

# Canal

Karel Klíč

November 30, 2012

# Contents

<b>1 Overview</b>	<b>7</b>
<b>2 Installation</b>	<b>9</b>
<b>I Concepts</b>	<b>11</b>
<b>3 Preliminaries</b>	<b>13</b>
<b>4 LLVM</b>	<b>15</b>
<b>5 Abstract Interpretation</b>	<b>17</b>
<b>6 Memory Model</b>	<b>19</b>
<b>7 Array Abstract Domains</b>	<b>21</b>
<b>8 Structure Abstract Domain</b>	<b>23</b>
<b>9 Integer Abstract Domains</b>	<b>25</b>
9.1 Integer Interval Domain $\mathcal{D}_i^\sharp$ . . . . .	25
9.2 Integer Bitfield Domain $\mathcal{D}_b^\sharp$ . . . . .	25
<b>10 Abstract Domains for Floating-Point Numbers</b>	<b>27</b>
<b>11 Pointer Abstract Domains</b>	<b>29</b>
<b>12 Abstract Domain Combination</b>	<b>31</b>
<b>13 Wishlist</b>	<b>33</b>
13.1 Reduced Product of Abstract Domains . . . . .	33
13.2 Widening and Narrowing . . . . .	33
13.3 String Abstract Domains . . . . .	33
13.4 Trace Partitioning . . . . .	33
13.5 Boolean Partitioning . . . . .	33
13.6 Fixpoint Recalculation . . . . .	33
13.7 Multi Threading . . . . .	33
13.8 Symbolic Abstract Domains . . . . .	34
13.9 Weakly-Relational Abstract Domains . . . . .	34
13.10 Compositional Analysis . . . . .	34
13.11 Parallelization . . . . .	34
13.12 Custom Domains . . . . .	34

<b>II Implementation</b>	<b>35</b>
<b>14 Overview</b>	<b>37</b>
<b>15 Library Class Index</b>	<b>39</b>
15.1 Class Hierarchy . . . . .	39
15.2 Class List . . . . .	40
<b>16 Library Class Documentation</b>	<b>43</b>
16.1 Canal::Interpreter::BasicBlock Class Reference . . . . .	43
16.2 Canal::Integer::Bitfield Class Reference . . . . .	43
16.2.1 Detailed Description . . . . .	46
16.2.2 Member Function Documentation . . . . .	46
16.2.3 Member Data Documentation . . . . .	48
16.3 Canal::Constructors Class Reference . . . . .	49
16.3.1 Member Function Documentation . . . . .	50
16.4 Canal::Integer::Container Class Reference . . . . .	50
16.4.1 Member Function Documentation . . . . .	52
16.5 Canal::Widening::DataInterface Class Reference . . . . .	53
16.6 Canal::Widening::DataIterationCount Class Reference . . . . .	54
16.7 Canal::Domain Class Reference . . . . .	55
16.7.1 Detailed Description . . . . .	57
16.7.2 Constructor & Destructor Documentation . . . . .	58
16.7.3 Member Function Documentation . . . . .	58
16.8 Canal::Environment Class Reference . . . . .	58
16.9 Canal::Array::ExactSize Class Reference . . . . .	59
16.9.1 Detailed Description . . . . .	61
16.9.2 Constructor & Destructor Documentation . . . . .	62
16.9.3 Member Function Documentation . . . . .	62
16.10 Canal::Interpreter::Function Class Reference . . . . .	63
16.10.1 Member Function Documentation . . . . .	64
16.11 Canal::FunctionModel Class Reference . . . . .	64
16.12 Canal::FunctionModelManager Class Reference . . . . .	64
16.13 Canal::Array::Interface Class Reference . . . . .	65
16.13.1 Member Function Documentation . . . . .	65
16.14 Canal::Widening::Interface Class Reference . . . . .	67
16.15 Canal::Interpreter::Interpreter Class Reference . . . . .	67
16.15.1 Constructor & Destructor Documentation . . . . .	68
16.16 Canal::Integer::Interval Class Reference . . . . .	68
16.16.1 Detailed Description . . . . .	71
16.16.2 Constructor & Destructor Documentation . . . . .	71
16.16.3 Member Function Documentation . . . . .	71
16.16.4 Member Data Documentation . . . . .	73
16.17 Canal::Float::Interval Class Reference . . . . .	73
16.17.1 Member Function Documentation . . . . .	75
16.18 Canal::Interpreter::Iterator Class Reference . . . . .	76
16.18.1 Detailed Description . . . . .	76
16.18.2 Member Function Documentation . . . . .	76
16.19 Canal::Interpreter::IteratorCallback Class Reference . . . . .	76
16.20 Canal::Widening::Manager Class Reference . . . . .	77
16.21 MemoryOverlay Class Reference . . . . .	78
16.22 Canal::Interpreter::Module Class Reference . . . . .	78
16.23 Canal::Widening::NumericalInfinity Class Reference . . . . .	79
16.24 Canal::Operations Class Reference . . . . .	79

16.24.1 Detailed Description . . . . .	82
16.24.2 Member Function Documentation . . . . .	83
16.25 Canal::OperationsCallback Class Reference . . . . .	85
16.25.1 Member Function Documentation . . . . .	85
16.26 Canal::Interpreter::OperationsCallback Class Reference . . . . .	86
16.26.1 Member Function Documentation . . . . .	87
16.27 Canal::Pointer::Pointer Class Reference . . . . .	87
16.27.1 Detailed Description . . . . .	89
16.27.2 Constructor & Destructor Documentation . . . . .	89
16.27.3 Member Function Documentation . . . . .	89
16.27.4 Member Data Documentation . . . . .	90
16.28 Canal::Widening::Pointers Class Reference . . . . .	91
16.29 Canal::APIntUtils::SCompare Struct Reference . . . . .	91
16.30 Canal::Integer::Set Class Reference . . . . .	92
16.30.1 Member Function Documentation . . . . .	94
16.31 Canal::SharedData Class Reference . . . . .	96
16.31.1 Constructor & Destructor Documentation . . . . .	97
16.32 Canal::SharedDataPointer< T > Class Template Reference . . . . .	97
16.33 Canal::Array::SingleItem Class Reference . . . . .	97
16.33.1 Detailed Description . . . . .	100
16.33.2 Constructor & Destructor Documentation . . . . .	100
16.33.3 Member Function Documentation . . . . .	100
16.33.4 Member Data Documentation . . . . .	102
16.34 Canal::SlotTracker Class Reference . . . . .	102
16.34.1 Detailed Description . . . . .	103
16.34.2 Member Function Documentation . . . . .	103
16.35 Canal::State Class Reference . . . . .	103
16.35.1 Detailed Description . . . . .	104
16.35.2 Member Function Documentation . . . . .	104
16.36 Canal::StateMap Class Reference . . . . .	105
16.37 Canal::Array::StringPrefix Class Reference . . . . .	106
16.37.1 Detailed Description . . . . .	108
16.37.2 Constructor & Destructor Documentation . . . . .	109
16.37.3 Member Function Documentation . . . . .	109
16.38 Canal::StringStream Class Reference . . . . .	110
16.38.1 Detailed Description . . . . .	110
16.39 Canal::Structure Class Reference . . . . .	111
16.39.1 Member Function Documentation . . . . .	112
16.40 Canal::Pointer::Target Class Reference . . . . .	114
16.40.1 Detailed Description . . . . .	115
16.40.2 Constructor & Destructor Documentation . . . . .	115
16.40.3 Member Data Documentation . . . . .	116
16.41 Canal::APIntUtils::UCompare Struct Reference . . . . .	116
16.42 Canal::Pointer::Utils Class Reference . . . . .	116
16.43 Canal::VariableArguments Class Reference . . . . .	117
16.43.1 Member Function Documentation . . . . .	117
<b>17 Tool Class Index</b>	<b>119</b>
17.1 Class Hierarchy . . . . .	119
17.2 Class List . . . . .	119
<b>18 Tool Class Documentation</b>	<b>121</b>
18.1 Arguments Class Reference . . . . .	121
18.2 Command Class Reference . . . . .	123

18.2.1 Member Function Documentation . . . . .	124
18.3 CommandBreak Class Reference . . . . .	125
18.3.1 Member Function Documentation . . . . .	126
18.4 CommandCd Class Reference . . . . .	126
18.4.1 Member Function Documentation . . . . .	127
18.5 CommandContinue Class Reference . . . . .	128
18.5.1 Member Function Documentation . . . . .	129
18.6 CommandDump Class Reference . . . . .	129
18.6.1 Member Function Documentation . . . . .	130
18.7 CommandFile Class Reference . . . . .	131
18.7.1 Member Function Documentation . . . . .	132
18.8 CommandFinish Class Reference . . . . .	132
18.8.1 Member Function Documentation . . . . .	133
18.9 CommandHelp Class Reference . . . . .	134
18.9.1 Member Function Documentation . . . . .	135
18.10 CommandInfo Class Reference . . . . .	135
18.10.1 Member Function Documentation . . . . .	136
18.11 CommandPrint Class Reference . . . . .	137
18.11.1 Member Function Documentation . . . . .	138
18.12 CommandPwd Class Reference . . . . .	139
18.12.1 Member Function Documentation . . . . .	140
18.13 CommandQuit Class Reference . . . . .	140
18.13.1 Member Function Documentation . . . . .	141
18.14 CommandRun Class Reference . . . . .	142
18.14.1 Member Function Documentation . . . . .	143
18.15 Commands Class Reference . . . . .	143
18.16 CommandSet Class Reference . . . . .	144
18.16.1 Member Function Documentation . . . . .	145
18.17 CommandShell Class Reference . . . . .	146
18.17.1 Member Function Documentation . . . . .	147
18.18 CommandShow Class Reference . . . . .	147
18.18.1 Member Function Documentation . . . . .	148
18.19 CommandStart Class Reference . . . . .	149
18.19.1 Member Function Documentation . . . . .	150
18.20 CommandStep Class Reference . . . . .	150
18.20.1 Member Function Documentation . . . . .	151
18.21 FunctionDetailedInfo Struct Reference . . . . .	152
18.22 FunctionEntry Struct Reference . . . . .	152
18.23 FunctionInfo Class Reference . . . . .	153
18.24 IteratorCallback Class Reference . . . . .	154
18.25 LoopTree Class Reference . . . . .	155
18.26 State Class Reference . . . . .	155
<b>19 Known Bugs</b>	<b>157</b>
<b>20 Wishlist</b>	<b>159</b>
20.1 Callbacks Interface . . . . .	159
20.2 Models . . . . .	159
20.3 Support of Multiple Platforms . . . . .	159
20.4 Automatic Tests . . . . .	159
20.5 Graphical User Interface . . . . .	159
<b>Bibliography</b>	<b>161</b>



# Chapter 1

## Overview

For a sufficiently complex software system, its maintainability and extensibility is limited by our ability to understand and correctly approximate the behaviour of the system, trace the impact of system parts to each other, control the impact of modifications, ensure correctness of the critical parts, and fixing bugs before they cause serious consequences in production.

The maintainability and extensibility is affected by the programming language of the implementation. Efficient low-level languages such as C and C++ increase the complexity of the system by being closely aligned with hardware. Systems must handle memory management, operate on machine-dependent integers and floating point numbers, and cooperate with an environment with complex invariants and interdependencies.

Canal is a static analysis tool designed to analyze behaviour of application programs. It is based on the theoretical framework of abstract interpretation, with focus on the scalability to large programs and proper handling of real-world source code.



# Chapter 2

## Installation

This chapter provides instructions for installing Canal on supported platforms. Canal can be built and installed on most GNU/Linux operating systems.

Canal comes with two build systems: Autotools (also known as GNU build system) and CMake. The systems are equivalent in terms of features, but they come with different advantages and disadvantages. Autotools system works on Unix-like operating systems only, and CMake also works on other operating systems such as Microsoft Windows. CMake tool is required to be installed to build Canal via CMake while Autotools require no tool to be present on the system. Supporting multiple versions of the CMake tool is not straightforward, but it is something that needs to be done when supporting CMake for multiple operating systems.

### **Building and installing Canal with Autotools**

The first step is to configure the source code, telling Canal where various dependencies are located and where various files will be installed. To do so, go to the directory with the Canal source code tree and run the `configure` script:

```
./configure
```

It prints messages telling which features it checks and which dependencies it found. With default settings, the build system is configured to optimize the resulting code to make Canal run faster. If you need to debug Canal, set compiler flags to avoid any optimization:

```
./configure CXXFLAGS="-g -O0"
```

Once the `configure` script has been run, you can compile Canal by running `make`:

```
make
```

After compiling Canal, you can verify your compiled program can operate well by running the test suite and seeing that all tests pass:

```
make check
```

Canal can be now installed. If it is going to be installed to system directories, special privileges might be needed. To install Canal, run

```
make install
```

### **Building and installing Canal with CMake**

The first step is to configure the source code, telling Canal where various dependencies are located and where various files will be installed. To do so, go to the directory with the Canal source code tree and run

the `cmake` program:

```
cmake .
```

It prints messages telling which dependencies it found.

Once the `cmake` tool has been run, you can compile Canal by running `make`:

```
make
```

After compiling Canal, you can verify your compiled program can operate well by running the test suite and seeing that all tests pass:

```
make test
```

#### Installation from source code on Fedora and Red Hat Enterprise Linux

Canal can be built, installed, and developed on a computer with the Red Hat Enterprise Linux 6 or Fedora 17 operating systems.

Specific software packages are required by the build process and should be installed prior to building Canal:

- The `llvm-devel` and `clang` packages. On Red Hat Enterprise Linux, these packages can be obtained from Extra Packages for Enterprise Linux (EPEL) software repository.
- The `elfutils-devel` and `readline-devel` packages. These are needed for the command-line user interface tool. If `elfutils-devel` is not present, the tool is built without the ELF support.
- The `doxygen`, `graphviz`, and `texlive-latex` packages. These are needed to build the documentation. If not present, the documentation is not built.

Both Autotools and CMake can be used to build Canal on Fedora and Red Hat Enterprise Linux.

**Part I**

# **Concepts**



## Chapter 3

# Preliminaries

As a preliminary step we shall define terms from the order theory. Detailed explanation can be found in [7] and [8].

A binary relation  $\sqsubseteq$  is *reflexive* on a set  $\mathcal{D}$  if every element is related to itself:  $a \sqsubseteq a$  for all  $a \in \mathcal{D}$ . A binary relation  $\sqsubseteq$  is *antisymmetric* on a set  $\mathcal{D}$  if the following implication holds:  $a \sqsubseteq b$  and  $b \sqsubseteq a$  implies  $a = b$ . A binary relation  $\sqsubseteq$  is *transitive* on a set  $\mathcal{D}$  if whenever an element  $a$  is related to an element  $b$ , and  $b$  is in turn related to an element  $c$ , then  $a$  is also related to  $c$ :  $a \sqsubseteq b$  and  $b \sqsubseteq c$  implies  $a \sqsubseteq c$ .

