## ns3::CommandLine Class Reference

**Core » Command Line Parsing** 

Parse command-line arguments. More...

#include "command-line.h"

▶ Collaboration diagram for ns3::CommandLine:

## Classes

class CallbackItem

An argument Item using a Callback to parse the input. More...

class Item

The argument abstract base class. More...

class StringItem

class UserItem

An argument Item assigning to POD. More...

## **Public Member Functions**

CommandLine ()

Constructor. More...

CommandLine (const CommandLine &cmd)

Copy constructor. More...

~CommandLine ()

Destructor. More...

template<typename T >

void AddNonOption (const std::string name, const std::string help, T &value)

Add a non-option argument, assigning to POD. More...

template<typename T >

void AddValue (const std::string &name, const std::string &help, T &value)

Add a program argument, assigning to POD. More...

void AddValue (const std::string &name, const std::string &help, Callback< bool, std::string > callback)

Add a program argument, using a Callback to parse the value. More...

void AddValue (const std::string &name, const std::string &attributePath)

Add a program argument as a shorthand for an Attribute. More...

std::string GetExtraNonOption (std::size\_t i) const

Get extra non-option arguments by index. More...

std::string GetName () const

Get the program name. More...

std::size\_t GetNExtraNonOptions (void) const

Get the total number of non-option arguments found, including those configured with **AddNonOption()** and extra non-option arguments. **More...** 

## **Private Types**

typedef std::vector< Item \* > Items

Argument list container. More...

## **Private Member Functions**

void Clear (void)

Remove all arguments, Usage(), name. More...

void Copy (const CommandLine &cmd)

Copy constructor. More...

void HandleArgument (const std::string &name, const std::string &value) const

Match name against the program or general arguments, and dispatch to the appropriate handler. More...

bool HandleNonOption (const std::string &value)

Handle a non-option. More...

bool HandleOption (const std::string &param) const

Handle an option in the form param=value. More...

void PrintAttributes (std::ostream &os, const std::string &type) const

Handler for -- PrintAttributes: print the attributes for a given type. More...

void PrintGlobals (std::ostream &os) const

Handler for -- PrintGlobals: print all global variables and values. More...

void PrintGroup (std::ostream &os, const std::string &group) const

Handler for -- PrintGroup: print all types belonging to a given group. More...

void PrintGroups (std::ostream &os) const

Handler for -- PrintGroups: print all Typeld group names. More...

void PrintTypelds (std::ostream &os) const

Handler for -- PrintTypeIds: print all TypeId names. More...

### Static Private Member Functions

static bool HandleAttribute (const std::string name, const std::string value)

Callback function to handle attributes. More...

### **Private Attributes**

```
std::string m_name
The program name. More...

std::size_t m_NNonOptions
The expected number of non-option arguments. More...

std::size_t m_nonOptionCount
The number of actual non-option arguments seen so far. More...

Items m_nonOptions
The list of non-option arguments. More...

Items m_options
The list of option arguments. More...

std::string m_usage
The Usage string. More...
```

## **Detailed Description**

Parse command-line arguments.

Instances of this class can be used to parse command-line arguments. Programs can register a general usage message with CommandLine::AddValue. Argument variable types with input streamers (operator>>) can be set directly; more complex argument parsing can be accomplished by providing a Callback.

CommandLine also provides handlers for these standard arguments:

```
--PrintGlobals: Print the list of globals.
--PrintGroups: Print the list of groups.
--PrintGroup=[group]: Print all TypeIds of group.
--PrintTypeIds: Print all TypeIds.
--PrintAttributes=[typeid]: Print all attributes of typeid.
--PrintHelp: Print this help message.
```

The more common —help is a synonym for —PrintHelp; an example is given below.

**CommandLine** can also handle non-option arguments (often called simply "positional" parameters: arguments which don't begin with "-" or "--"). These can be parsed directly in to variables, by registering arguments with AddNonOption in the order expected. Additional non-option arguments encountered will be captured as strings.

Finally, CommandLine processes Attribute and GlobalValue arguments. Default values for specific attributes can be set using a shorthand argument name.

