nlib2

Generated by Doxygen 1.9.1

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

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

Module Documentation

5.1 Modflow: a graph based modular interface

Classes

- class nlib2::Channel

  Defines a channel that each module can create and to which other modules can connect.
- class nlib2::ResourceManager

  Allow sharing resources of generic types among modules.
- class nlib2::Module

  This is the core node of a Modflow graph. Inherit this class to define the main computation units to happen in this module. Each module can define new Channels to which it can emit output events after computation is done, and request channels to connect to when data is transmitted on such channels.
- class nlib2::Sources

  Sources are a particular Module whose channels are declared by the parent object, which emits signals on such channels externally. The Modflow object will automatically load a module, available externally via Modflow::sources.
- class nlib2::SerializedSlot

  Type erasure wrapper for storing generic functions. For internal use.
- class nlib2::Modflow

  This is the main class that handles the call flow between modules. First, you will need to derive this class and override loadModules, calling loadModule with the type of each module you want to load as template parameter. Then:

5.1.1 Detailed Description

Chapter 6

Class Documentation

6.1 nlib2::Channel Class Reference

Defines a channel that each module can create and to which other modules can connect.

`#include <nl_modflow.h>`

Public Member Functions

- template<typename ... typeids>
  Channel (ChannelId id, const std::string &name, const Module *owner, bool isSink, const typeids *...ids)
  Create a new channel given:
- **Channel** (const Channel &)=default
- ChannelId id () const
  Get unique identifier of the channel.
- const std::string & name () const
  Get name of the channel.
- template<typename ... T>
  bool checkType () const
  Check whether supplied type is compatible with Channel type.
- std::vector< std::type_index > types () const
  Return vector of the types of the channel.
- std::string ownerName () const
  Return name of owner module.
- bool checkOwnership (const Module *caller) const
  Verify that a module is the effective owner of the channel.

6.1.1 Detailed Description

Defines a channel that each module can create and to which other modules can connect.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Channel()

```
template<typename ...  typeids>
nlib2::Channel::Channel (
            ChannelId id,
            const std::string & name,
            const Module * owner,
            bool isSink,
            const typeids *...  ids )
```

Create a new channel given:

Parameters

id	Unique identifier of the channel
name	Unique name that can resolve to the id from a ModFlow handler
ids	Channel type(s) identifier: only slots with same type(s) as channel can be connected
owner	Pointer to owner module: only owner can emit events on channels has itself created
isSink	Sink channels are connected to Parent methods, external to modflow

6.1.3 Member Function Documentation

6.1.3.1 checkType()

```
template<typename ...  ChannelTs>
bool nlib2::Channel::checkType
```

Check whether supplied type is compatible with Channel type.

Template Parameters

T	Type(s) to check Channel-type with

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

- include/nlib2/nl_modflow.h
- include/nlib2/nl_modflow_impl.hpp

6.2 nlib2::Event Class Reference

Public Member Functions

- **Event** (const Module ∗module, const Channel ∗channel)
- **Event** (const Event::SharedPtr &parent, const Module ∗module, const Channel ∗channel)

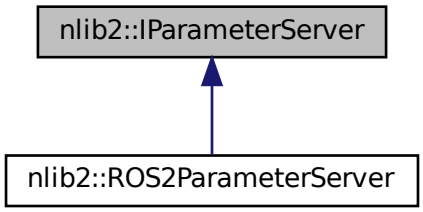
- Event::SharedPtr **branch** (const Module *module, const Channel *channel)
- bool **channelInAncestors** (const std::string &name) const
- bool **moduleInAncestors** (const std::string &name) const
- const std::string & **moduleName** () const
- const std::string & **channelName** () const
- int **depth** () const

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

- include/nlib2/nl_modflow.h
- include/nlib2/nl_modflow_impl.hpp

6.3 nlib2::IParameterServer Struct Reference

Inheritance diagram for nlib2::IParameterServer:



Public Member Functions

- template<typename ParameterT >
  void **declareParameter** (const std::string &name, const std::optional< ParameterT > &defaultValue)
- template<typename ParameterT >
  ParameterT **getParameter** (const std::string &name)

Protected Member Functions

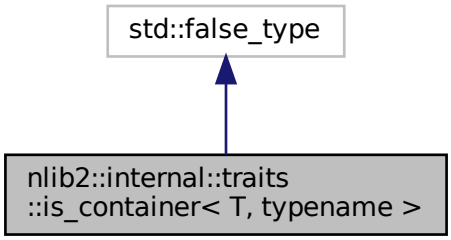
- virtual void **declareParameter** (const std::string &name, const std::optional< std::any > &defaultValue, const std::type_index &type)=0
- virtual std::any **getParameter** (const std::string &name, const std::type_index &type)=0