A *partial order*  $\sqsubseteq$  is a binary relation on a set  $\mathcal{D}$  which is reflexive, antisymmetric and transitive. A *partial ordered set* or *poset* for short is an ordered pair  $(\mathcal{D}, \sqsubseteq)$  of a set  $\mathcal{D}$  together with a partial ordering  $\sqsubseteq$ .

An element  $a$  in a poset  $(\mathcal{D}, \sqsubseteq)$  is called *maximal* if it is not less than any other element in  $\mathcal{D}$ :  $\nexists b \in \mathcal{D}, a \sqsubset b$ . If there is an unique maximal element, we call it the *greatest element* and denote it by  $\top$ . Similarly, an element  $a$  in a poset  $(\mathcal{D}, \sqsubseteq)$  is called *minimal* if it is not greater than any other element in  $\mathcal{D}$ :  $\nexists b \in \mathcal{D}, b \sqsubset a$ . If there is an unique minimal element, we call it the *least element* and denote it by  $\perp$ .

Let  $(\mathcal{D}, \sqsubseteq)$  be a poset and  $A \subseteq \mathcal{D}$ . An element  $u \in \mathcal{D}$  is an *upper bound* of  $A$  if  $a \sqsubseteq u$  for all elements  $a \in A$ . The *least upper bound* or *lub* for short is an element  $x$  that is an upper bound on a subset  $A$  and is less than all other upper bounds on  $A$ ; such an element is denoted by  $\sqcup A$ . Similarly, an element  $l \in \mathcal{D}$  is a *lower bound* of  $A$  if  $l \sqsubseteq a$  for all elements  $a \in A$ . The *greatest lower bound* or *glb* for short is an element  $x$  that is a lower bound on a subset  $A$  and is greater than all other lower bounds on  $A$ ; such an element is denoted by  $\sqcap A$ .

A *lattice*  $(\mathcal{D}, \sqsubseteq, \sqcup, \sqcap)$  is a partially ordered set in which any two elements  $a, b \in \mathcal{D}$  have both a least upper bound, denoted by  $a \sqcup b$ , and a greatest lower bound, denoted by  $a \sqcap b$ . A *complete lattice*  $(\mathcal{D}, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$  is a partially ordered set in which every subset  $A \subseteq \mathcal{D}$  has a least upper bound and a greatest lower bound. A complete lattice therefore has the greatest element  $\top$  defined as  $\sqcup \mathcal{D}$ , and the lowest element  $\perp$  defined as  $\sqcap \mathcal{D}$ .

A function  $F \in \mathcal{D}_1 \rightarrow \mathcal{D}_2$  between two posets  $(\mathcal{D}_1, \sqsubseteq_1)$  and  $(\mathcal{D}_2, \sqsubseteq_2)$  is *monotonic* if  $X \sqsubseteq_1 Y \implies F(X) \sqsubseteq_2 F(Y)$ . A function  $F \in \mathcal{D}_1 \rightarrow \mathcal{D}_2$  is *strict* if  $F(\perp_1) = \perp_2$ . A function  $F \in \mathcal{D}_1 \rightarrow \mathcal{D}_2$  is *continuous* if it preserves the existing limits of increasing chains  $(X_i)_{i \in I}$ :  $F(\sqcup_1 \{X_i \mid i \in I\}) = \sqcup_2 \{F(X_i) \mid i \in I\}$  whenever  $\sqcup_1 \{X_i \mid i \in I\}$  exists.

A *fixpoint* of a function  $F : \mathcal{D} \rightarrow \mathcal{D}$  on a poset  $(\mathcal{D}, \sqsubseteq)$  is an element  $x \in \mathcal{D}$  such that  $F(x) = x$ . A *prefixpoint* is an element  $x \in \mathcal{D}$  such that  $x \sqsubseteq F(x)$ . A *postfixpoint* is an element  $x \in \mathcal{D}$  such that  $F(x) \sqsubseteq x$ . A set of all fixpoints is denoted by  $\text{fp}(F)$ . A set of all prefixpoints is denoted by  $\text{prefp}(F)$ . A set of all postfixpoints is denoted by  $\text{postfp}(F)$ . The *least fixpoint* or *lfp* of a function  $F$  on a poset  $(\mathcal{D}, \sqsubseteq)$  satisfies  $\text{lfp} \in \text{fp}(F)$  and  $\forall p \in \text{fp}(F) : \text{lfp} \sqsubseteq p$ .

A *Galois connection* is a pair of two functions  $\alpha : \mathcal{D}_1 \rightarrow \mathcal{D}_2$  and  $\gamma : \mathcal{D}_2 \rightarrow \mathcal{D}_1$  on two preordered sets  $(\mathcal{D}_1, \sqsubseteq_1)$  and  $(\mathcal{D}_2, \sqsubseteq_2)$  iff  $\forall d_1 \in \mathcal{D}_1, \forall d_2 \in \mathcal{D}_2 : \alpha(d_1) \sqsubseteq_2 d_2 \equiv d_1 \sqsubseteq_1 \gamma(d_2)$ . It is denoted by

$$(\mathcal{D}_1, \sqsubseteq_1) \xrightleftharpoons[\alpha]{\gamma} (\mathcal{D}_2, \sqsubseteq_2).$$

# Chapter 4

## LLVM

Canal is built on the top of the LLVM [10] (Low-level Virtual Machine) compiler technology framework. Canal performs its static analysis over the LLVM intermediate representation, which is independent of source language and hide the complexity of target architecture. Canal is tested with C and C++ front-ends on 32-bit and 64-bit operating systems with little-endian memory layout, but it is expected that other source languages and platforms are supportable at low cost.

LLVM is suitable for efficient static analysis due to its design. Due to its type safety and Static Single Assignment (SSA) nature, most operations can be easily and precisely handled in static analysis. However, it is low enough level to support not only type conversion (creating a value of one data type from a value of another data type), but also type casting (changing the interpretation of the bit pattern representing a value from one type to another), pointer arithmetics, and manual memory management.

A subset of LLVM intermediate representation has been formalized in [17]. Figure 4.1 presents an updated abstract syntax that captures all attributes handled by Canal.

A module *mod* represents a translation unit of the input program. Most importantly, a module specifies list of *prod* that can be function declarations, function definitions, and global variables. It might also specify a target specific data layout string *layout* that specifies how data is to be laid out in memory, module-level inline assembler blocks *asm*, named types *namedt* that make the program shorter and easier to read, named metadata *namedm* that provide a collection of metadata, and aliases *alias* that act as a second name for the aliasee.

Types *typ* include arbitrary bit-width integers  $\mathbf{isz} \mid sz \in \mathbb{N}^*$ , such as **i1**, **i8**, **i32**, **i64**. They also include floating point types *fp*. The **void** type does not represent any value and has no size. Pointers *typ\** are used to specify memory locations. Arrays  $[sz \times typ]$  have statically known size *sz*. Structures  $\{\overline{typ}_j\}$  are defined as a list of types. Functions *typ*  $\overline{typ}_j$  consist of a return type and a list of parameter types. Types can also be named by identifiers *id*, which is useful for the definition of recursive types. The **label** type represents code labels. The **metadata** type represents embedded metadata.

Modules	<i>mod</i>	::= <i>layout</i> <i>asm</i> <i>namedt</i> <i>namedm</i> <i>alias</i> <i>prod</i>
Layouts	<i>layout</i>	::= <b>bigendian</b>   <b>little endian</b>   <b>ptr</b> <i>sz align<sub>abi</sub> align<sub>pref</sub></i>   <b>int</b> <i>sz align<sub>abi</sub> align<sub>pref</sub></i>   <b>float</b> <i>sz align<sub>abi</sub> align<sub>pref</sub></i>   <b>aggr</b> <i>sz align<sub>abi</sub> align<sub>pref</sub></i>   <b>vec</b> <i>sz align<sub>abi</sub> align<sub>pref</sub></i>   <b>stack</b> <i>sz align<sub>abi</sub> align<sub>pref</sub></i>
Products	<i>prod</i>	::= <i>id = global typ const align</i>   <b>define</b> <i>typ id{arg}</i> { <i>b</i> }   <b>declare</b> <i>typ id{arg}</i>
Floats	<i>fp</i>	::= <b>half</b>   <b>float</b>   <b>double</b>   <b>x86_fp80</b>   <b>fp128</b>   <b>ppc_fp128</b>
Vec types	<i>vtyp</i>	::= <i>fp</i>   <b>isz</b>   <i>fp*</i>   <b>isz*</b>
Types	<i>typ</i>	::= <b>isz</b>   <i>fp</i>   <b>void</b>   <i>typ*</i>   <b>[sz × typ]</b>   <b>[sz × vtyp]</b>   <b>{typ<sub>j</sub>}</b>   <i>typ typ<sub>j</sub></i>   <i>id</i>   <b>label</b>   <b>metadata</b>
Values	<i>val</i>	::= <i>id</i>   <i>cnst</i>
Binops	<i>bop</i>	::= <b>add</b>   <b>sub</b>   <b>mul</b>   <b>udiv</b>   <b>sdiv</b>   <b>urem</b>   <b>srem</b>   <b>shl</b>   <b>lshr</b>   <b>ashr</b>   <b>and</b>   <b>or</b>   <b>xor</b>
Float ops	<i>fbop</i>	::= <b>fadd</b>   <b>fsub</b>   <b>fmul</b>   <b>fdiv</b>   <b>frem</b>
Extension	<i>eop</i>	::= <b>zext</b>   <b>sext</b>   <b>fpext</b>
Cast ops	<i>cop</i>	::= <b>fptoui</b>   <b>ptrtoint</b>   <b>inttoptr</b>   <b>bitcast</b>
Trunc ops	<i>trop</i>	::= <b>trunc<sub>int</sub></b>   <b>trunc<sub>fp</sub></b>
Constants	<i>cnst</i>	::= <b>isz Int</b>   <b>fp Float</b>   <b>typ * id</b>   <b>(typ*) null</b>   <b>typ zeroinitializer</b>   <b>typ[cnst<sub>j</sub>]</b>   <b>{cnst<sub>j</sub>}</b>   <b>typ undef</b>   <b>bop cnst<sub>1</sub> cnst<sub>2</sub></b>   <b>fbop cnst<sub>1</sub> cnst<sub>2</sub></b>   <b>trop cnst to typ</b>   <b>eop cnst to typ</b>   <b>cop cnst to typ</b>   <b>getelementptr cnst cnst<sub>j</sub></b>   <b>select cnst<sub>0</sub> cnst<sub>1</sub> cnst<sub>2</sub></b>   <b>icmp cond cnst<sub>1</sub> cnst<sub>2</sub></b>   <b>fcmp fcond cnst<sub>1</sub> cnst<sub>2</sub></b>
Blocks	<i>b</i>	::= <i>l φ c tmn</i>
φ nodes	<i>φ</i>	::= <i>id = phi typ [val<sub>j</sub>, l<sub>j</sub>]<sup>j</sup></i>
Tmns	<i>tmn</i>	::= <b>br val l<sub>1</sub> l<sub>2</sub></b>   <b>br l</b>   <b>ret typ val</b>   <b>ret void</b>   <b>unreachable</b>
Commands	<i>c</i>	::= <i>id = bop (int sz) val<sub>1</sub> val<sub>2</sub></i>   <i>id = fbop fp val<sub>1</sub> val<sub>2</sub></i>   <b>store typ val<sub>1</sub> val<sub>2</sub> align</b>   <i>id = malloc typ val align</i>   <b>free (typ*) val</b>   <i>id = alloca typ val align</i>   <i>id = trop typ<sub>1</sub> val to typ<sub>2</sub></i>   <i>id = eop typ<sub>1</sub> val to typ<sub>2</sub></i>   <i>id = cop typ<sub>1</sub> val to typ<sub>2</sub></i>   <i>id = select val<sub>0</sub> typ val<sub>1</sub> val<sub>2</sub></i>   <b>option id = call typ<sub>0</sub> val<sub>0</sub> param</b>   <i>id = icmp cond typ val<sub>1</sub> val<sub>2</sub></i>   <i>id = fcmp fcond fp val<sub>1</sub> val<sub>2</sub></i>   <i>id = getelementptr (typ*) val val<sub>j</sub></i>   <i>id = load (typ*) val align</i>   <i>id = extractelement [sz × vtyp] val<sub>1</sub> val<sub>2</sub></i>   <i>id = insertelement [sz × vtyp] val<sub>1</sub> val<sub>2</sub> val<sub>3</sub></i>

Figure 4.1: Abstract syntax for a subset of LLVM.

## Chapter 5

# Abstract Interpretation

Define: context sensitivity context sensitivity lattice (infinite height due to recursion) path sensitivity path sensitivity lattice (infinite height due to loops) flow sensitivity

Call graph Call stack Operational fixpoint calculation. Equation-based fixpoint calculation.

Abstract interpreter can be either operational or equation-based. Our interpreter is operational.



# Chapter 6

## Memory Model

Memory abstraction appeared in [13].

Our memory abstraction for abstract interpretation recognizes four kinds of memory:

**Register-like stack memory** This is function-level memory that is released automatically when function returns. We denote such a memory by LLVM-style names starting with the percent sign %. Memory either has a name (e.g. %result) or a number is generated to serve as a name (e.g. %32 denotes thirty-second unnamed instruction call in a function).

**Stack memory allocated by alloca** This is also a function-level memory that is released automatically when function returns. The difference to register-like stack memory is that this memory is accessed by LLVM exclusively via pointers. We denote such a memory by names starting with %^. Every piece of memory has a name corresponding to the place where the memory has been allocated (alloca has been called). So if the memory has been allocated by an instruction call %ptr = alloca i32, align 4, it can be denoted by %^ptr.

**Global variables** Global variables are module-wise and are valid for the whole program run. We denote such a memory by LLVM-style names starting with @.

**Heap memory** Heap memory is also valid for the whole program run. We denote such a memory by names starting by @^. Every piece of memory has a name corresponding to the place where the memory has been allocated (malloc or similar function has been called). Name of the function is also included in the place name, so if a function createString contains an instruction call %result = call i8\* @malloc(i32 1), we can denote the memory allocated on this place by @^createString:result.

As it can be seen from the style of memory denotation, every piece of memory is associated with a place in the program. This means all operations affecting a memory block allocated at certain place forms a single abstract value. Context-sensite abstract interpretation helps to increase the precision of this memory abstraction.



## **Chapter 7**

# **Array Abstract Domains**



## **Chapter 8**

# **Structure Abstract Domain**



# Chapter 9

## Integer Abstract Domains

### 9.1 Integer Interval Domain $\mathcal{D}_i^\sharp$

The interval domain was first presented in [1]. It was particularly well described in [9]. More precise machine integer interval domain appeared in [16].

$$\mathcal{D}_i^\sharp \stackrel{\text{def}}{=} \{[l, h] \mid l, h \in \mathbb{Z} \cup \{\pm\infty\}\}$$

### 9.2 Integer Bitfield Domain $\mathcal{D}_b^\sharp$

Described in [16]. The domain associates two integers  $z$  and  $o$  to each variable. The integers represent bit masks for bits that can be set to 0 (zero) and to 1 (one).