In use, arguments are given in the form

```
--arg=value --toggle first-non-option
```

Most arguments expect a value, as in the first form, -arg=value. Toggles, corresponding to boolean arguments, can be given in any of the forms

```
--toggle1 --toggle2=1 --toggle3=t --toggle4=true
```

The first form changes the state of toggle1 from its default; all the rest set the corresponding boolean variable to true. 0, f and false are accepted to set the variable to false. Option arguments can appear in any order on the command line, even intermixed with non-option arguments. The order of non-option arguments is preserved.

Option arguments can be repeated on the command line; the last value given will be the final value used. For example,

```
--arg=one --toggle=f --arg=another --toggle
```

The variable set by -arg will end up with the value "another"; the boolean set by -toggle will end up as true.

Because arguments can be repeated it can be hard to decipher what value each variable ended up with, especially when using boolean toggles. Suggested best practice is for scripts to report the values of all items settable through **CommandLine**, as done by the example below.

**CommandLine** can set the initial value of every attribute in the system with the —TypeIdName::AttributeName=value syntax, for example

```
--Application::StartTime=3s
```

In some cases you may want to highlight the use of a particular attribute for a simulation script. For example, you might want to make it easy to set the Application::StartTime using the argument—start, and have its help string show as part of the help message. This can be done using the AddValue (name, attributePath) method.

CommandLine can also set the value of every GlobalValue in the system with the -GlobalValueName=value syntax, for example

```
--SchedulerType=HeapScheduler
```

A simple example of CommandLine is in src/core/example/command-line-example.cc See that file for an example of handling non-option arguments.

The heart of that example is this code:

```
int
             intArg
                     = 1;
             boolArg = false;
strArg = "strArg default";
bool
std::string strArg
CommandLine cmd;
cmd Usage ("CommandLine example program.\n"
            "This little program demonstrates how to use CommandLine.");
                            "an int argument",
"a bool argument",
cmd.AddValue (
               "intArg"
                                                        intArg);
              ("boolArg"
("strArg"
                                                        boolArg);
cmd.AddValue
               "strArg
                            "a string argument"
cmd.AddValue
                                                        strArg)
              ("anti"
                            "ns3::RandomVariableStream::Antithetic");
cmd.AddValue
cmd AddValue ("cbArg"
                            "a string via callback", MakeCallback (SetCbArg));
cmd.Parse (argc, argv);
```

after which it prints the values of each variable. (The SetCbArg function is not shown here; see src/core/example/command-line-example.cc)

Here is the output from a few runs of that program:

```
$ ./waf --run="command-line-example"
intArg: 1
boolArg: false
strArg: "strArg default"
cbArg: "cbArg default"

$ ./waf --run="command-line-example --intArg=2 --boolArg --strArg=Hello --cbArg=World"
intArg: 2
boolArg: true
```

```
strarg:
          "негго"
          "World"
cbArg:
$ ./waf --run="command-line-example --help"
ns3-dev-command-line-example-debug [Program Arguments] [General Arguments]
CommandLine example program.
This little program demonstrates how to use CommandLine.
Program Arguments:
    --intArg:
                an int argument [1]
    --boolArg: a bool argument [false]
              a string argument [strArg default]
    --strArg:
                Set this RNG stream to generate antithetic values (ns3::RandomVariableStre
    --anti:
    --cbArg:
                a string via callback
General Arguments:
    --PrintGlobals:
                                 Print the list of globals.
                                 Print the list of groups.
    --PrintGroups:
                                 Print all TypeIds of group.
    --PrintGroup=[group]:
                                 Print all TypeIds.
    --PrintTypeIds:
                                 Print all attributes of typeid.
    --PrintAttributes=[typeid]:
                                 Print this help message.
    --PrintHelp:
```