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

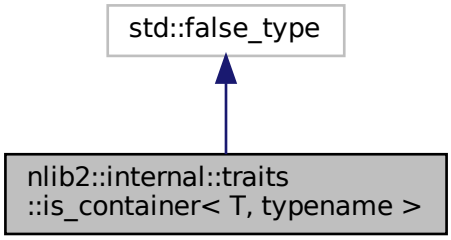
- include/nlib2/nl_params.h
- include/nlib2/nl_params_impl.hpp

6.4 nlib2::internal::traits::is_container< T, typename > Struct Template Reference

Inheritance diagram for nlib2::internal::traits::is_container< T, typename >:



Collaboration diagram for nlib2::internal::traits::is_container< T, typename >:

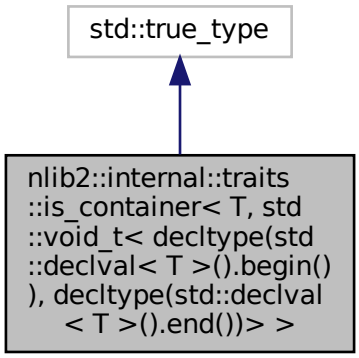


The documentation for this struct was generated from the following file:

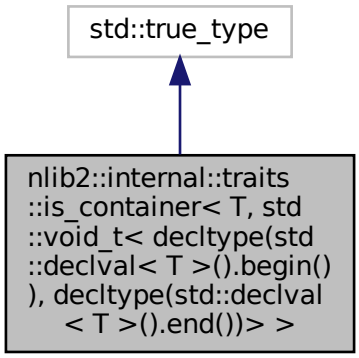
- include/nlib2/nl_node_impl.hpp

6.5 nlib2::internal::traits::is_container< T, std::void_t< decltype(std::declval< T >().begin()), decltype(std::declval< T >().end())> > Struct Template Reference

Inheritance diagram for nlib2::internal::traits::is_container< T, std::void_t< decltype(std::declval< T >().begin()), decltype(std::declval< T >().end())> >:



Collaboration diagram for nlib2::internal::traits::is_container< T, std::void_t< decltype(std::declval< T >().begin()), decltype(std::declval< T >().end())> >:



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

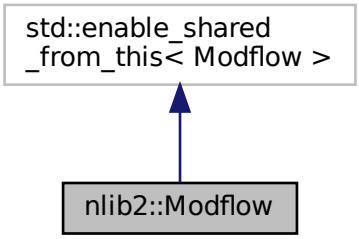
- include/nlib2/nl_node_impl.hpp

6.6 nlib2::Modflow Class Reference

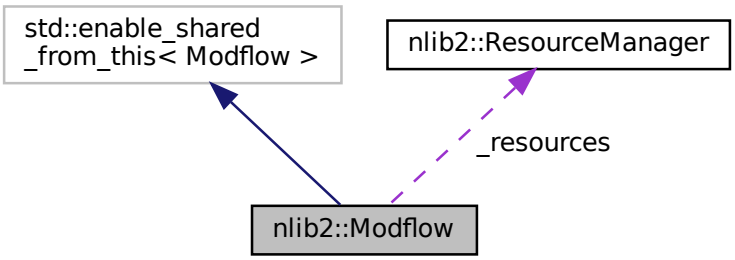
This is the main class that handles the call flow between modules. First, you will need to derive this class and override loadModules, calling loadModule with the type of each module you want to load as template parameter. Then:

```
#include <nl_modflow.h>
```

Inheritance diagram for nlib2::Modflow:



Collaboration diagram for nlib2::Modflow:



Public Member Functions

- void init (const NlParams::SharedPtr &nlParams=nullptr)

  This is the first function to be called. It loads sources and sink modules. After that, it calls the virtual function loadModules, that each Module derived class must override, that is supposed to load all the modules, in the provided order. `nlParams` is the overall parameter server. If initialized with nullptr all parameter-related methods call will be disabled.
- void finalize ()

  To be called after the declaration of sources and sinks. For each module in the same order as specified in setupNetwork, it calls Module::initParams (if a NlParams parameter server has been supplied) and then Module::setupNetwork.

- Sources::SharedPtr sources ()

  Get a pointer to the source module object.
- Sinks::SharedPtr sinks ()

  Get a pointer to the sinks module object.
- void onParameterChange (const std::string &fullPath, const NlParams::SharedPtr &nlParams)

  From the full parameter path, it extracts, the module name, finds the module in the module list, and calls Module::onParameterChange with the rest of the parameter path.

Static Public Member Functions

- template<class DerivedModflow >
  static std::enable_if_t< std::is_base_of_v< Modflow, DerivedModflow >, Modflow::SharedPtr > create ()

  A Modflow object must be created via this method, which creates a shared pointer, instantiated with the derived class type supplied as template parameter.