## **Chapter 10**

# **Abstract Domains for Floating-Point Numbers**

Precise machine floating-point abstraction appeared in [16].



## Chapter 11

# Pointer Abstract Domains

Pointer can be casted to a number via the `ptrtoint` instruction. Usually, the resulting memory offset is used to achieve pointer arithmetics that are not available via `getelementptr` semantics.



## **Chapter 12**

# **Abstract Domain Combination**

Trees of abstract domains as done in ASTRÉE are described in [14].



# **Chapter 13**

## **Wishlist**

### **13.1 Reduced Product of Abstract Domains**

Including cooperation.

### **13.2 Widening and Narrowing**

Implement widening and narrowing operators for integers and other abstract domains as required.

### **13.3 String Abstract Domains**

Implement abstract domains specific for C strings.

### **13.4 Trace Partitioning**

Move context sensitivity to an abstract domain based on trace partitioning [11]. This change will allow us to introduce path-sensitivity just by extending this domain.

### **13.5 Boolean Partitioning**

### **13.6 Fixpoint Recalculation**

Allow to recompute fixpoint with a few variables changing their abstract domain layout.

### **13.7 Multi Threading**

Multi-threading abstraction for Abstract Interpretation appeared in [15].

## 13.8 Symbolic Abstract Domains

Mine. Symbolic Methods to Enhance the Precision of Numerical Abstract Domains.

## 13.9 Weakly-Relational Abstract Domains

Implement weakly relational integer and floating-point abstract domains.

## 13.10 Compositional Analysis

Analyze functions, modules, or libraries separately, and merge the results afterwards. Theory can be found in [4] and [6].

## 13.11 Parallelization

Make abstract interpreter to use multiple threads for fixpoint calculation on symmetric multiprocessor systems. See [12].

## 13.12 Custom Domains

Active user/group (uid/gid) domain, privileges domain. Opened files domain. Environment variables domain. File domain.

## **Part II**

# **Implementation**



# Chapter 14

## Overview

Canal can be used for a static analysis of real-world complex software systems written in efficient low-level languages C and C++. It uses the LLVM intermediate representation for the static analysis.

Canal is implemented in the C++ language as defined in the C++98 standard (ISO/IEC 14882:1998). It uses the C++ standard library and some additional libraries:

- LLVM core libraries. All versions from 2.8 up to 3.1 are supported.
- Clang compiler. Any version working with a supported version of LLVM should work.
- GNU readline. Any BSD-licensed reimplementation can be used as an alternative.
- elfutils. This library is used only on Linux-based operating systems.



# Chapter 15

## Library Class Index

### 15.1 Class Hierarchy

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

Canal::Interpreter::BasicBlock . . . . .	43
Canal::Constructors . . . . .	49
Canal::Widening::DataInterface . . . . .	53
Canal::Widening::DataIterationCount . . . . .	54
Canal::Environment . . . . .	58
Canal::Interpreter::Function . . . . .	63
Canal::FunctionModel . . . . .	64
Canal::FunctionModelManager . . . . .	64
Canal::Array::Interface . . . . .	65
Canal::Array::ExactSize . . . . .	59
Canal::Array::SingleItem . . . . .	97
Canal::Array::StringPrefix . . . . .	106
Canal::Structure . . . . .	111
Canal::Widening::Interface . . . . .	67
Canal::Widening::NumericalInfinity . . . . .	79
Canal::Widening::Pointers . . . . .	91
Canal::Interpreter::Interpreter . . . . .	67
Canal::Interpreter::Iterator . . . . .	76
Canal::Interpreter::IteratorCallback . . . . .	76
Canal::Widening::Manager . . . . .	77
MemoryOverlay . . . . .	78
Canal::Interpreter::Module . . . . .	78
Canal::Operations . . . . .	79
Canal::OperationsCallback . . . . .	85
Canal::Interpreter::OperationsCallback . . . . .	86
Canal::APIntUtils::SCompare . . . . .	91
Canal::SharedData . . . . .	96
Canal::Domain . . . . .	55
Canal::Array::ExactSize . . . . .	59
Canal::Array::SingleItem . . . . .	97
Canal::Array::StringPrefix . . . . .	106
Canal::Float::Interval . . . . .	73

Canal::Integer::Bitfield . . . . .	43
Canal::Integer::Container . . . . .	50
Canal::Integer::Interval . . . . .	68
Canal::Integer::Set . . . . .	92
Canal::Pointer::Pointer . . . . .	87
Canal::Structure . . . . .	111
Canal::SharedDataPointer< T > . . . . .	97
Canal::SlotTracker . . . . .	102
Canal::State . . . . .	103
Canal::StateMap . . . . .	105
Canal::StringStream . . . . .	110
Canal::Pointer::Target . . . . .	114
Canal::APIntUtils::UCompare . . . . .	116
Canal::Pointer::Utils . . . . .	116
Canal::VariableArguments . . . . .	117

## 15.2 Class List

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

Canal::Interpreter::BasicBlock . . . . .	43
Canal::Integer::Bitfield . . . . .	43
Canal::Constructors . . . . .	49
Canal::Integer::Container . . . . .	50
Canal::Widening::DataInterface . . . . .	53
Canal::Widening::DataIterationCount . . . . .	54
Canal::Domain Base class for all abstract domains . . . . .	55
Canal::Environment . . . . .	58
Canal::Array::ExactSize . . . . .	59
Canal::Interpreter::Function . . . . .	63
Canal::FunctionModel . . . . .	64
Canal::FunctionModelMAnager . . . . .	64
Canal::Array::Interface . . . . .	65
Canal::Widening::Interface . . . . .	67
Canal::Interpreter::Interpreter . . . . .	67
Canal::Integer::Interval Abstracts integer values as a interval min - max . . . . .	68
Canal::Float::Interval . . . . .	73
Canal::Interpreter::Iterator . . . . .	76
Canal::Interpreter::IteratorCallback . . . . .	76
Canal::Widening::Manager . . . . .	77
MemoryOverlay . . . . .	78
Canal::Interpreter::Module . . . . .	78
Canal::Widening::NumericalInfinity . . . . .	79
Canal::Operations . . . . .	79
Canal::OperationsCallback . . . . .	85
Canal::Interpreter::OperationsCallback . . . . .	86
Canal::Pointer::Pointer Inclusion-based flow-insensitive abstract pointer . . . . .	87
Canal::Widening::Pointers . . . . .	91
Canal::APIntUtils::SCompare . . . . .	91

Canal::Integer::Set . . . . .	92
Canal::SharedData . . . . .	96
Canal::SharedDataPointer< T > . . . . .	97
Canal::Array::SingleItem This array type is very imprecise . . . . .	97
Canal::SlotTracker . . . . .	102
Canal::State Abstract memory state . . . . .	103
Canal::StateMap . . . . .	105
Canal::Array::StringPrefix . . . . .	106
Canal::StringStream . . . . .	110
Canal::Structure . . . . .	111
Canal::Pointer::Target . . . . .	114
Canal::APIntUtils::UCompare . . . . .	116
Canal::Pointer::Utils . . . . .	116
Canal::VariableArguments . . . . .	117



# Chapter 16

## Library Class Documentation

### 16.1 Canal::Interpreter::BasicBlock Class Reference

#### Public Member Functions

- **BasicBlock** (const llvm::BasicBlock &basicBlock, const Constructors &constructors)
- const llvm::BasicBlock & **getLlvmBasicBlock** () const
- llvm::BasicBlock::const\_iterator **begin** () const
- llvm::BasicBlock::const\_iterator **end** () const
- State & **getInputState** ()
- State & **getOutputState** ()
- size\_t **memoryUsage** () const

*Get memory usage (used byte count) of this basic block interpretation.*
- std::string **toString** () const

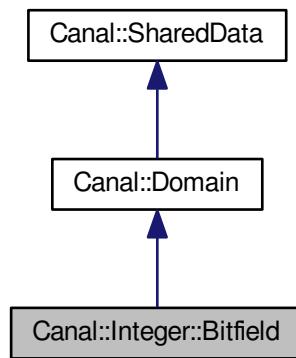
The documentation for this class was generated from the following files:

- lib/InterpreterBasicBlock.h
- lib/InterpreterBasicBlock.cpp

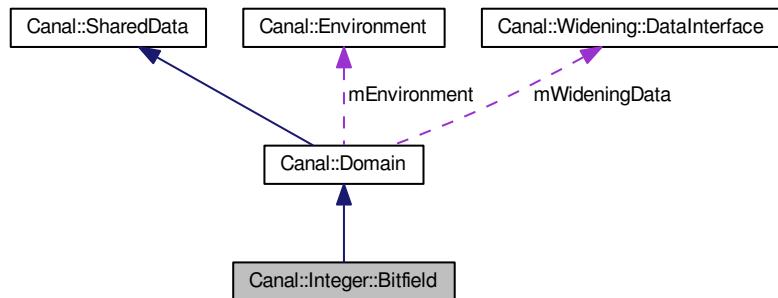
### 16.2 Canal::Integer::Bitfield Class Reference

```
#include <IntegerBitfield.h>
```

Inheritance diagram for Canal::Integer::Bitfield:



Collaboration diagram for Canal::Integer::Bitfield:



## Public Member Functions

- Bitfield (const Environment &environment, unsigned bitWidth)  
*Initializes to the lowest value.*
- Bitfield (const Environment &environment, const llvm::APInt &number)  
*Initializes to the given value.*
- Bitfield (const Bitfield &value)  
*Copy constructor.*
- unsigned getBitWidth () const  
*Return the number of bits of the represented number.*
- int getBitValue (unsigned pos) const

- void setBitValue (unsigned pos, int value)
- bool signedMin (llvm::APInt &result) const
- bool signedMax (llvm::APInt &result) const
- bool unsignedMin (llvm::APInt &result) const
- bool unsignedMax (llvm::APInt &result) const
- bool isConstant () const
  - Does these bits represent single value?*
- bool isTrue () const
  - Does the interval represent signle bit that is set to 1?*
- bool isFalse () const
  - Does the interval represent signle bit that is set to 0?*
- virtual Bitfield \* clone () const
  - Covariant return type.*
- virtual size\_t memoryUsage () const
  - Get memory usage (used byte count) of this abstract value.*
- virtual std::string toString () const
  - Create a string representation of the abstract value.*
- virtual void setZero (const llvm::Value \*place)
- virtual bool operator== (const Domain &value) const
- virtual bool operator< (const Domain &value) const
- virtual bool operator> (const Domain &value) const
- virtual Bitfield & join (const Domain &value)
  - Merge another value into this one.*
- virtual Bitfield & meet (const Domain &value)
- virtual bool isBottom () const
  - Is it the lowest value.*
- virtual void setBottom ()
  - Set to the lowest value.*
- virtual bool isTop () const
  - Is it the highest value.*
- virtual void setTop ()
  - Set it to the top value of lattice.*
- virtual float accuracy () const
- virtual Bitfield & add (const Domain &a, const Domain &b)
- virtual Bitfield & sub (const Domain &a, const Domain &b)
- virtual Bitfield & mul (const Domain &a, const Domain &b)
- virtual Bitfield & udiv (const Domain &a, const Domain &b)
  - Unsigned division.*
- virtual Bitfield & sdiv (const Domain &a, const Domain &b)
  - Signed division.*
- virtual Bitfield & urem (const Domain &a, const Domain &b)
- virtual Bitfield & srem (const Domain &a, const Domain &b)
- virtual Bitfield & shl (const Domain &a, const Domain &b)
- virtual Bitfield & lshr (const Domain &a, const Domain &b)
- virtual Bitfield & ash (const Domain &a, const Domain &b)
- virtual Bitfield & and\_ (const Domain &a, const Domain &b)
- virtual Bitfield & or\_ (const Domain &a, const Domain &b)
- virtual Bitfield & xor\_ (const Domain &a, const Domain &b)

- virtual Bitfield & icmp (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual Bitfield & **fcmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual Bitfield & **trunc** (const Domain &value)
- virtual Bitfield & **zext** (const Domain &value)
- virtual Bitfield & **sext** (const Domain &value)
- virtual Bitfield & **fptoui** (const Domain &value)
- virtual Bitfield & **fptosi** (const Domain &value)
- virtual bool **isValue** () const
- virtual const llvm::IntegerType & **getValueType** () const
 

*Covariant return type.*
- virtual bool **hasValueExactSize** () const
- virtual Domain \* **getValueCell** (uint64\_t offset) const
- virtual void **mergeValueCell** (uint64\_t offset, const Domain &value)

## Public Attributes

- llvm::APInt mZeroes
- llvm::APInt mOnes

## Additional Inherited Members

### 16.2.1 Detailed Description

Abstracts integers as a bitfield.

For every bit, we have 4 possible states:

#### mZeroes mOnes State

0 0 Nothing was set to the bit (lowest lattice value - bottom)  
 1 0 The bit is set to 0  
 0 1 The bit is set to 1  
 1 1 The bit can be both 0 and 1 (highest lattice value - top)

### 16.2.2 Member Function Documentation

#### 16.2.2.1 float Canal::Integer::Bitfield::accuracy ( ) const [virtual]

Get accuracy of the abstract value (0 - 1). In finite-height lattices, it is determined by the position of the value in the lattice.

Accuracy 0 means that the value represents all possible values (top). Accuracy 1 means that the value represents the most precise and exact value (bottom).

Reimplemented from Canal::Domain.

#### 16.2.2.2 int Canal::Integer::Bitfield::getBitValue ( unsigned pos ) const

Returns 0 if the bit is known to be 0. Returns 1 if the bit is known to be 1. Returns -1 if the bit value is unknown. Returns 2 if the bit is either 1 or 0.

---

**16.2.2.3 Bitfield & Canal::Integer::Bitfield::icmp ( const Domain & a, const Domain & b, llvm::CmpInst::Predicate *predicate* ) [virtual]**

$(A < B) \rightarrow A \geq B \rightarrow$  no useful information

$(A < B) \rightarrow A \geq B$

$(A > B) \rightarrow A \leq B \rightarrow$  no useful information

$(A > B) \rightarrow A \leq B$

$(A < B) \rightarrow A \geq B \rightarrow$  no useful information

$(A < B) \rightarrow A \geq B$

$(A > B) \rightarrow A \leq B \rightarrow$  no useful information

$(A > B) \rightarrow A \leq B$

Reimplemented from Canal::Domain.

**16.2.2.4 bool Canal::Integer::Bitfield::operator== ( const Domain & value ) const [virtual]**

Implementing this is mandatory. Values are compared while computing the fixed point.

Implements Canal::Domain.

**16.2.2.5 void Canal::Integer::Bitfield::setBitValue ( unsigned pos, int value )**

Sets the bit. If value is 0 or 1, the bit is set to represent exactly 0 or 1. If value is -1, the bit is set to represent unknown value. If value is 2, the bit is set to represent both 0 and 1.

**16.2.2.6 void Canal::Integer::Bitfield::setZero ( const llvm::Value \* place ) [virtual]**

Set value of this domain to represent zeroed memory. Needed for constants with zero initializer. This can only be called right after creating a domain.