Having parsed the arguments, some programs will need to perform some additional validation of the received values. A common issue at this point is to discover that the supplied arguments are incomplete or incompatible. Suggested best practice is to supply an error message and the complete usage message. For example,

```
int value1;
int value2;

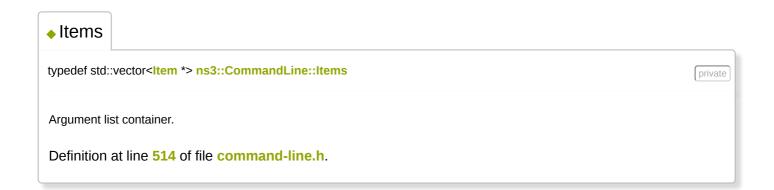
CommandLine cmd;
cmd.Usage ("...");
cmd.AddValue ("value1", "first value", value1);
cmd.AddValue ("value2", "second value", value1);

cmd.Parse (argc, argv);

if (value1 * value2 < 0)
{
    std::cerr << "value1 and value2 must have the same sign!" << std::endl;
    std::cerr << cmd;
    exit (-1);
}</pre>
```

Definition at line 213 of file command-line.h.

## Member Typedef Documentation



CommandLine() [1/2]

ns3::CommandLine::CommandLine ( )

Constructor.

Definition at line 48 of file command-line.cc.

References NS\_LOG\_FUNCTION.

CommandLine() [2/2]

ns3::CommandLine::CommandLine ( const CommandLine & cmd )

Copy constructor.

**Parameters** 

[in] cmd The CommandLine to copy from

Definition at line 54 of file command-line.cc.

References **second::cmd**, and **Copy()**.

▶ Here is the call graph for this function:

◆~CommandLine()

ns3::CommandLine::~CommandLine ( )

Destructor.

Definition at line 65 of file command-line.cc.

References Clear(), and NS\_LOG\_FUNCTION.

▶ Here is the call graph for this function:

**Member Function Documentation** 

AddNonOption()

Add a non-option argument, assigning to POD.

#### **Parameters**

- [in] **name** The name of the program-supplied argument
- [in] help The help text used by --PrintHelp
- [out] value A reference to the variable where the value parsed will be stored (if no value is parsed, this variable is not modified).

Definition at line 596 of file command-line.h.

 $References \ \textbf{ns3}:: \textbf{CommandLine}:: \textbf{UserItem} < \textbf{T} > :: \textbf{m\_default}, \ \textbf{ns3}:: \textbf{CommandLine}:: \textbf{Item}:: \textbf{m\_help}, \\$ 

ns3::CommandLine::Item::m\_name, m\_NNonOptions, m\_nonOptions, and

ns3::CommandLine::UserItem< T >::m\_valuePtr.

# ◆AddValue() [1/3]

template<typename T >

 $void\ ns 3 \hbox{::} Command \hbox{Line} \hbox{::} Add Value\ (\ const\ std \hbox{::} string\ \&\ \ name,$ 

const std::string & help,

T & value

)

Add a program argument, assigning to POD.

### **Parameters**

- [in] name The name of the program-supplied argument
- [in] help The help text used by --PrintHelp
- [out] value A reference to the variable where the value parsed will be stored (if no value is parsed, this variable is not modified).

Definition at line 578 of file command-line.h.

References ns3::CommandLine::UserItem< T >::m\_default, ns3::CommandLine::Item::m\_help, ns3::CommandLine::Item::m\_name, m\_options, and ns3::CommandLine::UserItem< T >::m\_valuePtr.

Referenced by AddValue().

▶ Here is the caller graph for this function:

```
◆AddValue() [2/3]
```

Add a program argument, using a Callback to parse the value.

#### **Parameters**

[in] name The name of the program-supplied argument

[in] help The help text used by -help

[in] callback A Callback function that will be invoked to parse and store the value.

The callback should have the signature bool callback (const std::string value)

Definition at line 540 of file command-line.cc.

References ns3::CommandLine::CallbackItem::m\_callback, ns3::CommandLine::Item::m\_help, ns3::CommandLine::Item::m\_name, m\_options, and NS\_LOG\_FUNCTION.

# ◆AddValue() [3/3]

Add a program argument as a shorthand for an Attribute.

#### **Parameters**

[in] **name** The name of the program-supplied argument.

[out] attributePath The fully-qualified name of the Attribute

Definition at line 553 of file command-line.cc.