Protected Member Functions

- template<class DerivedModule , typename ... Args>
  DerivedModule::SharedPtr loadModule (Args &&...args)

  Construct a dynamically allocated object of type `DerivedModule`, as shared pointer, and stores into Modflow's modules list.
- virtual void loadModules ()=0

  Method to be overridden to specify the modules to be loaded via loadModule.
- template<typename ... ChannelTs>
  Channel createChannel (const std::string &name, const Module *owner, bool isSink=false)

  Declare a new channel of types ...ChannelTs. The channel is owned by module `owner`, that is the only module that can emit events on this channel.
- Channel resolveChannel (const std::string &name)

  Get a Channel object given its name.
- template<typename ReturnT , typename ... ChannelTs>
  void createConnection (const Channel &channel, const Slot< ReturnT, ChannelTs... > &slot, const std::string &name)

  Create a connection named `name` from `channel` to a `slot` function.
- template<typename ReturnT , typename ... ChannelTs>
  std::enable_if_t< std::is_same_v< ReturnT, void >, ReturnT > emit (const Channel &channel, const Module *caller, ChannelTs &&...args)

  Emit a signal on `channel`. This will call the slot methods associated to the channel of each module, supplying ...args.
- template<typename ReturnT , typename ... ChannelTs>
  std::enable_if_t<!std::is_same_v< ReturnT, void >, ReturnT > **emit** (const Channel &channel, const Module *caller, ChannelTs &&...args)
- template<typename ReturnT , typename ... ChannelTs>
  ReturnT **emit** (const std::string &channelname, const Module *caller, ChannelTs &&...args)

Protected Attributes

- NlParams::SharedPtr **_nlParams**
- ResourceManager **_resources**

Friends

- class **Module**
- class **Sources**
- class **Sinks**

6.6.1 Detailed Description

This is the main class that handles the call flow between modules. First, you will need to derive this class and override loadModules, calling loadModule with the type of each module you want to load as template parameter. Then:

- Create Modflow::SharedPtr with Modflow::create, with your Modflow derived class template
- Initialize the modflow object with init, which will initialize sources and sinks modules
- Declare source channels via sources() -> Sources::declareSourceChannel, to create entry point channels
- Declare sink channels via sinks() -> Sinks::declareSink, to connect exit points to external callbacks
- Finalize the Modflow initialization via finalize (), which will configure each module by colling Module::initParams and Module::setupNetwork, that each module shall override. The initialization of the modules is done in the same order they have been loaded.

After initialization, sources can be triggered by calling sources() -> Sources::callSource, which will emit signals on regular channels as defined by Sources::declareSourceChannels. Modules connected to such channels will receive a function call on its associated slot, which can in turn emit other signals on other channels, which can then be connected to other modules or to sinks. Differently from regular channels, sink channels are connected to an external callback that acts as exit point of Modflow.

6.6.2 Member Function Documentation

6.6.2.1 createChannel()

```
template<typename ...  ChannelTs>
Channel nlib2::Modflow::createChannel (
             const std::string & name,
             const Module * owner,
             bool isSink = false )  [protected]
```

Declare a new channel of types ...ChannelTs. The channel is owned by module `owner`, that is the only module that can emit events on this channel.

Returns

A new Channel object with the created channel information

6.6.2.2 createConnection()

```
template<typename ReturnT , typename ...  ChannelTs>
void nlib2::Modflow::createConnection (
             const Channel & channel,
             const Slot< ReturnT, ChannelTs...  > & slot,
             const std::string & name )  [protected]
```

Create a connection named `name` from `channel` to a `slot` function.

Note

Do not call this method directly. Use Module::requestConnection or Sinks::declareSink

6.6.2.3 emit()

```
template<typename ReturnT , typename ...  ChannelTs>
std::enable_if_t< std::is_same_v< ReturnT, void >, ReturnT > nlib2::Modflow::emit (
             const Channel & channel,
             const Module * caller,
             ChannelTs &&...  args )  [protected]
```

Emit a signal on `channel`. This will call the slot methods associated to the channel of each module, supplying ...args.

Returns

If ReturnT is not void, forwards the return value of `slot`. In this case only one slot can be connected per channel

6.6.2.4 loadModule()

```
template<class DerivedModule , typename ...  Args>
DerivedModule::SharedPtr nlib2::Modflow::loadModule (
             Args &&...  args )  [protected]
```

Construct a dynamically allocated object of type `DerivedModule`, as shared pointer, and stores into Modflow's modules list.

