Plugins call this function when they encounter a breakpoint.

For the full list of `SledDebugger` functions, see `sce::Sled`.

sce::Sled::SledDebuggerPlugin class

Summary

sce::Sled::SledDebuggerPlugin

Language plugin abstract base class.

Definition

```
#include <plugin.h>
class SledDebuggerPlugin {};
```

Description

Language plugin abstract base class. All language plugins must derive from this class.

Methods Summary

Methods	Description
getId	Get ID of plugin.
getName	Get name of plugin.
getVersion	Get version information of plugin.
SledDebuggerPlugin	Constructor.
~SledDebuggerPlugin	Destructor.

Constructors and Destructors

SledDebuggerPlugin

Constructor.

Definition

```
#include <plugin.h>
namespace sce {
    namespace Sled {
        class SledDebuggerPlugin {
            inline SledDebuggerPlugin();
        }
    }
}
```

Return Values

None

Description

`SledDebuggerPlugin` constructor.

~SledDebuggerPlugin

Destructor.

Definition

```
#include <plugin.h>
namespace sce {
    namespace Sled {
        class SledDebuggerPlugin {
            virtual inline ~SledDebuggerPlugin();
        }
    }
}
```

Return Values

None

Description

`SledDebuggerPlugin` destructor.

Public Instance Methods

getId

Get ID of plugin.

Definition

```
#include <plugin.h>
namespace sce {
    namespace Sled {
        class SledDebuggerPlugin {
            virtual uint16_t getId()=0 const;
        }
    }
}
```

Calling Conditions

Not multithread safe.

Return Values

A number greater than zero

Description

Get the ID of the plugin. The ID must be unique across all other language plugins. The ID of 0 (zero) is reserved for the `SledDebugger`.

See Also

`getName`, `getVersion`

getName

Get name of plugin.

Definition

```
#include <plugin.h>
namespace sce {
    namespace Sled {
        class SledDebuggerPlugin {
            virtual const char *getName()=0 const;
        }
    }
}
```

Calling Conditions

Not multithread safe.

Return Values

Name of the plugin

Description

Get the name of the plugin.

See Also

`getId`, `getVersion`

getVersion

Get version information of plugin.

Definition

```
#include <plugin.h>
namespace sce {
    namespace Sled {
        class SledDebuggerPlugin {
            virtual const Version getVersion()=0 const;
        }
    }
}
```

Calling Conditions

Not multithread safe.

Return Values

`Version` information of plugin

Description

Get the version information of the plugin.

See Also

`getId`, `getName`

sce::Sled::Timer class

Summary

sce::Sled::Timer

Multi-platform timer.

Definition

```
#include <timer.h>
class Timer {};
```

Description

Multi-platform timer class.

Methods Summary

Methods	Description
`create`	Create `Timer` instance.
`elapsed`	Get elapsed time of `Timer`.
`requiredMemory`	Calculate size in bytes required for `Timer` instance.
`reset`	Reset `Timer`.
`shutdown`	Shut down `Timer` instance.

Public Static Methods

create

Create `Timer` instance.

Definition

```
#include <timer.h>
namespace sce {
    namespace Sled {
        class Timer {
            static int32_t create(
                void *pLocation,
                Timer **ppTimer
            );
        }
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

pLocation	Location in memory in which to place the `Timer` instance. It needs to be as big as the value returned by `requiredMemory()`.
ppTimer	`Timer` instance that is created

Return Values

Value	Description
0	Success

Description

Create a `Timer` instance.

See Also

`requiredMemory`, `shutdown`, `reset`

requiredMemory

Calculate size in bytes required for `Timer` instance.

Definition

```
#include <timer.h>
namespace sce {
    namespace Sled {
        class Timer {
            static int32_t requiredMemory(
                std::size_t *iRequiredMemory
            );
        }
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

`iRequiredMemory`	The amount of memory that is needed for the `Timer` instance

Return Values

Value	Description
`0`	Success

Description

Calculate the size in bytes required for a `Timer` instance.

See Also

`create`

shutdown

Shut down `Timer` instance.

Definition

```
#include <timer.h>
namespace sce {
    namespace Sled {
        class Timer {
            static void shutdown(
                Timer *pTimer
            );
        }
    }
}
```

Calling Conditions

Not multithread safe.

Arguments

pTimer	`Timer` instance to shut down

Return Values

None

Description

Shut down a `Timer` instance.

See Also

`create`, `reset`

Public Instance Methods

elapsed

Get elapsed time of Timer.

Definition

```
#include <timer.h>
namespace sce {
    namespace Sled {
        class Timer {
            float elapsed() const;
        }
    }
}
```

Return Values

Elapsed time of Timer.

Description

Get the elapsed time of the Timer.

reset

Reset `Timer`.

Definition

```
#include <timer.h>
namespace sce {
    namespace Sled {
        class Timer {
            void reset();
        }
    }
}
```

Return Values

None

Description

Reset the `Timer`.

See Also

`shutdown`

sce::Sled::BreakpointParams struct

Summary

sce::Sled::BreakpointParams

Breakpoint params struct.