References AddValue(), ns3::Typeld::AttributeInformation::checker, HandleAttribute(), ns3::Typeld::AttributeInformation::help, ns3::Typeld::AttributeInformation::initialValue, ns3::Typeld::LookupAttributeByName(), ns3::Typeld::LookupByNameFailSafe(), ns3::MakeBoundCallback(), NS\_FATAL\_ERROR, NS\_LOG\_DEBUG, and NS\_LOG\_FUNCTION.

▶ Here is the call graph for this function:



void ns3::CommandLine::Clear ( void )

private

Remove all arguments, Usage(), name.

Definition at line 83 of file command-line.cc.

References m\_name, m\_NNonOptions, m\_nonOptions, m\_options, m\_usage, and NS\_LOG\_FUNCTION.

Referenced by operator=(), and ~CommandLine().

▶ Here is the caller graph for this function:

## Copy()

void ns3::CommandLine::Copy ( const CommandLine & cmd )

private

Copy constructor.

#### **Parameters**

[in] cmd CommandLine to copy

Definition at line 71 of file command-line.cc.

References second::cmd, m\_name, m\_NNonOptions, m\_nonOptions, m\_options, m\_usage, and NS LOG FUNCTION.

Referenced by CommandLine(), and operator=().

▶ Here is the caller graph for this function:

# GetExtraNonOption()

std::string ns3::CommandLine::GetExtraNonOption ( std::size\_t i ) const

Get extra non-option arguments by index.

This allows CommandLine to accept more non-option arguments than have been configured explicitly with AddNonOption().

This is only valid after calling Parse().

### **Parameters**

[in] i The index of the non-option argument to return.

### Returns

The i'th non-option argument, as a string.

Definition at line 585 of file command-line.cc.

References m\_NNonOptions, m\_nonOptions, and ns3::CommandLine::StringItem::m\_value.

# GetName()

std::string ns3::CommandLine::GetName ( void ) const

Get the program name.

### Returns

The program name. Only valid after calling Parse()

Definition at line 109 of file command-line.cc.

References m\_name.

# GetNExtraNonOptions()

 $std::size\_t \ ns3::CommandLine::GetNExtraNonOptions \ (\ void \ \ ) \ const$ 

Get the total number of non-option arguments found, including those configured with AddNonOption() and extra non-option arguments.

This is only valid after calling Parse().

### Returns

the number of non-option arguments found.

Definition at line **601** of file **command-line.cc**.

References m\_NNonOptions, and m\_nonOptions.

# HandleArgument()

static private

# HandleAttribute()

```
bool ns3::CommandLine::HandleAttribute ( const std::string name, const std::string value )
```

Callback function to handle attributes.

▶ Here is the caller graph for this function:

## **Parameters**

[in] name The full name of the Attribute.

[in] value The value to assign to name.

### Returns

true if the value was set successfully, false otherwise.

Definition at line 616 of file command-line.cc.

References ns3::Config::SetDefaultFailSafe(), and ns3::Config::SetGlobalFailSafe().

Referenced by AddValue().

- ▶ Here is the call graph for this function:
- ▶ Here is the caller graph for this function:

# HandleNonOption()

bool ns3::CommandLine::HandleNonOption ( const std::string & value )

Handle a non-option.

Parameters

[in] value The command line non-option value.

Returns

true if value could be parsed correctly.

Definition at line 193 of file command-line.cc.

References ns3::CommandLine::Item::m\_name, m\_nonOptionCount, m\_nonOptions, NS\_LOG\_FUNCTION, and NS\_LOG\_LOGIC().

Referenced by Parse().

Here is the call graph for this function:

# HandleOption()

bool ns3::CommandLine::HandleOption ( const std::string & param ) const

private

Handle an option in the form param=value.

▶ Here is the caller graph for this function:

### **Parameters**

[in] param The option string.

### Returns

true if this was really an option.

Definition at line 152 of file command-line.cc.

References HandleArgument().

Referenced by Parse().

- ▶ Here is the call graph for this function:
- ▶ Here is the caller graph for this function:

# operator=()

CommandLine & ns3::CommandLine::operator= ( const CommandLine & cmd )

Assignment.

### **Parameters**

[in] cmd The CommandLine to assign from

### Returns

The CommandLine

Definition at line 59 of file command-line.cc.