Returns

Pointer to created module

6.6.2.5 resolveChannel()

```
Channel nlib2::Modflow::resolveChannel (
            const std::string & name )  [inline], [protected]
```

Get a Channel object given its name.

Complexity Logarithmic in the number of channels

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

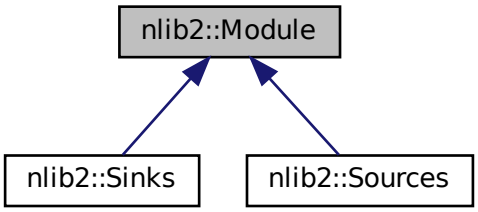
- include/nlib2/nl_modflow.h
- include/nlib2/nl_modflow_impl.hpp

6.7 nlib2::Module Class Reference

This is the core node of a Modflow graph. Inherit this class to define the main computation units to happen in this module. Each module can define new Channels to which it can emit output events after computation is done, and request channels to connect to when data is transmitted on such channels.

`#include <nl_modflow.h>`

Inheritance diagram for nlib2::Module:



Public Member Functions

- const std::string & name () const

  Get module name.
- Event::SharedPtr lastEvent () const

  For internal use.
- virtual void initParams (const NlParams::SharedPtr &nlParams)

  Override this method to (optionally) initialize the parameters Parameters are supplied during the execution of Modflow::finalize. The nlParams object is already scoped with the parameters accessed by e.g. "moduleX" must be defined as moduleX.params.
- virtual void setupNetwork ()=0

  Implement this method to handle the connections. Create channels (see Module::createChannel) for outbound connections and request connections from inbound channels (see Module::requestConnection)
- bool isEnabled () const

  Retruns true when all "enabling channels" have been triggered at least once (see requestEnablingChannel)
- void onParamChange (const std::string &name, const NlParams::SharedPtr &nlParams)

  Should only be called by the Modflow object. If _automaticParamUpdate is true, it re-execute initParams, and every parameter of the module is re-read. Otherwise, it calls your implementation of Module::updateParam.

Protected Member Functions

- Module (const std::shared_ptr< Modflow > &modflow, const std::string &name, bool automaticParam↩ Update=true)

  The module can only be created via Modflow::loadModule, which handles its allocation as shared pointer. For each module, the constructor should have a Modflow shared pointer as first argument, and then the optional other argumetns, and forward the pointer to the parent's constructor.
- virtual void updateParam (const std::string &name, const NlParams::SharedPtr &nlParams)

  This function is called after onParamChange when _automaticParamUpdate is false.
- template<typename ... ChannelTs, typename DerivedModule , typename ReturnT > std::enable_if_t< std::is_base_of_v< Module, DerivedModule > > requestConnection (const std::string &channelName, ReturnT(DerivedModule::∗slot)(ChannelTs ...))

  Bind signals emitted on a given channel name to a member function `slot` of the derived module The channel types are automatically deduced from the slot arguments.
- void requestEnablingChannel (const std::string &channelName)

  Request a channel to be enabling of the module. Until all enabling channels have been triggered at least once, all other inbound connections are disabled.
- void **requestEnablingChannel** (const Channel &channelName)
- template<typename ... ChannelTs> Channel createChannel (const std::string &name)

  Create a standard channel of types `ChannelTs` named `name`, owned by this module (.
- template<typename ... ChannelTs> Channel requireSink (const std::string &sinkName)

  Ensures the parent object has declared a sink named `sinkName` with types `ChannelTs`.
- template<typename ... ChannelTs> void emit (const Channel &channel, ChannelTs &&...value)

  Emit a signal on `channel`. All slot of every module connected to the specified channel in will be called with the data `value`.
- template<typename ... ChannelTs> void emit (const std::string &channel, ChannelTs &&...value)

  As.
- template<typename ReturnT , typename ... ChannelTs> ReturnT callService (const Channel &channel, ChannelTs &&...value)

  Emit a signal on a `channel` that has non-void return value. The principle is the same as.
- template<typename ReturnT , typename ... ChannelTs> ReturnT **callService** (const std::string &channel, ChannelTs &&...value)
- const ResourceManager & resources () const

  Handle to centralized Modflow.
- ResourceManager & **resources** ()

Protected Attributes

- std::shared_ptr< Modflow > **_modflow**
- Event::SharedPtr **_lastEvent**

6.7.1 Detailed Description

This is the core node of a Modflow graph. Inherit this class to define the main computation units to happen in this module. Each module can define new Channels to which it can emit output events after computation is done, and request channels to connect to when data is transmitted on such channels.

6.7.2 Member Function Documentation

6.7.2.1 callService()

```
template<typename ReturnT , typename ...  ChannelTs>
ReturnT nlib2::Module::callService (
            const Channel & channel,
            ChannelTs &&...  value )  [protected]
```

Emit a signal on a `channel` that has non-void return value. The principle is the same as.