Implements Canal::Domain.

**16.2.2.7 bool Canal::Integer::Bitfield::signedMax ( llvm::APInt & result ) const**

Highest signed number represented by this abstract domain.

**Parameters**

<i>result</i>	Filled by the maximum value if it is known. Otherwise, the value is undefined.
---------------	--

**Returns**

True if the result is known and the parameter was set to correct value.

**16.2.2.8 bool Canal::Integer::Bitfield::signedMin ( llvm::APInt & result ) const**

Lowest signed number represented by this abstract domain.

**Parameters**

<i>result</i>	Filled by the minimum value if it is known. Otherwise, the value is undefined.
---------------	--

**Returns**

True if the result is known and the parameter was set to correct value.

**16.2.2.9 bool Canal::Integer::Bitfield::unsignedMax ( llvm::APInt & *result* ) const**

Highest unsigned number represented by this abstract domain.

**Parameters**

<i>result</i>	Filled by the maximum value if it is known. Otherwise, the value is undefined.
---------------	--

**Returns**

True if the result is known and the parameter was set to correct value.

**16.2.2.10 bool Canal::Integer::Bitfield::unsignedMin ( llvm::APInt & *result* ) const**

Lowest unsigned number represented by this abstract domain.

**Parameters**

<i>result</i>	Filled by the minimum value if it is known. Otherwise, the value is undefined.
---------------	--

**Returns**

True if the result is known and the parameter was set to correct value.

**16.2.3 Member Data Documentation****16.2.3.1 llvm::APInt Canal::Integer::Bitfield::mOnes**

When a bit in mOnes is 1, the value is known to contain one at this position.

**16.2.3.2 llvm::APInt Canal::Integer::Bitfield::mZeroes**

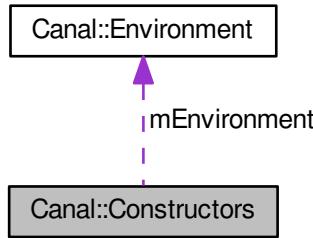
When a bit in mZeroes is 1, the value is known to contain zero at this position.

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

- lib/IntegerBitfield.h
- lib/IntegerBitfield.cpp

## 16.3 Canal::Constructors Class Reference

Collaboration diagram for Canal::Constructors:



### Public Member Functions

- **Constructors** (Environment &environment)
- const Environment & **getEnvironment** () const
- Domain \* **create** (const llvm::Type &type) const
- Domain \* **create** (const llvm::Constant &value, const llvm::Value &place, const State \*state) const
- Domain \* **createInteger** (unsigned bitWidth) const
- Domain \* **createInteger** (const llvm::APInt &number) const
- Domain \* **createFloat** (const llvm::fltSemantics &semantics) const
- Domain \* **createFloat** (const llvm::APFloat &number) const
- Domain \* **createArray** (const llvm::SequentialType &type) const
- Domain \* **createArray** (const llvm::SequentialType &type, Domain \*size) const
- Domain \* **createArray** (const llvm::SequentialType &type, const std::vector< Domain \* > &values) const
- Domain \* **createPointer** (const llvm::PointerType &type) const
- Domain \* **createStructure** (const llvm::StructType &type, const std::vector< Domain \* > &members) const

### Protected Member Functions

- Domain \* **createGetElementPtr** (const llvm::ConstantExpr &value, const Domain &variable, const llvm::Value &place) const
- Domain \* **createBitCast** (const llvm::ConstantExpr &value, const Domain &variable, const llvm::Value &place) const

### Protected Attributes

- const Environment & **mEnvironment**

### 16.3.1 Member Function Documentation

16.3.1.1 `Domain * Canal::Constructors::create ( const llvm::Constant & value, const llvm::Value & place, const State * state ) const`

#### Parameters

<code>state</code>	State is used only for constant expressions such as getelementptr and bitcast. For other types of constants it might be NULL.
--------------------	---

#### Returns

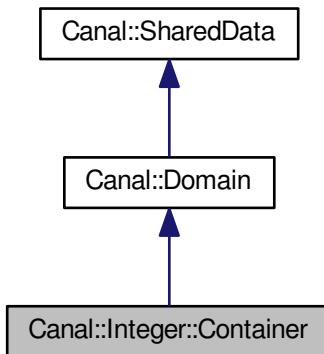
Returns a newly allocated value or NULL. Caller takes ownership of the returned value.

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

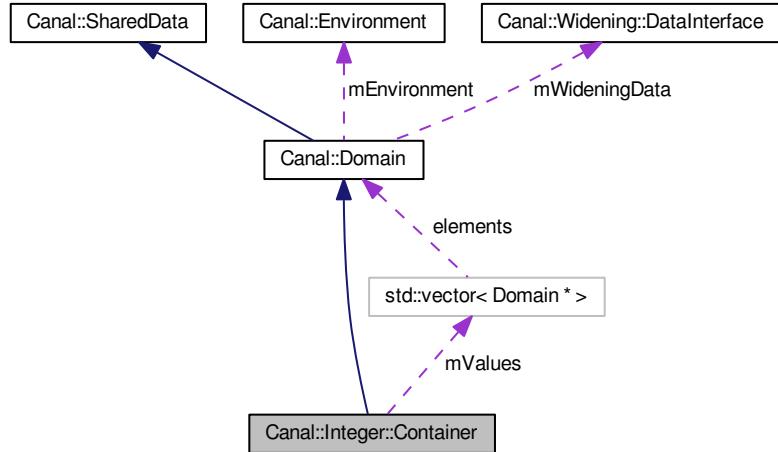
- lib/Constructors.h
- lib/Constructors.cpp

## 16.4 Canal::Integer::Container Class Reference

Inheritance diagram for Canal::Integer::Container:



Collaboration diagram for Canal::Integer::Container:



## Public Member Functions

- **Container** (const Environment &environment)
- Container (const Container &value)
 

*Copy constructor. Creates independent copy of the container.*
- virtual ~Container ()
 

*Destructor. Deletes the contents of the container.*
- virtual Container \* clone () const
 

*Covariant return type.*
- virtual size\_t memoryUsage () const
 

*Get memory usage (used byte count) of this abstract value.*
- virtual std::string toString () const
 

*Create a string representation of the abstract value.*
- virtual void setZero (const llvm::Value \*place)
- virtual bool operator== (const Domain &value) const
- virtual bool operator< (const Domain &value) const
- virtual bool operator> (const Domain &value) const
- virtual Container & join (const Domain &value)
 

*Merge another value into this one.*
- virtual Container & **meet** (const Domain &value)
- virtual bool isBottom () const
 

*Is it the lowest value.*
- virtual void setBottom ()
 

*Set to the lowest value.*
- virtual bool isTop () const
 

*Is it the highest value.*

- virtual void **setTop** ()
 

*Set it to the top value of lattice.*
- virtual float **accuracy** () const
- virtual Container & **add** (const Domain &a, const Domain &b)
- virtual Container & **sub** (const Domain &a, const Domain &b)
- virtual Container & **mul** (const Domain &a, const Domain &b)
- virtual Container & **udiv** (const Domain &a, const Domain &b)
 

*Unsigned division.*
- virtual Container & **sdiv** (const Domain &a, const Domain &b)
 

*Signed division.*
- virtual Container & **urem** (const Domain &a, const Domain &b)
- virtual Container & **srem** (const Domain &a, const Domain &b)
- virtual Container & **shl** (const Domain &a, const Domain &b)
- virtual Container & **lshr** (const Domain &a, const Domain &b)
- virtual Container & **ashr** (const Domain &a, const Domain &b)
- virtual Container & **and\_** (const Domain &a, const Domain &b)
- virtual Container & **or\_** (const Domain &a, const Domain &b)
- virtual Container & **xor\_** (const Domain &a, const Domain &b)
- virtual Container & **icmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual Container & **fcmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual Container & **trunc** (const Domain &value)
- virtual Container & **zext** (const Domain &value)
- virtual Container & **sext** (const Domain &value)
- virtual Container & **fptoui** (const Domain &value)
- virtual Container & **fptosi** (const Domain &value)
- virtual bool **isValue** () const
- virtual const llvm::Type & **getValueType** () const
- virtual bool **hasValueExactSize** () const
- virtual Domain \* **getValueAbstractSize** () const
- virtual Domain \* **getValueCell** (uint64\_t offset) const
- virtual void **mergeValueCell** (uint64\_t offset, const Domain &value)

## Public Attributes

- std::vector< Domain \* > **mValues**

## Additional Inherited Members

### 16.4.1 Member Function Documentation

#### 16.4.1.1 float Canal::Integer::Container::accuracy ( ) const [virtual]

Get accuracy of the abstract value (0 - 1). In finite-height lattices, it is determined by the position of the value in the lattice.

Accuracy 0 means that the value represents all possible values (top). Accuracy 1 means that the value represents the most precise and exact value (bottom).

Reimplemented from Canal::Domain.

#### 16.4.1.2 `bool Canal::Integer::Container::operator== ( const Domain & value ) const [virtual]`

Implementing this is mandatory. Values are compared while computing the fixed point.

Implements Canal::Domain.

#### 16.4.1.3 `void Canal::Integer::Container::setZero ( const llvm::Value * place ) [virtual]`

Set value of this domain to represent zeroed memory. Needed for constants with zero initializer. This can only be called right after creating a domain.

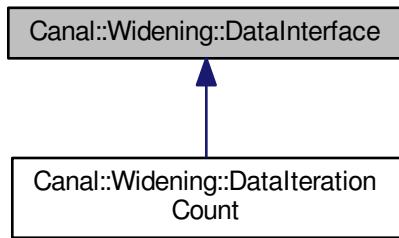
Implements Canal::Domain.

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

- lib/IntegerContainer.h
- lib/IntegerContainer.cpp

## 16.5 Canal::Widening::DataInterface Class Reference

Inheritance diagram for Canal::Widening::DataInterface:



### Public Member Functions

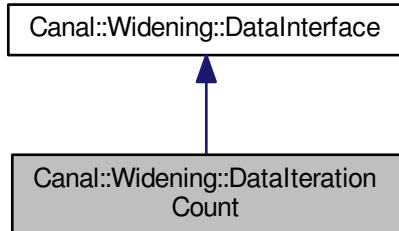
- `virtual DataInterface * clone () const =0`

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

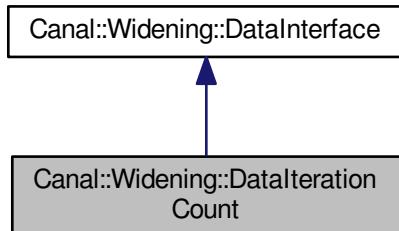
- lib/WideningDataInterface.h

## 16.6 Canal::Widening::DataIterationCount Class Reference

Inheritance diagram for Canal::Widening::DataIterationCount:



Collaboration diagram for Canal::Widening::DataIterationCount:



### Public Member Functions

- void **increase** (const llvm::BasicBlock &block)
- int **count** (const llvm::BasicBlock &block) const
- virtual DataIterationCount \* **clone** () const

### Static Public Attributes

- static int **ITERATION\_COUNT** = 2

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

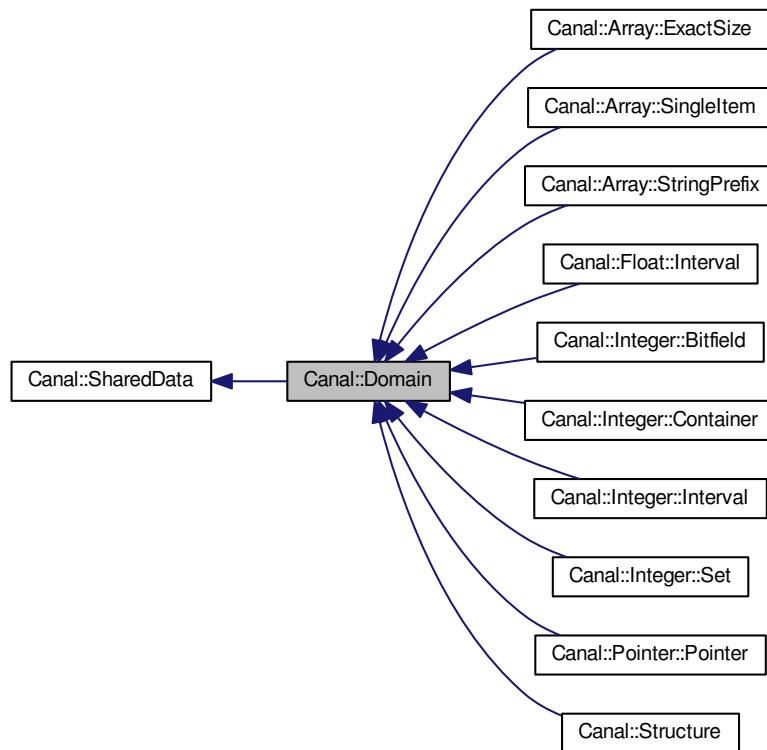
- lib/WideningDataIterationCount.h
- lib/WideningDataIterationCount.cpp

## 16.7 Canal::Domain Class Reference

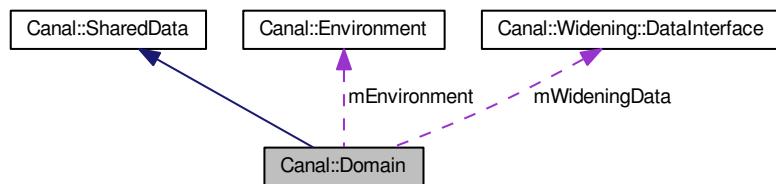
Base class for all abstract domains.

```
#include <Domain.h>
```

Inheritance diagram for Canal::Domain:



Collaboration diagram for Canal::Domain:



## Public Types

- `typedef Domain &(Domain::* CastOperation )(const Domain &)`
- `typedef Domain &(Domain::* BinaryOperation )(const Domain &, const Domain &)`
- `typedef Domain &(Domain::* CmpOperation )(const Domain &, const Domain &, llvm::CmpInst::Predicate predicate)`