Definition

```
#include <params.h>
struct BreakpointParams {};
```

Description

Breakpoint parameters struct.

Fields

Public Static Fields

`static const uint16_t` *`kRelFilePathLen`*	Maximum length for RelFilePath.

Public Instance Fields

`uint32_t` *`lineNumber`*	Line number of the hit breakpoint.
`uint16_t` *`pluginId`*	Plugin that hit the breakpoint.
`char` *`relFilePath[kRelFilePathLen]`*	Relative path (from the asset directory) of the file that contains the breakpoint that was hit.

Methods Summary

Methods	Description
`BreakpointParams`	Constructor.
`BreakpointParams`	Constructor with parameters.
`BreakpointParams`	Copy constructor.
`operator=`	Assignment operator.

Constructors and Destructors

BreakpointParams

Constructor.

Definition

```
#include <params.h>
BreakpointParams();
```

Return Values

None

Description

`BreakpointParams` constructor.

BreakpointParams

Constructor with parameters.

Definition

```
#include <params.h>
BreakpointParams(
    uint16_t iPluginId,
    uint32_t iLineNumber,
    const char *pszRelFilePath
);
```

Arguments

iPluginId	ID of the plugin that hit the breakpoint
iLineNumber	Line number of the hit breakpoint
pszRelFilePath	Relative path (from the asset directory) of the file that contains the breakpoint that was hit

Return Values

None

Description

`BreakpointParams` constructor with parameters.

BreakpointParams

Copy constructor.

Definition

```
#include <params.h>
inline BreakpointParams(
    const BreakpointParams &rhs
);
```

Arguments

rhs	Item to copy from

Return Values

None

Description

`BreakpointParams` copy constructor.

Operator Methods

operator=

Assignment operator.

Definition

```
#include <params.h>
inline BreakpointParams &operator=(
    const BreakpointParams &rhs
);
```

Arguments

rhs	Item to copy from

Return Values

Assigned value

Description

`BreakpointParams` assignment operator.

sce::Sled::NetworkParams struct

Summary

sce::Sled::NetworkParams

Struct that describes details of network configuration structure.

Definition

```
#include <params.h>
struct NetworkParams {};
```

Description

Structure that describes details of the network configuration structure. The `NetworkParams` structure defines which network protocol to use: TCP, which port to use (if the protocol is TCP), and whether or not to wait for SLED to connect before continuing execution from `debuggerStartNetworking()`.

Fields

Public Instance Fields

`bool` *`blockUntilConnect`*	Whether or not to block program execution until SLED connects.
`uint16_t` *`port`*	Port to use. Relevant only if the protocol is TCP.
`Protocol::Enum` *`protocol`*	Network protocol to use: TCP.

Methods Summary

Methods	Description
`NetworkParams`	Constructor.
`NetworkParams`	Copy constructor.
`operator=`	Assignment operator.
`setup`	Setup function.

Constructors and Destructors

NetworkParams

Constructor.

Definition

```
#include <params.h>
NetworkParams();
```

Return Values

None

Description

`NetworkParams` constructor.

NetworkParams

Copy constructor.

Definition

```
#include <params.h>
inline NetworkParams(
    const NetworkParams &rhs
);
```

Arguments

rhs	Item to copy from

Return Values

None

Description

`NetworkParams` copy constructor.

Operator Methods

operator=

Assignment operator.

Definition

```
#include <params.h>
inline NetworkParams &operator=(
    const NetworkParams &rhs
);
```

Arguments

rhs	Item to copy from

Return Values

Assigned value

Description

`NetworkParams` assignment operator.

Public Instance Methods

setup

Setup function.

Definition

```
#include <params.h>
void setup(
    Protocol::Enum kProtocol,
    uint16_t iPort,
    bool bBlockUntilConnect
);
```

Calling Conditions

Not multithread safe.

Arguments

kProtocol	Network protocol to use
iPort	Network port to use
bBlockUntilConnect	Whether or not to block program execution until SLED connects

Return Values

None

Description

`NetworkParams` setup function.

sce::Sled::SCMP::Base struct

Summary

sce::Sled::SCMP::Base

SLED Control Message Protocol base network message structure.

Definition

```
#include <scmp.h>
struct Base {};
```

Description

SLED Control Message Protocol base network message structure. All network messages must derive from Base.

Fields

Public Static Fields

static const int *kSizeOfBase*	Size of the SCMP::Base structure in bytes (8)
static const int *kSizeOfdouble*	Size of a double in bytes (8)
static const int *kSizeOffloat*	Size of a float in bytes (4)
static const int *kSizeOfint16_t*	Size of a int16_t in bytes (2)
static const int *kSizeOfint32_t*	Size of a int32_t in bytes (4)
static const int *kSizeOfint64_t*	Size of a int64_t in bytes (8)
static const int *kSizeOfuint16_t*	Size of a uint16_t in bytes (2)
static const int *kSizeOfuint32_t*	Size of a uint32_t in bytes (4)
static const int *kSizeOfuint64_t*	Size of a uint64_t in bytes (8)
static const int *kSizeOfuint8_t*	Size of a uint8_t in bytes (1)
static const int *kStringLen*	Default string length used in SCMP messages that contain strings.