References Clear(), second::cmd, and Copy().

▶ Here is the call graph for this function:

# • Parse() [1/2]

Parse the program arguments.

### **Parameters**

[in] argc The 'argc' variable: number of arguments (including the main program name as first element).

[in] argv The 'argv' variable: a null-terminated array of strings, each of which identifies a command-line argument.

Obviously, this method will parse the input command-line arguments and will attempt to handle them all.

As a side effect, this method saves the program basename, which can be retrieved by GetName().

Definition at line 224 of file command-line.cc.

References NS\_LOG\_FUNCTION.

• Parse() [2/2]

```
void ns3::CommandLine::Parse ( std::vector< std::string > args )
```

Parse the program arguments.

This version may be convenient when synthesizing arguments programmatically. Other than the type of argument this behaves identically to Parse(int, char \*)

### **Parameters**

[in] args The vector of arguments.

Definition at line 120 of file command-line.cc.

References ns3::Singleton< DesMetrics >::Get(), HandleNonOption(), HandleOption(), ns3::DesMetrics::Initialize(), m\_name, m\_nonOptionCount, NS\_ASSERT\_MSG(), NS\_LOG\_FUNCTION, and ns3::SystemPath::Split().

▶ Here is the call graph for this function:

## PrintAttributes()

```
    void ns3::CommandLine::PrintAttributes ( std::ostream & os,

    const std::string & type

    )
    const

private
```

Handler for -- PrintAttributes: print the attributes for a given type.

#### **Parameters**

[in, out] os the output stream.

[in] type The Typeld whose Attributes should be displayed

Definition at line 346 of file command-line.cc.

References ns3::Typeld::AttributeInformation::checker, ns3::Typeld::GetAttribute(), ns3::Typeld::GetAttributeFullName(), ns3::Typeld::GetAttributeN(), ns3::Typeld::GetAttributeInformation::help, ns3::Typeld::AttributeInformation::initialValue, ns3::Typeld::LookupByNameFailSafe(), NS\_FATAL\_ERROR, and NS\_LOG\_FUNCTION.

Referenced by HandleArgument().

▶ Here is the call graph for this function:

▶ Here is the caller graph for this function:

# PrintGlobals()

```
void ns3::CommandLine::PrintGlobals ( std::ostream & os ) const

Handler for --PrintGlobals: print all global variables and values.

Parameters

[in, out] os The output stream to print on.

Definition at line 314 of file command-line.cc.

References ns3::GlobalValue::Begin(), ns3::GlobalValue::End(), ns3::StringValue::Get(), and Ns_LOG_FUNCTION.

Referenced by HandleArgument().

Here is the call graph for this function:

Here is the caller graph for this function:
```

```
PrintGroup()
void ns3::CommandLine::PrintGroup ( std::ostream &
                                                   OS.
                                  const std::string & group
                                 )
                                                   const
                                                                                                            private
Handler for -- PrintGroup: print all types belonging to a given group.
Parameters
       [in,out] os
                        The output stream.
       [in]
                 group The name of the Typeld group to display
Definition at line 382 of file command-line.cc.
References ns3::TypeId::GetGroupName(), ns3::TypeId::GetName(), ns3::TypeId::GetRegistered(),
ns3::TypeId::GetRegisteredN(), and NS_LOG_FUNCTION.
Referenced by HandleArgument().
▶ Here is the call graph for this function:
▶ Here is the caller graph for this function:
```

# PrintGroups()

```
void ns3::CommandLine::PrintGroups ( std::ostream & os ) const

Handler for --PrintGroups: print all Typeld group names.

Parameters

[in, out] os The output stream.

Definition at line 436 of file command-line.cc.

References ns3::Typeld::GetGroupName(), ns3::Typeld::GetRegistered(), ns3::Typeld::GetRegisteredN(), and NS_LOG_FUNCTION.

Referenced by HandleArgument().

Here is the call graph for this function:
```

## PrintHelp()

void ns3::CommandLine::PrintHelp ( std::ostream & os ) const

Print program usage to the desired output stream.