## Public Member Functions

- `Domain (const Environment &environment)`  
*Standard constructor.*
- `Domain (const Domain &value)`
- `virtual ~Domain ()`  
*Virtual destructor.*
- `const Environment & getEnvironment () const`
- `virtual Domain * clone () const =0`  
*Create a copy of this value.*
- `virtual size_t memoryUsage () const =0`  
*Get memory usage (used byte count) of this abstract value.*
- `virtual std::string toString () const =0`  
*Create a string representation of the abstract value.*
- `virtual void setZero (const llvm::Value *place)=0`
- `virtual bool operator==(const Domain &value) const =0`
- `virtual bool operator!=(const Domain &value) const`  
*Inequality is implemented by calling the equality operator.*
- `virtual bool operator< (const Domain &value) const =0`
- `virtual bool operator<= (const Domain &value) const`
- `virtual bool operator> (const Domain &value) const =0`
- `virtual bool operator>= (const Domain &value) const`
- `virtual Domain & join (const Domain &value)=0`  
*Merge another value into this one.*
- `virtual Domain & meet (const Domain &value)=0`
- `virtual bool isBottom () const`  
*Is it the lowest value.*
- `virtual void setBottom ()`  
*Set to the lowest value.*
- `virtual bool isTop () const`  
*Is it the highest value.*
- `virtual void setTop ()`  
*Set it to the top value of lattice.*
- `virtual float accuracy () const`
- `virtual Domain & add (const Domain &a, const Domain &b)`
- `virtual Domain & fadd (const Domain &a, const Domain &b)`
- `virtual Domain & sub (const Domain &a, const Domain &b)`
- `virtual Domain & fsub (const Domain &a, const Domain &b)`
- `virtual Domain & mul (const Domain &a, const Domain &b)`
- `virtual Domain & fmul (const Domain &a, const Domain &b)`
- `virtual Domain & udiv (const Domain &a, const Domain &b)`

*Unsigned division.*

- virtual Domain & sdiv (const Domain &a, const Domain &b)

*Signed division.*

- virtual Domain & fdiv (const Domain &a, const Domain &b)

*Floating point division.*

- virtual Domain & **urem** (const Domain &a, const Domain &b)
- virtual Domain & **srem** (const Domain &a, const Domain &b)
- virtual Domain & **frem** (const Domain &a, const Domain &b)
- virtual Domain & **shl** (const Domain &a, const Domain &b)
- virtual Domain & **lshr** (const Domain &a, const Domain &b)
- virtual Domain & **ashr** (const Domain &a, const Domain &b)
- virtual Domain & **and\_** (const Domain &a, const Domain &b)
- virtual Domain & **or\_** (const Domain &a, const Domain &b)
- virtual Domain & **xor\_** (const Domain &a, const Domain &b)
- virtual Domain & **icmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual Domain & **fcmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual Domain & **trunc** (const Domain &value)
- virtual Domain & **zext** (const Domain &value)
- virtual Domain & **sext** (const Domain &value)
- virtual Domain & **fptrunc** (const Domain &value)
- virtual Domain & **fpext** (const Domain &value)
- virtual Domain & **fptoui** (const Domain &value)
- virtual Domain & **fptosi** (const Domain &value)
- virtual Domain & **uitofp** (const Domain &value)
- virtual Domain & **sitofp** (const Domain &value)
- Widening::DataInterface \* **getWideningData** () const
- void setWideningData (Widening::DataInterface \*wideningData)

*This class takes ownership of the wideningData memory.*

- virtual bool **isValue** () const
- virtual const llvm::Type & **getValueType** () const
- virtual bool **hasValueExactSize** () const
- virtual uint64\_t **getValueExactSize** ()
- virtual Domain \* **getValueAbstractSize** () const
- virtual Domain \* **getValueCell** (uint64\_t offset) const
- virtual void **mergeValueCell** (uint64\_t offset, const Domain &value)

## Public Attributes

- const Environment & **mEnvironment**
- Widening::DataInterface \* **mWideningData**

### 16.7.1 Detailed Description

Base class for all abstract domains.

## 16.7.2 Constructor & Destructor Documentation

### 16.7.2.1 `Canal::Domain::Domain ( const Domain & value )`

Copy constructor. Careful! Copy constructor of base class is not called by automatically generated copy constructor of an inherited class.

## 16.7.3 Member Function Documentation

### 16.7.3.1 `float Canal::Domain::accuracy ( ) const [virtual]`

Get accuracy of the abstract value (0 - 1). In finite-height lattices, it is determined by the position of the value in the lattice.

Accuracy 0 means that the value represents all possible values (top). Accuracy 1 means that the value represents the most precise and exact value (bottom).

Reimplemented in `Canal::Integer::Interval`, `Canal::Integer::Bitfield`, `Canal::Integer::Set`, `Canal::Array::ExactSize`, `Canal::Array::SingleItem`, `Canal::Float::Interval`, `Canal::Array::StringPrefix`, `Canal::Integer::Container`, and `Canal::Structure`.

### 16.7.3.2 `virtual bool Canal::Domain::operator== ( const Domain & value ) const [pure virtual]`

Implementing this is mandatory. Values are compared while computing the fixed point.

Implemented in `Canal::Integer::Interval`, `Canal::Pointer::Pointer`, `Canal::Integer::Bitfield`, `Canal::Integer::Set`, `Canal::Float::Interval`, `Canal::Array::ExactSize`, `Canal::Array::SingleItem`, `Canal::Array::StringPrefix`, `Canal::Integer::Container`, and `Canal::Structure`.

### 16.7.3.3 `virtual void Canal::Domain::setZero ( const llvm::Value * place ) [pure virtual]`

Set value of this domain to represent zeroed memory. Needed for constants with zero initializer. This can only be called right after creating a domain.

Implemented in `Canal::Integer::Interval`, `Canal::Pointer::Pointer`, `Canal::Integer::Bitfield`, `Canal::Integer::Set`, `Canal::Float::Interval`, `Canal::Array::ExactSize`, `Canal::Array::SingleItem`, `Canal::Array::StringPrefix`, `Canal::Integer::Container`, and `Canal::Structure`.

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

- lib/Domain.h
- lib/Domain.cpp

## 16.8 `Canal::Environment` Class Reference

### Public Member Functions

- **Environment** (`llvm::Module *module`)
- `llvm::LLVMContext & getContext () const`
- `llvm::Module & getModule () const`
- `SlotTracker & getSlotTracker () const`
- `const Constructors & getConstructors () const`

- void **setConstructors** (Constructors \*constructors)
- uint64\_t **getTypeStoreSize** (const llvm::Type &type) const

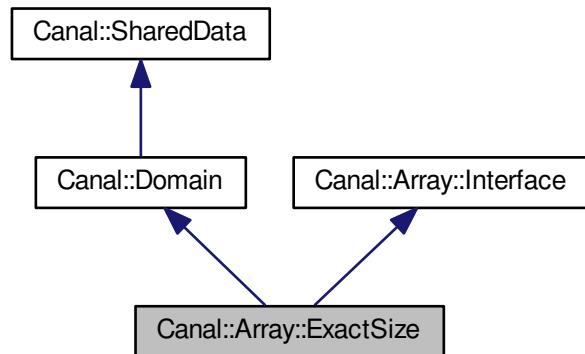
The documentation for this class was generated from the following files:

- lib/Environment.h
- lib/Environment.cpp

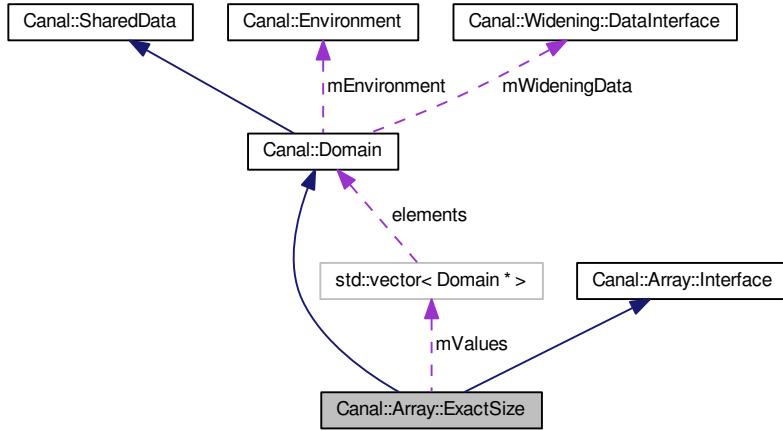
## 16.9 Canal::Array::ExactSize Class Reference

```
#include <ArrayExactSize.h>
```

Inheritance diagram for Canal::Array::ExactSize:



Collaboration diagram for Canal::Array::ExactSize:



## Public Member Functions

- `ExactSize (const Environment &environment, const llvm::SequentialType &type)`
- `ExactSize (const Environment &environment, const llvm::SequentialType &type, const std::vector< Domain * > &values)`
- **`ExactSize (const Environment &environment, const llvm::SequentialType &type, Domain *size)`**
- `ExactSize (const ExactSize &value)`

*Copy constructor.*

- `size_t size () const`
- `virtual ExactSize * clone () const`

*Covariant return type.*

- `virtual size_t memoryUsage () const`

*Get memory usage (used byte count) of this abstract value.*

- `virtual std::string toString () const`

*Create a string representation of the abstract value.*

- `virtual void setZero (const llvm::Value *place)`
- `virtual bool operator== (const Domain &value) const`
- `virtual bool operator< (const Domain &value) const`
- `virtual bool operator> (const Domain &value) const`
- `virtual ExactSize & join (const Domain &value)`

*Merge another value into this one.*

- `virtual ExactSize & meet (const Domain &value)`
- `virtual bool isBottom () const`

*Check if all items in the array are bottom.*

- `virtual void setBottom ()`

*Set all items in the array to bottom.*

- `virtual bool isTop () const`

*Check if all items in the array are top.*

- virtual void **setTop** ()
 

*Set all items in the array to top.*
- virtual float **accuracy** () const
- virtual ExactSize & **add** (const Domain &a, const Domain &b)
- virtual ExactSize & **fadd** (const Domain &a, const Domain &b)
- virtual ExactSize & **sub** (const Domain &a, const Domain &b)
- virtual ExactSize & **fsub** (const Domain &a, const Domain &b)
- virtual ExactSize & **mul** (const Domain &a, const Domain &b)
- virtual ExactSize & **fmul** (const Domain &a, const Domain &b)
- virtual ExactSize & **udiv** (const Domain &a, const Domain &b)
 

*Unsigned division.*
- virtual ExactSize & **sdiv** (const Domain &a, const Domain &b)
 

*Signed division.*
- virtual ExactSize & **fdiv** (const Domain &a, const Domain &b)
 

*Floating point division.*
- virtual ExactSize & **urem** (const Domain &a, const Domain &b)
- virtual ExactSize & **srem** (const Domain &a, const Domain &b)
- virtual ExactSize & **frem** (const Domain &a, const Domain &b)
- virtual ExactSize & **shl** (const Domain &a, const Domain &b)
- virtual ExactSize & **lshr** (const Domain &a, const Domain &b)
- virtual ExactSize & **ashr** (const Domain &a, const Domain &b)
- virtual ExactSize & **and\_** (const Domain &a, const Domain &b)
- virtual ExactSize & **or\_** (const Domain &a, const Domain &b)
- virtual ExactSize & **xor\_** (const Domain &a, const Domain &b)
- virtual ExactSize & **icmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual ExactSize & **fcmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual bool **isValue** () const
- virtual const
 

llvm::SequentialType & **getValueType** () const

*Covariant return type.*
- virtual bool **hasValueExactSize** () const
- virtual Domain \* **getValueCell** (uint64\_t offset) const
- virtual void **mergeValueCell** (uint64\_t offset, const Domain &value)
- virtual std::vector< Domain \* > **getItem** (const Domain &offset) const
- virtual Domain \* **getItem** (uint64\_t offset) const
- virtual void **setItem** (const Domain &offset, const Domain &value)
- virtual void **setItem** (uint64\_t offset, const Domain &value)

## Public Attributes

- std::vector< Domain \* > mValues
 

*Empty array indicates the top value.*
- const llvm::SequentialType & mType
 

*The type of the arrayx.*

## Additional Inherited Members

### 16.9.1 Detailed Description

Array with exact size and limited length. It keeps all array members separately, not losing precision at all.

## 16.9.2 Constructor & Destructor Documentation

### 16.9.2.1 `Canal::Array::ExactSize::ExactSize ( const Environment & environment, const IIR::SequentialType & type )`

Standard constructor.

If the type is suitable for exact size array, bottom domains of element type are created in mValues.

### 16.9.2.2 `Canal::Array::ExactSize::ExactSize ( const Environment & environment, const IIR::SequentialType & type, const std::vector< Domain * > & values )`

#### Parameters

<code>values</code>	This class takes ownership of the values.
---------------------	---

## 16.9.3 Member Function Documentation

### 16.9.3.1 `float Canal::Array::ExactSize::accuracy ( ) const [virtual]`

Get accuracy of the abstract value (0 - 1). In finite-height lattices, it is determined by the position of the value in the lattice.

Accuracy 0 means that the value represents all possible values (top). Accuracy 1 means that the value represents the most precise and exact value (bottom).

Reimplemented from `Canal::Domain`.

### 16.9.3.2 `std::vector< Domain * > Canal::Array::ExactSize::getItem ( const Domain & offset ) const [virtual]`

Get the array items pointed by the provided offset. Returns internal array items that are owned by the array. Caller must not delete the items.

Implements `Canal::Array::Interface`.

### 16.9.3.3 `Domain * Canal::Array::ExactSize::getItem ( uint64_t offset ) const [virtual]`

Get the array item pointed by the provided offset. Returns internal array item that is owned by the array. Caller must not delete the item.

#### Note

The `uint64_t` offset variant exists because of the `extractvalue` instruction, which provides exact numeric offsets.

For future array domains it might be necessary to extend this method to return a list of values.

Implements `Canal::Array::Interface`.

### 16.9.3.4 `bool Canal::Array::ExactSize::operator== ( const Domain & value ) const [virtual]`

Implementing this is mandatory. Values are compared while computing the fixed point.

Implements `Canal::Domain`.

**16.9.3.5 void Canal::Array::ExactSize::setItem ( const Domain & offset, const Domain & value ) [virtual]**

**Parameters**

<i>value</i>	The method does not take the ownership of this memory. It copies the contents of the value instead.
--------------	---

Implements Canal::Array::Interface.

**16.9.3.6 void Canal::Array::ExactSize::setItem ( uint64\_t offset, const Domain & value ) [virtual]**

**Parameters**

<i>value</i>	The method does not take the ownership of this memory. It copies the contents of the value instead.
--------------	---

**Note**

The uint64\_t offset variant exists because of the insertvalue instruction, which provides exact numeric offsets.

Implements Canal::Array::Interface.

**16.9.3.7 void Canal::Array::ExactSize::setZero ( const llvm::Value \* place ) [virtual]**

Set value of this domain to represent zeroed memory. Needed for constants with zero initializer. This can only be called right after creating a domain.

Implements Canal::Domain.

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

- lib/ArrayExactSize.h
- lib/ArrayExactSize.cpp

## 16.10 Canal::Interpreter::Function Class Reference

### Public Member Functions

- **Function** (const llvm::Function &function, const Constructors &constructors)
- const llvm::Function & **getLlvmFunction** () const
- const llvm::BasicBlock & **getLlvmEntryBlock** () const
- BasicBlock & **getBasicBlock** (const llvm::BasicBlock &llvmBasicBlock) const
- std::vector< BasicBlock \* > ::const\_iterator **begin** () const
- std::vector< BasicBlock \* > ::const\_iterator **end** () const
- State & **getInputState** ()
- const State & **getInputState** () const
- const State & **getOutputState** () const

- `llvm::StringRef getName () const`
- `void initializeInputState (BasicBlock &basicBlock, State &state) const`
- `void updateOutputState ()`  
*Update function output state from basic block output states.*
- `size_t memoryUsage () const`  
*Get memory usage (used byte count) of this function interpretation.*
- `std::string toString () const`

### 16.10.1 Member Function Documentation

#### 16.10.1.1 `void Canal::Interpreter::Function::initializeInputState ( BasicBlock & basicBlock, State & state ) const`

Update basic block input state from its predecessors and function input state.