Public Instance Fields

int32_t *length*	Length of the message in bytes.
uint16_t *pluginId*	Plugin that this message should be sent to.
uint16_t *typeCode*	Property that identifies what type of message this is.

Methods Summary

Methods	Description
isBreakpoint	Convenience method to see if message represents breakpoint command.
isDebug	Convenience method to see if message represents debug command.
isReady	Convenience method to see if message represents ready command.

Public Instance Methods

isBreakpoint

Convenience method to see if message represents breakpoint command.

Definition

```
#include <scmp.h>
inline bool isBreakpoint() const;
```

Calling Conditions

Not multithread safe.

Return Values

True if breakpoint command; false if not

Description

Convenience method to see if message represents a breakpoint command.

See Also

`isDebug`, `isReady`,

isDebug

Convenience method to see if message represents debug command.

Definition

```
#include <scmp.h>
inline bool isDebug() const;
```

Calling Conditions

Not multithread safe.

Return Values

True if debug command; false if not

Description

Convenience method to see if message represents a debug command.

See Also

`isBreakpoint`, `isReady`,

isReady

Convenience method to see if message represents ready command.

Definition

```
#include <scmp.h>
inline bool isReady() const;
```

Calling Conditions

Not multithread safe.

Return Values

True if ready command; false if not

Description

Convenience method to see if message represents a ready command.

See Also

`isBreakpoint`, `isDebug`

sce::Sled::SledDebuggerConfig struct

Summary

sce::Sled::SledDebuggerConfig

Structure describing details of `SledDebugger` instance.

Definition

```
#include <params.h>
struct SledDebuggerConfig {};
```

Description

The `SledDebuggerConfig` structure describes the details of a `SledDebugger` instance.

Fields

Public Instance Fields

`uint16_t` *`maxPlugins`*	Maximum number of plugins that the `SledDebugger` will manage.
`uint32_t` *`maxRecvBufferSize`*	Maximum size of the receive buffer (1024 recommended at a minimum)
`uint16_t` *`maxScriptCacheEntries`*	Maximum number of files that the script cache will hold.
`uint16_t` *`maxScriptCacheEntryLen`*	Maximum string length of a script cache file entry.
`uint32_t` *`maxSendBufferSize`*	Maximum size of the send buffer (1024 recommended at a minimum)
`NetworkParams` *`net`*	Network settings.

Methods Summary

Methods	Description
`operator=`	Assignment operator.
`SledDebuggerConfig`	Constructor.
`SledDebuggerConfig`	Copy constructor.

Constructors and Destructors

SledDebuggerConfig

Constructor.

Definition

```
#include <params.h>
inline SledDebuggerConfig();
```

Return Values

None

Description

`SledDebuggerConfig` constructor.

SledDebuggerConfig

Copy constructor.

Definition

```
#include <params.h>
inline SledDebuggerConfig(
    const SledDebuggerConfig &rhs
);
```

Arguments

rhs	Item to copy from

Return Values

None

Description

`SledDebuggerConfig` copy constructor.

Operator Methods

operator=

Assignment operator.

Definition

```
#include <params.h>
inline SledDebuggerConfig &operator=(
    const SledDebuggerConfig &rhs
);
```

Arguments

rhs	Item to copy from

Return Values

Assigned value

Description

`SledDebuggerConfig` assignment operator.

sce::Sled::Version struct

Summary

sce::Sled::Version

`Version` detail.

Definition

```
#include <params.h>
struct Version {};
```

Description

`Version` detail information.

Fields

Public Instance Fields

`uint16_t` *`majorNum`*	Major version number.
`uint16_t` *`minorNum`*	Minor version number.
`uint16_t` *`revisionNum`*	Revision version number.

Methods Summary

Methods	Description
`operator=`	Assignment operator.
`Version`	Constructor.
`Version`	Constructor with parameters.
`Version`	Copy constructor.

Constructors and Destructors

Version

Constructor.

Definition

```
#include <params.h>
inline Version();
```

Return Values

None

Description

`Version` constructor.

Version

Constructor with parameters.

Definition

```
#include <params.h>
inline Version(
    uint16_t iMajor,
    uint16_t iMinor,
    uint16_t iRevision
);
```

Arguments

iMajor	Major version number
iMinor	Minor version number
iRevision	Revision version number

Return Values

None

Description

`Version` constructor with parameters.

Version

Copy constructor.

Definition

```
#include <params.h>
inline Version(
   const Version &rhs
);
```

Arguments

rhs	Item to copy from

Return Values

None

Description

`Version` copy constructor.

Operator Methods

operator=

Assignment operator.

Definition

```
#include <params.h>
inline Version &operator=(
    const Version &rhs
);
```

Arguments

rhs	Item to copy from

Return Values

Assigned value

Description

`Version` assignment operator.