▶ Here is the caller graph for this function:

Handler for -- PrintHelp and --help: print Usage(), argument names, and help strings

Alternatively, an overloaded operator << can be used:

```
CommandLine cmd;
cmd.Parse (argc, argv);
...
std::cerr << cmd;</pre>
```

### **Parameters**

[in, out] os The output stream to print on.

Definition at line 232 of file command-line.cc.

References ns3::if(), m\_name, m\_NNonOptions, m\_nonOptions, m\_options, m\_usage, max, and NS\_LOG\_FUNCTION.

Referenced by HandleArgument().

- ▶ Here is the call graph for this function:
- ▶ Here is the caller graph for this function:

# PrintTypeIds()

void ns3::CommandLine::PrintTypeIds ( std::ostream & os ) const

Handler for --PrintTypeIds: print all TypeId names.

Parameters

[in, out] os The output stream.

Definition at line 411 of file command-line.cc.

References ns3::TypeId::GetName(), ns3::TypeId::GetRegistered(), ns3::TypeId::GetRegisteredN(), and NS\_LOG\_FUNCTION.

Referenced by HandleArgument().

Here is the call graph for this function:

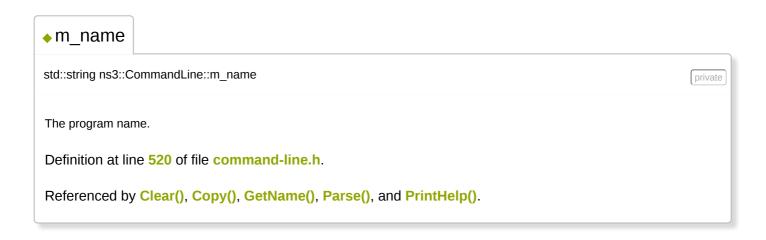
Here is the caller graph for this function:

◆ Usage()
 void ns3::CommandLine::Usage ( const std::string usage )
 Supply the program usage and documentation.
 Parameters

 [in] usage Program usage message to write with -help.

 Definition at line 103 of file command-line.cc.
 References m\_usage.

## Member Data Documentation



m\_NNonOptions

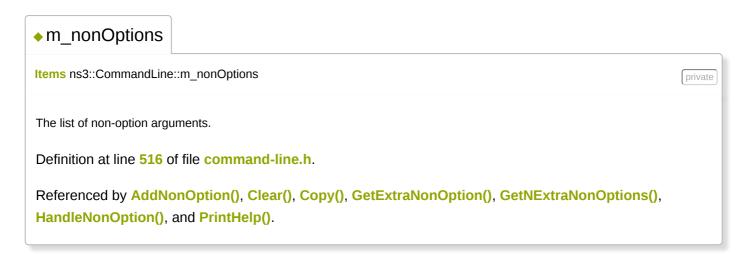
std::size\_t ns3::CommandLine::m\_NNonOptions

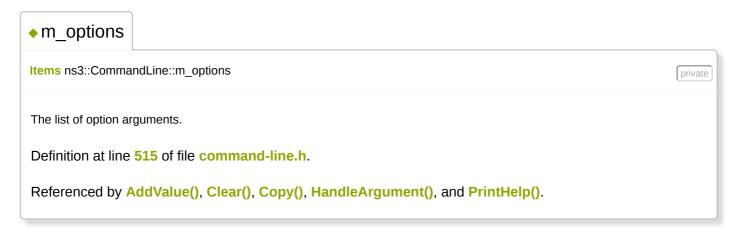
The expected number of non-option arguments.

Definition at line 517 of file command-line.h.

Referenced by AddNonOption(), Clear(), Copy(), GetExtraNonOption(), GetNExtraNonOptions(), and PrintHelp().

◆ m\_nonOptionCount
 std::size\_t ns3::CommandLine::m\_nonOptionCount
 The number of actual non-option arguments seen so far.
 Definition at line 518 of file command-line.h.
 Referenced by HandleNonOption(), and Parse().





m\_usage

std::string ns3::CommandLine::m\_usage

The Usage string.

Definition at line 519 of file command-line.h.

Referenced by Clear(), Copy(), PrintHelp(), and Usage().

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

- src/core/model/command-line.h
- src/core/model/command-line.cc