See also

Module::emit, but only one slot can be connected (

Module::requestConnection will fail when trying to connect to a channel to which another slot is already connected)

6.7.2.2 createChannel()

```
template<typename ...  ChannelTs>
Channel nlib2::Module::createChannel (
            const std::string & name )  [protected]
```

Create a standard channel of types `ChannelTs` named `name`, owned by this module (.

See also

Modflow::createChannel for details)

6.7.2.3 emit()

```
template<typename ...  ChannelTs>
void nlib2::Module::emit (
            const std::string & channel,
            ChannelTs &&...  value )  [protected]
```

As.

See also

Module::emit (const Channel &channel,...) but it resolves the channel name (small overhead for name resolution)

6.7.2.4 initParams()

```
virtual void nlib2::Module::initParams (
            const NlParams::SharedPtr & nlParams )  [inline], [virtual]
```

Override this method to (optionally) initialize the parameters Parameters are supplied during the execution of Modflow::finalize. The nlParams object is already scoped with the parameters accessed by e.g. "moduleX" must be defined as moduleX.params.

Parameters

nlParams	Obtained as nlParams->derived(module->name())

Reimplemented in nlib2::Sinks, and nlib2::Sources.

6.7.2.5 onParamChange()

```
void nlib2::Module::onParamChange (
            const std::string & name,
            const NlParams::SharedPtr & nlParams )  [inline]
```

Should only be called by the Modflow object. If _automaticParamUpdate is true, it re-execute initParams, and every parameter of the module is re-read. Otherwise, it calls your implementation of Module::updateParam.

Parameters

name	Scoped parameter name
nlParams	NlParam object pointer

6.7.2.6 requestConnection()

```
template<typename ...  ChannelTs, typename DerivedModule , typename ReturnT >
std::enable_if_t< std::is_base_of_v< Module, DerivedModule > > nlib2::Module::requestConnection
(
            const std::string & channelName,
            ReturnT(DerivedModule::*)(ChannelTs ...)  slot )  [protected]
```

Bind signals emitted on a given channel name to a member function `slot` of the derived module The channel types are automatically deduced from the slot arguments.

See also

Modflow::createConnection

6.7.2.7 resources()

```
const ResourceManager & nlib2::Module::resources ( ) const  [inline], [protected]
```

Handle to centralized Modflow.

See also

ResourceManager

6.7.2.8 updateParam()

```
virtual void nlib2::Module::updateParam (
            const std::string & name,
            const NlParams::SharedPtr & nlParams )  [inline], [protected], [virtual]
```

This function is called after onParamChange when _automaticParamUpdate is false.

Parameters

name	
nlParams	

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

- include/nlib2/nl_modflow.h
- include/nlib2/nl_modflow_impl.hpp

6.8 nlib2::NlNode< Derived > Class Template Reference

Public Member Functions

- **NlNode** (int argc, char **argv, const std::string &name, const rclcpp::NodeOptions &options=rclcpp::Node↩ Options())
- int **spin** ()
- const std::string & **name** () const

Protected Member Functions

- void **initParams** ()
- void **initROS** ()
- template<typename MessageT >
  void **addSubscription** (const std::string &name, const std::string &topic, const rclcpp::QoS &qos, void(Derived::*callback)(MessageT))
- template<typename MessageT >
  void **addSubscription** (const std::string &name, const rclcpp::QoS &qos, void(Derived::*callback)(MessageT))
- template<typename MessageT >
  void **addPublisher** (const std::string &name, const std::string &topic, const rclcpp::QoS &qos)
- template<typename MessageT >
  void **addPublisher** (const std::string &name, const rclcpp::QoS &qos)
- template<typename MessageT >
  void **publish** (const std::string &name, const MessageT &msg)
- void **onParamChange** (const std::string &paramName)
- void **onClock** ()

Protected Attributes

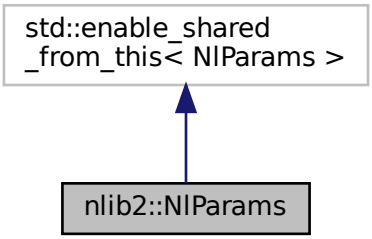
- std::string **_name**
- rclcpp::Node::SharedPtr **_node**
- ROS2ParameterServer::SharedPtr **_parameterServer**
- NlParams::SharedPtr **_nlParams**
- std::unordered_map< std::string, rclcpp::PublisherBase::SharedPtr > **_publishers**
- std::unordered_map< std::string, rclcpp::SubscriptionBase::SharedPtr > **_subscriptions**
- int **_argc**
- char ** **_argv**