##### Parameters

<code>basicBlock</code>	Must be a member of this function. Its input state is updated.
-------------------------	--

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

- `lib/InterpreterFunction.h`
- `lib/InterpreterFunction.cpp`

## 16.11 Canal::FunctionModel Class Reference

### Public Member Functions

- `bool canHandle (llvm::Function *function, bool implementationAvailable)`
- `void handle (llvm::Function *function, State &state)`

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

- `lib/FunctionModel.h`

## 16.12 Canal::FunctionModelManager Class Reference

### Public Member Functions

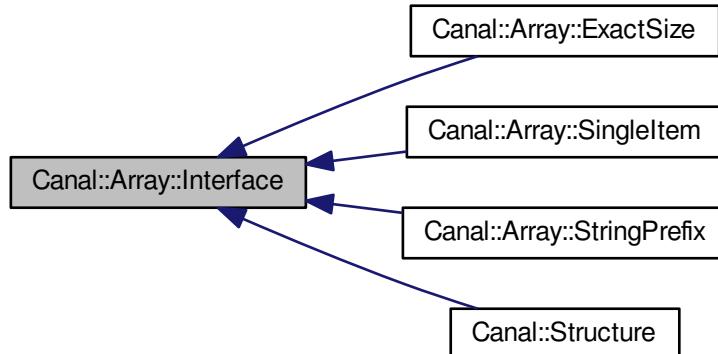
- `bool canHandle (llvm::Function *function, bool implementationAvailable)`
- `void handle (llvm::Function *function, State &state)`

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

- `lib/FunctionModelManager.h`

## 16.13 Canal::Array::Interface Class Reference

Inheritance diagram for Canal::Array::Interface:



### Public Member Functions

- Domain \* getValue (const Domain &offset) const
- Domain \* getValue (uint64\_t offset) const
- virtual std::vector< Domain \* > getItem (const Domain &offset) const =0
- virtual Domain \* getItem (uint64\_t offset) const =0
- virtual void setItem (const Domain &offset, const Domain &value)=0
- virtual void setItem (uint64\_t offset, const Domain &value)=0

#### 16.13.1 Member Function Documentation

##### 16.13.1.1 `virtual std::vector<Domain*> Canal::Array::Interface::getItem ( const Domain & offset ) const [pure virtual]`

Get the array items pointed by the provided offset. Returns internal array items that are owned by the array. Caller must not delete the items.

Implemented in Canal::Array::ExactSize, Canal::Array::SingleItem, Canal::Array::StringPrefix, and Canal::Structure.

##### 16.13.1.2 `virtual Domain* Canal::Array::Interface::getItem ( uint64_t offset ) const [pure virtual]`

Get the array item pointed by the provided offset. Returns internal array item that is owned by the array. Caller must not delete the item.

**Note**

The `uint64_t` offset variant exists because of the `extractvalue` instruction, which provides exact numeric offsets.

For future array domains it might be necessary to extend this method to return a list of values.

Implemented in `Canal::Array::ExactSize`, `Canal::Array::SingleItem`, `Canal::Array::StringPrefix`, and `Canal::Structure`.

**16.13.1.3 Domain \* Canal::Array::Interface::getValue ( const Domain & offset ) const**

Gets the value representing the array item or items pointed by the provided offset. Caller is responsible for deleting the returned value.

**16.13.1.4 Domain\* Canal::Array::Interface::getValue ( uint64\_t offset ) const**

Gets the value representing the array item pointed by the provided offset. Caller is responsible for deleting the returned value.

**Note**

The `uint64_t` offset variant exists because of the `extractvalue` instruction, which provides exact numeric offsets.

**16.13.1.5 virtual void Canal::Array::Interface::setItem ( const Domain & offset, const Domain & value ) [pure virtual]****Parameters**

<i>value</i>	The method does not take the ownership of this memory. It copies the contents of the value instead.
--------------	---

Implemented in `Canal::Array::ExactSize`, `Canal::Array::SingleItem`, `Canal::Array::StringPrefix`, and `Canal::Structure`.

**16.13.1.6 virtual void Canal::Array::Interface::setItem ( uint64\_t offset, const Domain & value ) [pure virtual]****Parameters**

<i>value</i>	The method does not take the ownership of this memory. It copies the contents of the value instead.
--------------	---

**Note**

The `uint64_t` offset variant exists because of the `insertvalue` instruction, which provides exact numeric offsets.

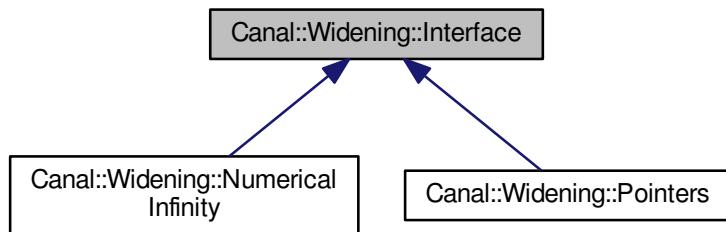
Implemented in `Canal::Array::ExactSize`, `Canal::Array::SingleItem`, `Canal::Array::StringPrefix`, and `Canal::Structure`.

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

- lib/ArrayInterface.h
- lib/ArrayInterface.cpp

## 16.14 Canal::Widening::Interface Class Reference

Inheritance diagram for Canal::Widening::Interface:



### Public Member Functions

- virtual void **widen** (const llvm::BasicBlock &wideningPoint, Domain &first, const Domain &second)=0

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

- lib/WideningInterface.h

## 16.15 Canal::Interpreter::Interpreter Class Reference

### Public Member Functions

- Interpreter (llvm::Module \*module)
- const Environment & **getEnvironment** () const
- SlotTracker & **getSlotTracker** () const
- const Constructors & **getConstructors** () const
- const Module & **getModule** () const
- const Operations & **getOperations** () const
- Iterator & **getIterator** ()
- const Iterator & **getIterator** () const
- const State & **getCurrentState** () const
- const Function & **getCurrentFunction** () const
- const BasicBlock & **getCurrentBasicBlock** () const
- const llvm::Instruction & **getCurrentInstruction** () const
- std::string **toString** () const

### 16.15.1 Constructor & Destructor Documentation

#### 16.15.1.1 `Canal::Interpreter::Interpreter ( llvm::Module * module )`

##### Parameters

<i>module</i>	Interpreter takes ownership of the module.
---------------	--

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

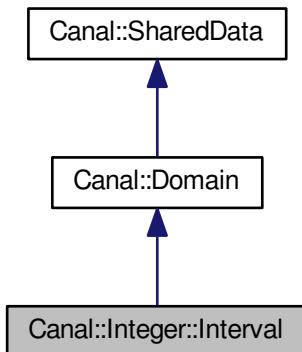
- lib/Interpreter.h
- lib/Interpreter.cpp

## 16.16 Canal::Integer::Interval Class Reference

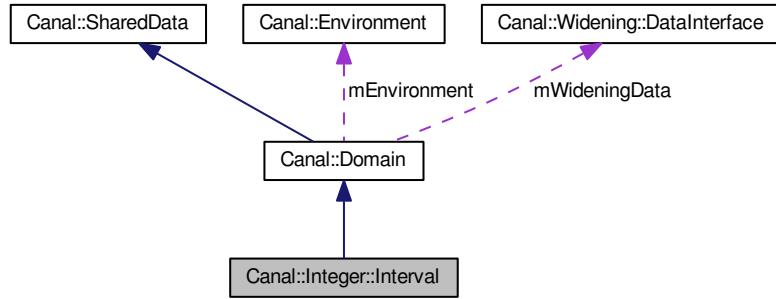
Abstracts integer values as a interval min - max.

```
#include <IntegerInterval.h>
```

Inheritance diagram for Canal::Integer::Interval:



Collaboration diagram for Canal::Integer::Interval:



## Public Member Functions

- `Interval (const Environment &environment, unsigned bitWidth)`  
*Standard constructor.*
- `Interval (const Environment &environment, const llvm::APInt &constant)`  
*Standard constructor.*
- `Interval (const Environment &environment, const llvm::APInt &from, const llvm::APInt &to)`  
*Standard constructor.*
- `Interval (const Interval &value)`  
*Copy constructor.*
- `unsigned getBitWidth () const`
- `bool signedMin (llvm::APInt &result) const`
- `bool signedMax (llvm::APInt &result) const`
- `bool unsignedMin (llvm::APInt &result) const`
- `bool unsignedMax (llvm::APInt &result) const`
- `bool isConstant () const`
- `bool isSignedConstant () const`  
*Returns true if the interval represents a signed single value.*
- `bool isUnsignedConstant () const`  
*Returns true if the interval represents a unsigned single value.*
- `bool isTrue () const`  
*Does the interval represent signle bit that is set to 1?*
- `bool isFalse () const`  
*Does the interval represent signle bit that is set to 0?*
- `virtual Interval * clone () const`  
*Covariant return type.*
- `virtual size_t memoryUsage () const`  
*Get memory usage (used byte count) of this abstract value.*
- `virtual std::string toString () const`  
*Create a string representation of the abstract value.*
- `virtual void setZero (const llvm::Value *place)`

- virtual bool operator== (const Domain &value) const
- virtual bool **operator<** (const Domain &value) const
- virtual bool **operator>** (const Domain &value) const
- virtual Interval & join (const Domain &value)

*Merge another value into this one.*

- virtual Interval & **meet** (const Domain &value) const
- virtual bool isBottom () const

*Is it the lowest value.*

- virtual void setBottom ()

*Set to the lowest value.*

- virtual bool isTop () const

*Is it the highest value.*

- virtual void setTop ()

*Set it to the top value of lattice.*

- virtual float accuracy () const
- virtual Interval & **add** (const Domain &a, const Domain &b)
- virtual Interval & **sub** (const Domain &a, const Domain &b)
- virtual Interval & **mul** (const Domain &a, const Domain &b)
- virtual Interval & **udiv** (const Domain &a, const Domain &b)

*Unsigned division.*

- virtual Interval & **sdiv** (const Domain &a, const Domain &b)

*Signed division.*

- virtual Interval & **urem** (const Domain &a, const Domain &b)
- virtual Interval & **srem** (const Domain &a, const Domain &b)
- virtual Interval & **shl** (const Domain &a, const Domain &b)
- virtual Interval & **lshr** (const Domain &a, const Domain &b)
- virtual Interval & **ashr** (const Domain &a, const Domain &b)
- virtual Interval & **and\_** (const Domain &a, const Domain &b)
- virtual Interval & **or\_** (const Domain &a, const Domain &b)
- virtual Interval & **xor\_** (const Domain &a, const Domain &b)
- virtual Interval & **icmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual Interval & **fcmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual Interval & **trunc** (const Domain &value)
- virtual Interval & **zext** (const Domain &value)
- virtual Interval & **sext** (const Domain &value)
- virtual Interval & **fptoui** (const Domain &value)
- virtual Interval & **fptosi** (const Domain &value)
- virtual bool **isValue** () const
- virtual const llvm::IntegerType & **getValueType** () const

*Covariant return type.*

- virtual bool **hasValueExactSize** () const
- virtual Domain \* **getValueCell** (uint64\_t offset) const
- virtual void **mergeValueCell** (uint64\_t offset, const Domain &value)

## Public Attributes

- bool mEmpty  
*Indicates an empty interval.*
- bool mSignedTop
- llvm::APInt mSignedFrom  
*The number is included in the interval.*
- llvm::APInt mSignedTo  
*The number is included in the interval.*
- bool mUnsignedTop
- llvm::APInt mUnsignedFrom  
*The number is included in the interval.*
- llvm::APInt mUnsignedTo  
*The number is included in the interval.*

## Additional Inherited Members

### 16.16.1 Detailed Description

Abstracts integer values as a interval min - max.

### 16.16.2 Constructor & Destructor Documentation

#### 16.16.2.1 Canal::Integer::Interval ( const Environment & environment, unsigned bitWidth )

Standard constructor.

Initializes an empty interval.

### 16.16.3 Member Function Documentation

#### 16.16.3.1 float Canal::Integer::Interval::accuracy ( ) const [virtual]

Get accuracy of the abstract value (0 - 1). In finite-height lattices, it is determined by the position of the value in the lattice.

Accuracy 0 means that the value represents all possible values (top). Accuracy 1 means that the value represents the most precise and exact value (bottom).

Reimplemented from Canal::Domain.

#### 16.16.3.2 bool Canal::Integer::Interval::isConstant ( ) const

Returns true if the interval represents a single number. Signed and unsigned representations might differ, though.

#### 16.16.3.3 bool Canal::Integer::Interval::operator== ( const Domain & value ) const [virtual]

Implementing this is mandatory. Values are compared while computing the fixed point.

Implements Canal::Domain.

**16.16.3.4 void Canal::Integer::Interval::setZero ( const llvm::Value \* *place* ) [virtual]**

Set value of this domain to represent zeroed memory. Needed for constants with zero initializer. This can only be called right after creating a domain.

Implements Canal::Domain.

**16.16.3.5 bool Canal::Integer::Interval::signedMax ( llvm::APInt & *result* ) const**

Highest signed number represented by this abstract domain.

**Parameters**

<i>result</i>	Filled by the maximum value if it is known. Otherwise, the value is undefined.
---------------	--

**Returns**

True if the result is known and the parameter was set to correct value.

**16.16.3.6 bool Canal::Integer::Interval::signedMin ( llvm::APInt & *result* ) const**

Lowest signed number represented by this abstract domain.

**Parameters**

<i>result</i>	Filled by the minimum value if it is known. Otherwise, the value is undefined.
---------------	--

**Returns**

True if the result is known and the parameter was set to correct value.

**16.16.3.7 bool Canal::Integer::Interval::unsignedMax ( llvm::APInt & *result* ) const**

Highest unsigned number represented by this abstract domain.

**Parameters**

<i>result</i>	Filled by the maximum value if it is known. Otherwise, the value is undefined.
---------------	--

**Returns**

True if the result is known and the parameter was set to correct value.

**16.16.3.8 bool Canal::Integer::Interval::unsignedMin ( llvm::APInt & *result* ) const**

Lowest unsigned number represented by this abstract domain.

**Parameters**

<i>result</i>	Filled by the minimum value if it is known. Otherwise, the value is undefined.
---------------	--

**Returns**

True if the result is known and the parameter was set to correct value.

## 16.16.4 Member Data Documentation

### 16.16.4.1 bool Canal::Integer::Interval::mEmpty

Indicates an empty interval.

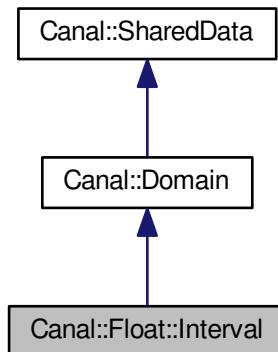
When it is set to true, other members' values are not considered as valid.