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

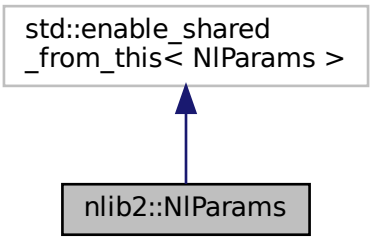
- include/nlib2/nl_node.h
- include/nlib2/nl_node_impl.hpp

6.9 nlib2::NlParams Class Reference

Inheritance diagram for nlib2::NlParams:



Collaboration diagram for nlib2::NlParams:



Public Member Functions

- template<typename ParameterT >
  ParameterT **get** (const std::string &name)
- template<typename ParameterT >
  void **declare** (const std::string &name, const std::optional< ParameterT > &defaultValue=std::nullopt)
- template<typename ParameterT >
  ParameterT **declareAndGet** (const std::string &name, const std::optional< ParameterT > &default↩
  Value=std::nullopt)
- NlParams::SharedPtr **derive** (const std::string &childName)
- std::string **fullName** (const std::string &childName)

Static Public Member Functions

- static NlParams::SharedPtr **initialize** (const IParameterServer::SharedPtr &parameterServer)

Protected Attributes

- NlParams::SharedPtr **_parent**
- std::string **_paramNamespace**
- IParameterServer::SharedPtr **_parameterServer**

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

- include/nlib2/nl_params.h
- include/nlib2/nl_params_impl.hpp

6.10 nlib2::ResourceManager Class Reference

Allow sharing resources of generic types among modules.

`#include <nl_modflow.h>`

Public Member Functions

- template<typename T , typename ... Args>
  void create (const std::string &name, Args &&...args)

  Create a new resource on heap, stored with type erasure.
- template<typename T >
  std::shared_ptr< T > get (const std::string &name)

  Get an existing resource of type T.
- template<typename ResourceT >
  std::shared_ptr< ResourceT > **get** (const std::string &name)

6.10.1 Detailed Description

Allow sharing resources of generic types among modules.

6.10.2 Member Function Documentation

6.10.2.1 create()

```
template<typename ResourceT , typename ...  Args>
void nlib2::ResourceManager::create (
             const std::string & name,
             Args &&...  args )
```

Create a new resource on heap, stored with type erasure.

Parameters

name	Unique name to access resource
args	Resource constructor arguments

6.10.2.2 get()

```
template<typename T >
std::shared_ptr<T> nlib2::ResourceManager::get (
             const std::string & name )
```

Get an existing resource of type T.

Parameters

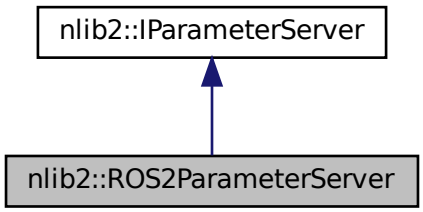
name	Resource unique name

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

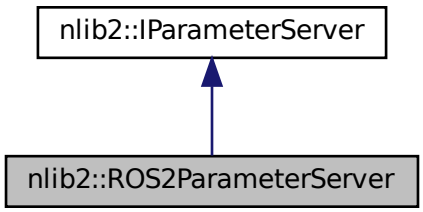
- include/nlib2/nl_modflow.h
- include/nlib2/nl_modflow_impl.hpp

6.11 nlib2::ROS2ParameterServer Class Reference

Inheritance diagram for nlib2::ROS2ParameterServer:



Collaboration diagram for nlib2::ROS2ParameterServer:



Public Member Functions

- **ROS2ParameterServer** (const rclcpp::Node::SharedPtr &node)

Protected Member Functions

- void **declareParameter** (const std::string &name, const std::optional< std::any > &defaultValuie, const std::type_index &type)
- std::any **getParameter** (const std::string &name, const std::type_index &type)

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

- include/nlib2/nl_node.h
- include/nlib2/nl_node_impl.hpp

6.12 nlib2::RosConfiguration Class Reference

Public Member Functions

- **RosConfiguration** (const rclcpp::Node::SharedPtr &node)
- void **setup** (bool centralized, const std::string &centralizedNodeName="")
- std::string **getTopic** (const std::string &name, bool sub)

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

- include/nlib2/nl_node.h
- include/nlib2/nl_node_impl.hpp

6.13 nlib2::SerializedSlot Class Reference

Type erasure wrapper for storing generic functions. For internal use.

`#include <nl_modflow.h>`

Public Member Functions

- template<typename ReturnT , typename ... SlotTs>
  SerializedSlot (const Slot< ReturnT, SlotTs... > &slt, const Channel &channel, const std::string &slotName, std::enable_if_t<(sizeof ...(SlotTs)<=1)> *=nullptr)
- template<typename ReturnT , typename ... SlotTs>
  SerializedSlot (const Slot< ReturnT, SlotTs... > &slt, const Channel &channel, const std::string &slotName, std::enable_if_t<(sizeof ...(SlotTs) > 1)> *=nullptr)
- template<typename ReturnT , typename SlotT >
  std::enable_if_t< std::is_same_v< ReturnT, void >, ReturnT > **invoke** (const Event::SharedPtr &event, SlotT &&arg) const
- template<typename ReturnT , typename ... SlotTs>
  std::enable_if_t<(sizeof ...(SlotTs) > 1) &&std::is_same_v< ReturnT, void >, ReturnT > **invoke** (const Event::SharedPtr &event, SlotTs &&...args) const
- template<typename ReturnT >
  std::enable_if_t< std::is_same_v< ReturnT, void >, ReturnT > **invoke** (const Event::SharedPtr &event) const
- template<typename ReturnT , typename SlotT >
  std::enable_if_t<!std::is_same_v< ReturnT, void >, ReturnT > **invoke** (const Event::SharedPtr &event, SlotT &&arg) const
- template<typename ReturnT , typename ... SlotTs>
  std::enable_if_t<(sizeof ...(SlotTs) > 1) &&!std::is_same_v< ReturnT, void >, ReturnT > **invoke** (const Event::SharedPtr &event, SlotTs &&...args) const
- template<typename ReturnT >
  std::enable_if_t<!std::is_same_v< ReturnT, void >, ReturnT > **invoke** (const Event::SharedPtr &event) const
- const std::string & **name** () const
- template<typename ReturnT , typename ... SlotTypes>
  SerializedSlot (const Slot< ReturnT, SlotTypes... > &slt, const Channel &channel, const std::string &slot↩Name, std::enable_if_t<(sizeof ...(SlotTypes)<=1)> *)

- template<typename ReturnT , typename ... SlotTypes>
  SerializedSlot (const Slot< ReturnT, SlotTypes... > &slt, const Channel &channel, const std::string &slotName, std::enable_if_t<(sizeof ...(SlotTypes) > 1)> *)
- template<typename ... SlotTypes, std::size_t ... is>
  std::enable_if_t<(sizeof ...(SlotTypes) > 1)> **serialize** (const Slot< void, SlotTypes... > &slot, std::index_sequence< is... >)
- template<typename ReturnT , typename ... SlotTypes, std::size_t ... is>
  std::enable_if_t<(sizeof ...(SlotTypes) > 1) &&!std::is_same_v< ReturnT, void > > **serialize** (const Slot< ReturnT, SlotTypes... > &slot, std::index_sequence< is... >)
- template<typename ReturnT , typename ... SlotTypes>
  std::enable_if_t<(sizeof ...(SlotTypes) > 1) &&std::is_same_v< ReturnT, void >, ReturnT > **invoke** (const Event::SharedPtr &event, SlotTypes &&...args) const
- template<typename ReturnT , typename ... SlotTypes>
  std::enable_if_t<(sizeof ...(SlotTypes) > 1) &&!std::is_same_v< ReturnT, void >, ReturnT > **invoke** (const Event::SharedPtr &event, SlotTypes &&...args) const

6.13.1 Detailed Description

Type erasure wrapper for storing generic functions. For internal use.

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

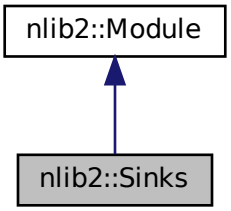
- include/nlib2/nl_modflow.h
- include/nlib2/nl_modflow_impl.hpp

6.14 nlib2::Sinks Class Reference

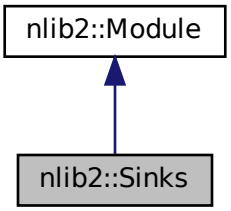
Sinks is a default module that does not create regular channels, but the Parent can create "sink" channels, to which Modules can regularly emit signals, but they are connected to the external parent's callback.

`#include <nl_modflow.h>`

Inheritance diagram for nlib2::Sinks:



Collaboration diagram for nlib2::Sinks:



Public Member Functions

- **Sinks** (const std::shared_ptr< Modflow > &modflow)
- void setupNetwork () override

  No network to setup.
- void initParams (const NlParams::SharedPtr &params) override

  No need for parameters.
- template<typename ... ChannelTs, typename Callback >
  void declareSink (const std::string &name, const Callback &parentCallback)

  Create a special channel marked as "sink", and connect signals emitted on such channel to `parentCallback`.

Additional Inherited Members

6.14.1 Detailed Description

Sinks is a default module that does not create regular channels, but the Parent can create "sink" channels, to which Modules can regularly emit signals, but they are connected to the external parent's callback.