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

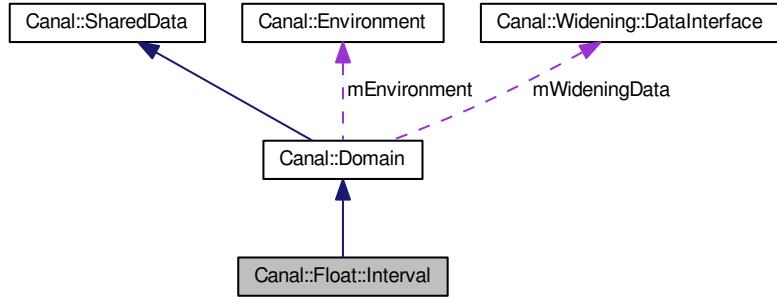
- lib/IntegerInterval.h
- lib/IntegerInterval.cpp

## 16.17 Canal::Float::Interval Class Reference

Inheritance diagram for Canal::Float::Interval:



Collaboration diagram for Canal::Float::Interval:



## Public Member Functions

- **Interval** (const Environment &environment, const llvm::fltSemantics &semantics)
  - **Interval** (const Environment &environment, const llvm::APFloat &constant)
  - **Interval** (const Environment &environment, const llvm::APFloat &from, const llvm::APFloat &to)
  - **Interval** (const Interval &value)
- Copy constructor.*
- int **compare** (const Interval &value, llvm::CmpInst::Predicate predicate) const
  - bool **isNaN** () const
  - const llvm::fltSemantics & **getSemantics** () const
  - bool **isConstant** () const
  - bool **intersects** (const Interval &value) const
  - llvm::APFloat **getMax** () const
  - llvm::APFloat **getMin** () const
  - virtual Interval \* **clone** () const
- Create a copy of this value.*
- virtual size\_t **memoryUsage** () const
- Get memory usage (used byte count) of this abstract value.*
- virtual std::string **toString** () const
- Create a string representation of the abstract value.*
- virtual void **setZero** (const llvm::Value \*place)
  - virtual bool **operator==** (const Domain &value) const
  - virtual bool **operator<** (const Domain &value) const
  - virtual bool **operator>** (const Domain &value) const
  - virtual Interval & **join** (const Domain &value)
- Merge another value into this one.*
- virtual Interval & **meet** (const Domain &value)
  - virtual bool **isBottom** () const
- Is it the lowest value.*
- virtual void **setBottom** ()
- Set to the lowest value.*

- virtual bool **isTop** () const  
*Is it the highest value.*
- virtual void **setTop** ()  
*Set it to the top value of lattice.*
- virtual float **accuracy** () const
- virtual Interval & **fadd** (const Domain &a, const Domain &b)
- virtual Interval & **fsub** (const Domain &a, const Domain &b)
- virtual Interval & **fmul** (const Domain &a, const Domain &b)
- virtual Interval & **fdiv** (const Domain &a, const Domain &b)  
*Floating point division.*
- virtual Interval & **frem** (const Domain &a, const Domain &b)
- virtual Interval & **uitofp** (const Domain &value)
- virtual Interval & **sitofp** (const Domain &value)
- virtual bool **isValue** () const
- virtual const llvm::Type & **getValueType** () const
- virtual bool **hasValueExactSize** () const
- virtual Domain \* **getValueCell** (uint64\_t offset) const
- virtual void **mergeValueCell** (uint64\_t offset, const Domain &value)

## Additional Inherited Members

### 16.17.1 Member Function Documentation

#### 16.17.1.1 float Canal::Float::Interval::accuracy ( ) const [virtual]

Get accuracy of the abstract value (0 - 1). In finite-height lattices, it is determined by the position of the value in the lattice.

Accuracy 0 means that the value represents all possible values (top). Accuracy 1 means that the value represents the most precise and exact value (bottom).

Reimplemented from Canal::Domain.

#### 16.17.1.2 bool Canal::Float::Interval::operator== ( const Domain & value ) const [virtual]

Implementing this is mandatory. Values are compared while computing the fixed point.

Implements Canal::Domain.

#### 16.17.1.3 void Canal::Float::Interval::setZero ( const llvm::Value \* place ) [virtual]

Set value of this domain to represent zeroed memory. Needed for constants with zero initializer. This can only be called right after creating a domain.

Implements Canal::Domain.

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

- lib/FloatInterval.h
- lib/FloatInterval.cpp

## 16.18 Canal::Interpreter::Iterator Class Reference

```
#include <InterpreterIterator.h>
```

### Public Member Functions

- **Iterator** (Module &module, Operations &operations, Widening::Manager &wideningManager)
- void **initialize** ()
- void **interpretInstruction** ()
- void **setCallback** (IteratorCallback &callback)
- bool **isInitialized** () const
- const State & **getCurrentState** () const
- const Function & **getCurrentFunction** () const
- const BasicBlock & **getCurrentBasicBlock** () const
- const llvm::Instruction & **getCurrentInstruction** () const
- std::string **toString** () const

### Protected Member Functions

- void **nextInstruction** ()

#### 16.18.1 Detailed Description

Basic iterator that iterates over the whole program until a fixpoint is reached.

#### 16.18.2 Member Function Documentation

##### 16.18.2.1 void Canal::Interpreter::Iterator::interpretInstruction ( )

One step of the interpreter. Interprets the instruction and moves to the next one.

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

- lib/InterpreterIterator.h
- lib/InterpreterIterator.cpp

## 16.19 Canal::Interpreter::IteratorCallback Class Reference

### Public Member Functions

- virtual void **onFixpointReached** ()
- virtual void **onModuleEnter** ()
- virtual void **onModuleExit** ()
- virtual void **onFunctionEnter** (Function &function)
- virtual void **onFunctionExit** (Function &function)
- virtual void **onBasicBlockEnter** (BasicBlock &basicBlock)
- virtual void **onBasicBlockExit** (BasicBlock &basicBlock)

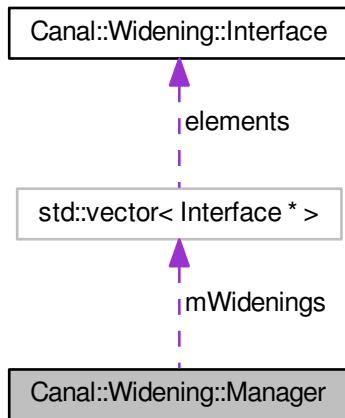
- virtual void **onInstructionEnter** (const llvm::Instruction &instruction)
- virtual void **onInstructionExit** (const llvm::Instruction &instruction)

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

- lib/InterpreterIteratorCallback.h

## 16.20 Canal::Widening::Manager Class Reference

Collaboration diagram for Canal::Widening::Manager:



### Public Member Functions

- void **widen** (const llvm::BasicBlock &wideningPoint, State &first, const State &second) const

### Protected Member Functions

- void **widen** (const llvm::BasicBlock &wideningPoint, StateMap &first, const StateMap &second) const
- void **widen** (const llvm::BasicBlock &wideningPoint, Domain &first, const Domain &second) const

### Protected Attributes

- std::vector< Interface \* > **mWidenings**

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

- lib/WideningManager.h
- lib/WideningManager.cpp

## 16.21 MemoryOverlay Class Reference

### Public Attributes

- uint64\_t **mOffset**
- Domain \* **mValue**

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

- lib/Memory.h

## 16.22 Canal::Interpreter::Module Class Reference

### Public Member Functions

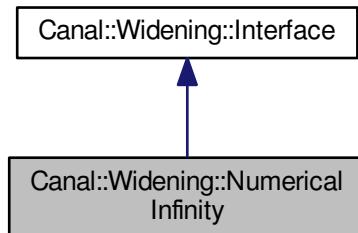
- **Module** (const llvm::Module &module, const Constructors &constructors)
- std::vector< Function \* >  
  ::const\_iterator **begin** () const
- std::vector< Function \* >  
  ::const\_iterator **end** () const
- Function \* **getFunction** (const char \*name) const
- Function \* **getFunction** (const std::string &name) const
- Function \* **getFunction** (const llvm::Function &function) const
- std::string **toString** () const
- void **updateGlobalState** ()

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

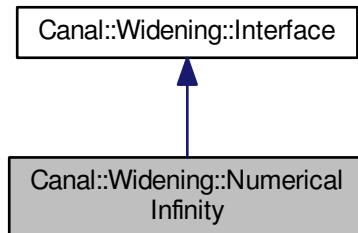
- lib/InterpreterModule.h
- lib/InterpreterModule.cpp

## 16.23 Canal::Widening::NumericalInfinity Class Reference

Inheritance diagram for Canal::Widening::NumericalInfinity:



Collaboration diagram for Canal::Widening::NumericalInfinity:



### Public Member Functions

- virtual void **widen** (const llvm::BasicBlock &wideningPoint, Domain &first, const Domain &second)

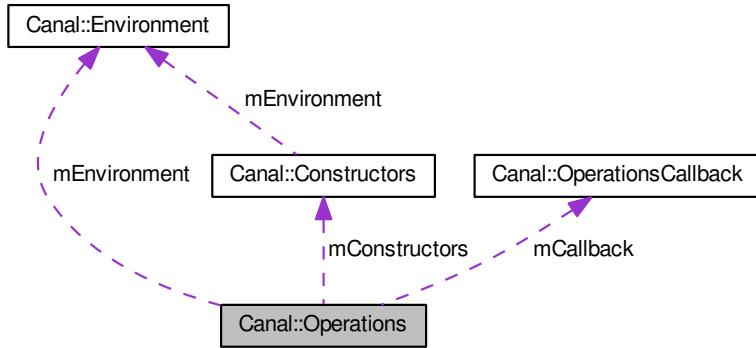
The documentation for this class was generated from the following files:

- lib/WideningNumericalInfinity.h
- lib/WideningNumericalInfinity.cpp

## 16.24 Canal::Operations Class Reference

```
#include <Operations.h>
```

Collaboration diagram for Canal::Operations:



## Public Member Functions

- **Operations** (const Environment &environment, const Constructors &constructors, OperationsCallback &callback)
- const Environment & **getEnvironment** () const
- void **interpretInstruction** (const llvm::Instruction &instruction, State &state)
 

*Interprets current instruction.*

## Protected Member Functions

- const Domain \* **variableOrConstant** (const llvm::Value &place, State &state, const llvm::Instruction &instruction, llvm::OwningPtr< Domain > &constant) const
- template<typename T>
 void **interpretCall** (const T &instruction, State &state)
- void **binaryOperation** (const llvm::BinaryOperator &instruction, State &state, Domain::BinaryOperation operation)
- template<typename T>
 bool **getElementPtrOffsets** (std::vector< Domain \* > &result, T iteratorStart, T iteratorEnd, const llvm::Value &place, const State &state)
- void **castOperation** (const llvm::CastInst &instruction, State &state, Domain::CastOperation operation)
- void **cmpOperation** (const llvm::CmpInst &instruction, State &state, Domain::CmpOperation operation)
- virtual void **ret** (const llvm::ReturnInst &instruction, State &state)
- virtual void **br** (const llvm::BranchInst &instruction, State &state)
- virtual void **switch\_** (const llvm::SwitchInst &instruction, State &state)
- virtual void **indirectbr** (const llvm::IndirectBrInst &instruction, State &state)
- virtual void **invoke** (const llvm::InvokeInst &instruction, State &state)
- virtual void **unreachable** (const llvm::UnreachableInst &instruction, State &state)
- virtual void **add** (const llvm::BinaryOperator &instruction, State &state)

*Sum of two operands. It's a binary operator.*

- virtual void fadd (const llvm::BinaryOperator &instruction, State &state)
- virtual void sub (const llvm::BinaryOperator &instruction, State &state)

*Difference of two operands. It's a binary operator.*

- virtual void fsub (const llvm::BinaryOperator &instruction, State &state)
- virtual void mul (const llvm::BinaryOperator &instruction, State &state)

*Product of two operands. It's a binary operator.*

- virtual void fmul (const llvm::BinaryOperator &instruction, State &state)

*Product of two operands. It's a binary operator.*

- virtual void udiv (const llvm::BinaryOperator &instruction, State &state)
- virtual void sdiv (const llvm::BinaryOperator &instruction, State &state)
- virtual void fdiv (const llvm::BinaryOperator &instruction, State &state)
- virtual void urem (const llvm::BinaryOperator &instruction, State &state)

*Unsigned division remainder. It's a binary operator.*

- virtual void srem (const llvm::BinaryOperator &instruction, State &state)

*Signed division remainder. It's a binary operator.*

- virtual void frem (const llvm::BinaryOperator &instruction, State &state)

*Floating point remainder. It's a binary operator.*

- virtual void shl (const llvm::BinaryOperator &instruction, State &state)

*It's a bitwise binary operator.*

- virtual void lshr (const llvm::BinaryOperator &instruction, State &state)

*It's a bitwise binary operator.*

- virtual void ashr (const llvm::BinaryOperator &instruction, State &state)

*It's a bitwise binary operator.*

- virtual void and\_ (const llvm::BinaryOperator &instruction, State &state)

*It's a bitwise binary operator.*

- virtual void or\_ (const llvm::BinaryOperator &instruction, State &state)

*It's a bitwise binary operator.*

- virtual void xor\_ (const llvm::BinaryOperator &instruction, State &state)

*It's a bitwise binary operator.*

- virtual void extractelement (const llvm::ExtractElementInst &instruction, State &state)

*It's a vector operation.*

- virtual void insertelement (const llvm::InsertElementInst &instruction, State &state)

*It's a vector operation.*

- virtual void shufflevector (const llvm::ShuffleVectorInst &instruction, State &state)

*It's a vector operation.*

- virtual void extractvalue (const llvm::ExtractValueInst &instruction, State &state)

*It's an aggregate operation.*

- virtual void insertvalue (const llvm::InsertValueInst &instruction, State &state)

*It's an aggregate operation.*

- virtual void alloca\_ (const llvm::AllocaInst &instruction, State &state)

*It's a memory access operation.*

- virtual void load (const llvm::LoadInst &instruction, State &state)

*It's a memory access operation.*

- virtual void store (const llvm::StoreInst &instruction, State &state)

*It's a memory access operation.*

- virtual void getelementptr (const llvm::GetElementPtrInst &instruction, State &state)

- virtual void trunc (const llvm::TruncInst &instruction, State &state)
 

*It's a memory addressing operation.*
- virtual void zext (const llvm::ZExtInst &instruction, State &state)
 

*It's a conversion operation.*
- virtual void sext (const llvm::SExtInst &instruction, State &state)
 

*It's a conversion operation.*
- virtual void fptrunc (const llvm::FPTruncInst &instruction, State &state)
 

*It's a conversion operation.*
- virtual void fpext (const llvm::FPExtInst &instruction, State &state)
 

*It's a conversion operation.*
- virtual void fptoui (const llvm::FPToUIInst &instruction, State &state)
 

*It's a conversion operation.*
- virtual void fptosi (const llvm::FPToSIInst &instruction, State &state)
 

*It's a conversion operation.*
- virtual void uitofp (const llvm::UIToFPIInst &instruction, State &state)
 

*It's a conversion operation.*
- virtual void sitofp (const llvm::SIToFPIInst &instruction, State &state)
 

*It's a conversion operation.*
- virtual void ptrtoint (const llvm::PtrToIntInst &instruction, State &state)
 

*It's a conversion operation.*
- virtual void inttoptr (const llvm::IntToPtrInst &instruction, State &state)
 

*It's a conversion operation.*
- virtual void bitcast (const llvm::BitCastInst &instruction, State &state)
 

*It's a conversion operation.*
- virtual void icmp (const llvm::ICmpInst &instruction, State &state)
- virtual void fcmp (const llvm::FCmpInst &instruction, State &state)
- virtual void phi (const llvm::PHINode &instruction, State &state)
- virtual void select (const llvm::SelectInst &instruction, State &state)
- virtual void call (const llvm::CallInst &instruction, State &state)
- virtual void va\_arg\_ (const llvm::VAArgInst &instruction, State &state)
- virtual void landingpad (const llvm::LandingPadInst &instruction, State &state)
- virtual void resume (const llvm::ResumeInst &instruction, State &state)
- virtual void fence (const llvm::FenceInst &instruction, State &state)
- virtual void cmpxchg (const llvm::AtomicCmpXchgInst &instruction, State &state)
- virtual void atomicrmw (const llvm::AtomicRMWInst &instruction, State &state)

## Protected Attributes

- const Environment & **mEnvironment**
- const Constructors & **mConstructors**
- OperationsCallback & **mCallback**

### 16.24.1 Detailed Description

Context-sensitive flow-insensitive operational abstract interpreter. Interprets instructions in abstract domain.