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

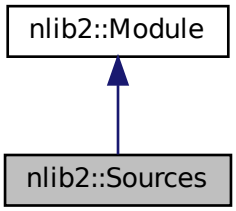
- include/nlib2/nl_modflow.h
- include/nlib2/nl_modflow_impl.hpp

6.15 nlib2::Sources Class Reference

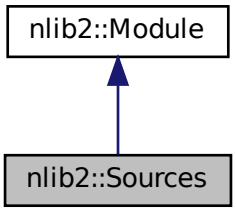
Sources are a particular Module whose channels are declared by the parent object, which emits signals on such channels externally. The Modflow object will automatically load a module, available externally via Modflow::sources.

```
#include <nl_modflow.h>
```

Inheritance diagram for nlib2::Sources:



Collaboration diagram for nlib2::Sources:



Protected Member Functions

- Sources (const std::shared_ptr< Modflow > &modflow)
- void initParams (const NlParams::SharedPtr &params) override

  This module cannot be overridden, so no parameters can be associated to it.
- void setupNetwork () override

  Channel are created externally via Sources::declareSource.
- template<typename ... ChannelTs>
  Channel declareSourceChannel (const std::string &name)

  Externally create a channel from the parent object of types ...ChannelTs.
- template<typename ... ChannelTs>
  void callSource (const Channel &channel, ChannelTs &&...args)

  Emit an event from the parent object on channel `channel` with data ...args.
- template<typename ... ChannelTs>
  void callSource (const std::string &name, ChannelTs &&...args)

  As Sources::callSources (const Channel &...) but resolves the channel `name`.

Additional Inherited Members

6.15.1 Detailed Description

Sources are a particular Module whose channels are declared by the parent object, which emits signals on such channels externally. The Modflow object will automatically load a module, available externally via Modflow::sources.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 Sources()

```
nlib2::Sources::Sources (
            const std::shared_ptr< Modflow > & modflow )  [inline], [protected]
```

See also

Module::Module

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

- include/nlib2/nl_modflow.h
- include/nlib2/nl_modflow_impl.hpp

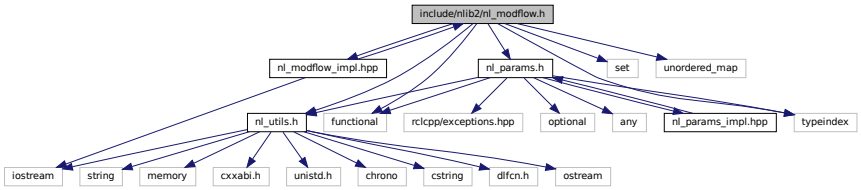
Chapter 7

File Documentation

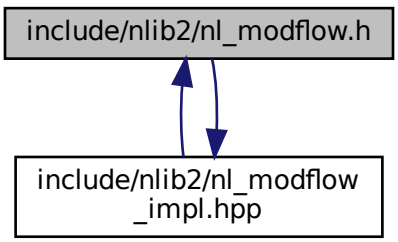
7.1 include/nlib2/nl_modflow.h File Reference

```
#include "nl_utils.h"
#include "nl_params.h"
#include <functional>
#include <set>
#include <typeindex>
#include <unordered_map>
#include "nl_modflow_impl.hpp"
```

Include dependency graph for nl_modflow.h:



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



Classes

- class nlib2::Event
- class nlib2::Channel

  Defines a channel that each module can create and to which other modules can connect.
- class nlib2::ResourceManager

  Allow sharing resources of generic types among modules.
- class nlib2::Module

  This is the core node of a Modflow graph. Inherit this class to define the main computation units to happen in this module. Each module can define new Channels to which it can emit output events after computation is done, and request channels to connect to when data is transmitted on such channels.
- class nlib2::Sources

  Sources are a particular Module whose channels are declared by the parent object, which emits signals on such channels externally. The Modflow object will automatically load a module, available externally via Modflow::sources.
- class nlib2::Sinks

  Sinks is a default module that does not create regular channels, but the Parent can create "sink" channels, to which Modules can regularly emit signals, but they are connected to the external parent's callback.
- class nlib2::SerializedSlot

  Type erasure wrapper for storing generic functions. For internal use.
- class nlib2::Modflow

  This is the main class that handles the call flow between modules. First, you will need to derive this class and override loadModules, calling loadModule with the type of each module you want to load as template parameter. Then:

Typedefs

- using **nlib2::ChannelId** = int64_t
- template<typename R , typename ... T>
  using **nlib2::Slot** = std::function< R(const Event::SharedPtr &, T ...)>

7.1.1 Detailed Description

Author

Nicola Lissandrini

Index