This is an abstract class, which is used as a base class for actual abstract interpretation implementations.

### 16.24.2 Member Function Documentation

**16.24.2.1 void Canal::Operations::br ( const llvm::BranchInst & *instruction*, State & *state* )**  
[protected], [virtual]

Transfer to a different basic block in the current function. It's a terminator instruction.

**16.24.2.2 void Canal::Operations::fadd ( const llvm::BinaryOperator & *instruction*, State & *state* )**  
[protected], [virtual]

Sum of two operands. It's a binary operator. The operands are floating point or vector of floating point values.

**16.24.2.3 void Canal::Operations::fdiv ( const llvm::BinaryOperator & *instruction*, State & *state* )**  
[protected], [virtual]

Quotient of two operands. It's a binary operator. The operands are floating point or vector of floating point values.

**16.24.2.4 void Canal::Operations::fsub ( const llvm::BinaryOperator & *instruction*, State & *state* )**  
[protected], [virtual]

Difference of two operands. It's a binary operator. The operands are floating point or vector of floating point values.

**16.24.2.5 void Canal::Operations::indirectbr ( const llvm::IndirectBrInst & *instruction*, State & *state* )**  
[protected], [virtual]

An indirect branch to a label within the current function, whose address is specified by "address". It's a terminator instruction.

**16.24.2.6 void Canal::Operations::invoke ( const llvm::InvokeInst & *instruction*, State & *state* )**  
[protected], [virtual]

Transfer to a specified function, with the possibility of control flow transfer to either the 'normal' label or the 'exception' label. It's a terminator instruction.

**16.24.2.7 void Canal::Operations::resume ( const llvm::ResumeInst & *instruction*, State & *state* )**  
[protected], [virtual]

Resume instruction is available since LLVM 3.0 A terminator instruction that has no successors. Resumes propagation of an existing (in-flight) exception whose unwinding was interrupted with a landingpad instruction.

**16.24.2.8 void Canal::Operations::ret ( const llvm::ReturnInst & *instruction*, State & *state* )**  
[protected], [virtual]

Return control flow (and optionally a value) from a function back to the caller. It's a terminator instruction.

**16.24.2.9 void Canal::Operations::sdiv ( const llvm::BinaryOperator & *instruction*, State & *state* )  
[protected], [virtual]**

Quotient of two operands. It's a binary operator. The operands are integer or vector of integer values.

**16.24.2.10 void Canal::Operations::switch\_ ( const llvm::SwitchInst & *instruction*, State & *state* )  
[protected], [virtual]**

Transfer control flow to one of several different places. It is a generalization of the 'br' instruction, allowing a branch to occur to one of many possible destinations. It's a terminator instruction.

**16.24.2.11 void Canal::Operations::udiv ( const llvm::BinaryOperator & *instruction*, State & *state* )  
[protected], [virtual]**

Quotient of two operands. It's a binary operator. The operands are integer or vector of integer values.

**16.24.2.12 void Canal::Operations::unreachable ( const llvm::UnreachableInst & *instruction*, State & *state* )  
[protected], [virtual]**

No defined semantics. This instruction is used to inform the optimizer that a particular portion of the code is not reachable. It's a terminator instruction.

**16.24.2.13 const Domain \* Canal::Operations::variableOrConstant ( const llvm::Value & *place*, State & *state*, const llvm::Instruction & *instruction*, llvm::OwningPtr< Domain > & *constant* ) const  
[protected]**

Given a place in source code, return the corresponding variable from the abstract interpreter state. If the place contains a constant, fill the provided constant variable with it.

### Returns

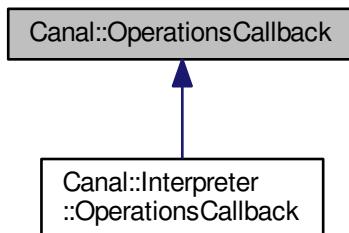
Returns a pointer to the variable if it is found in the state. Returns a pointer to the provided constant if the place contains a constant. Otherwise, it returns NULL.

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

- lib/Operations.h
- lib/Operations.cpp

## 16.25 Canal::OperationsCallback Class Reference

Inheritance diagram for Canal::OperationsCallback:



### Public Member Functions

- virtual void onFunctionCall (const llvm::Function &function, const State &callState, State &resultState, const llvm::Value &resultPlace)=0

#### 16.25.1 Member Function Documentation

**16.25.1.1 virtual void Canal::OperationsCallback::onFunctionCall ( const llvm::Function & *function*, const State & *callState*, State & *resultState*, const llvm::Value & *resultPlace* ) [pure virtual]**

##### Parameters

<i>callState</i>	Contains function arguments and referenced memory blocks.
<i>resultState</i>	A state where the changes caused by the function can be merged.
<i>resultPlace</i>	A place in the resultState where the returned value should be merged.

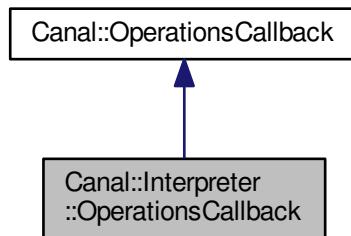
Implemented in Canal::Interpreter::OperationsCallback.

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

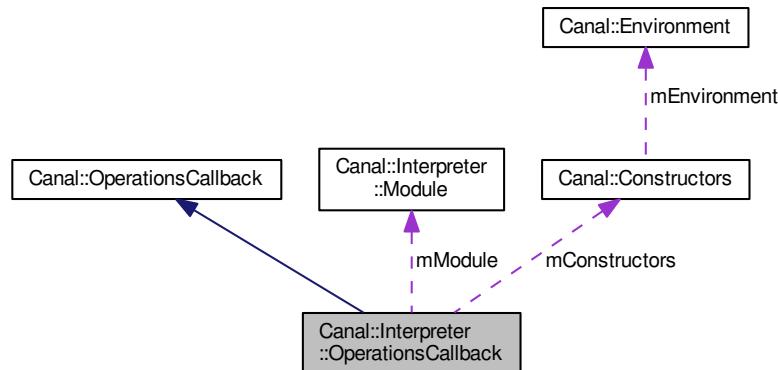
- lib/OperationsCallback.h

## 16.26 Canal::Interpreter::OperationsCallback Class Reference

Inheritance diagram for Canal::Interpreter::OperationsCallback:



Collaboration diagram for Canal::Interpreter::OperationsCallback:



### Public Member Functions

- **OperationsCallback** (Module &module, Constructors &mConstructors)
- virtual void onFunctionCall (const llvm::Function &function, const State &callState, State &resultState, const llvm::Value &resultPlace)

### Protected Attributes

- Module & **mModule**
- Constructors & **mConstructors**

### 16.26.1 Member Function Documentation

**16.26.1.1 void Canal::Interpreter::OperationsCallback::onFunctionCall ( const llvm::Function & *function*, const State & *callState*, State & *resultState*, const llvm::Value & *resultPlace* ) [virtual]**

#### Parameters

<i>callState</i>	Contains function arguments and referenced memory blocks.
<i>resultState</i>	A state where the changes caused by the function can be merged.
<i>resultPlace</i>	A place in the resultState where the returned value should be merged.

Implements Canal::OperationsCallback.

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

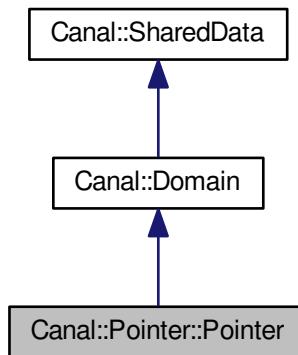
- lib/InterpreterOperationsCallback.h
- lib/InterpreterOperationsCallback.cpp

## 16.27 Canal::Pointer::Pointer Class Reference

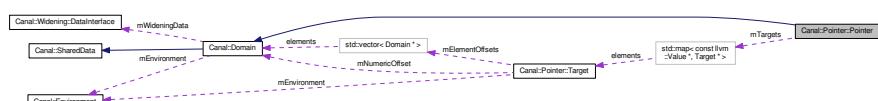
Inclusion-based flow-insensitive abstract pointer.

```
#include <Pointer.h>
```

Inheritance diagram for Canal::Pointer::Pointer:



Collaboration diagram for Canal::Pointer::Pointer:



## Public Member Functions

- Pointer (const Environment &environment, const llvm::PointerType &type)
 

*Standard constructor.*
- Pointer (const Pointer &value)
 

*Copy constructor.*
- Pointer (const Pointer &value, const llvm::PointerType &newType)
- virtual ~Pointer ()
 

*Standard destructor.*
- void addTarget (Target::Type type, const llvm::Value \*place, const llvm::Value \*target, const std::vector< Domain \* > &offsets, Domain \*numericOffset)
- Domain \* load (const State &state) const
- void **store** (const Domain &value, State &state) const
- Pointer \* getElementPtr (const std::vector< Domain \* > &offsets, const llvm::PointerType &type) const
- bool isConstant () const
 

*Does this pointer point to single target?*
- virtual Pointer \* clone () const
 

*Covariant return type.*
- virtual size\_t memoryUsage () const
 

*Get memory usage (used byte count) of this abstract value.*
- virtual std::string toString () const
 

*Create a string representation of the abstract value.*
- virtual void setZero (const llvm::Value \*place)
- virtual bool operator== (const Domain &value) const
- virtual bool **operator<** (const Domain &value) const
- virtual bool **operator>** (const Domain &value) const
- virtual Pointer & join (const Domain &value)
 

*Merge another value into this one.*
- virtual Pointer & **meet** (const Domain &value)
- virtual bool isBottom () const
 

*Is it the lowest value.*
- virtual void setBottom ()
 

*Set to the lowest value.*
- virtual bool **isValue** () const
- virtual const llvm::PointerType & getValueType () const
 

*Covariant return type.*
- virtual bool **hasValueExactSize** () const
- virtual Domain \* **getValueCell** (uint64\_t offset) const
- virtual void **mergeValueCell** (uint64\_t offset, const Domain &value)

## Public Attributes

- PlaceTargetMap mTargets
- const llvm::PointerType & mType

*The type object is owned by the LLVM framework.*

## Additional Inherited Members

### 16.27.1 Detailed Description

Inclusion-based flow-insensitive abstract pointer.

### 16.27.2 Constructor & Destructor Documentation

#### 16.27.2.1 `Canal::Pointer::Pointer ( const Pointer & value, const llvm::PointerType & newType )`

Copy constructor which changes the pointer type. Useful for bitcast and getelementptr operations.

### 16.27.3 Member Function Documentation

#### 16.27.3.1 `void Canal::Pointer::Pointer::addTarget ( Target::Type type, const llvm::Value * place, const llvm::Value * target, const std::vector<Domain * > & offsets, Domain * numericOffset )`

Add a new target to the pointer.

#### Parameters

<i>type</i>	Type of the referenced memory.
<i>place</i>	Place where the pointer target is added.
<i>target</i>	Represents the target memory block. If type is Constant, it must be NULL. Otherwise, it must be a valid pointer to an instruction. This is a key to State::mFunctionBlocks, State::mFunctionVariables, State::mGlobalBlocks, or State::mGlobalVariables, depending on the type.
<i>offsets</i>	Offsets in the getelementptr style. The provided vector might be empty. The newly created pointer target becomes the owner of the objects in the vector.
<i>numericOffset</i>	Numerical offset that is used in addition to the getelementptr style offset and after they have been applied. It might be NULL, which indicates the offset 0. The newly created pointer target becomes the owner of the numerical offset when it's provided. This parameter is mandatory for pointers of Constant type, because it contains the constant.

#### 16.27.3.2 `Pointer * Canal::Pointer::Pointer::getElementPtr ( const std::vector<Domain * > & offsets, const llvm::PointerType & type ) const`

Creates a copy of this pointer, but pointing to subtargets.

#### Parameters

<i>offsets</i>	Pointer takes ownership of the values inside the vector. The offsets must be converted to 64-bit integers before calling getElementPtr!
----------------	---

#### 16.27.3.3 `Domain * Canal::Pointer::Pointer::load ( const State & state ) const`

Dereference all targets and merge the results into single abstract value. The returned value is owned by the caller.

**Returns**

It might return NULL.

**16.27.3.4 bool Canal::Pointer::operator== ( const Domain & value ) const [virtual]**

Implementing this is mandatory. Values are compared while computing the fixed point.

Implements Canal::Domain.

**16.27.3.5 void Canal::Pointer::setZero ( const llvm::Value \* place ) [virtual]**

Set value of this domain to represent zeroed memory. Needed for constants with zero initializer. This can only be called right after creating a domain.

Implements Canal::Domain.

**16.27.4 Member Data Documentation****16.27.4.1 PlaceTargetMap Canal::Pointer::mTargets**

llvm::Value represents a position in the program. It points to the instruction where the target was assigned/stored to the pointer.

**16.27.4.2 const llvm::PointerType& Canal::Pointer::mType**

The type object is owned by the LLVM framework.

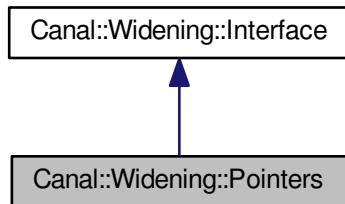
Type of the object the pointer is pointing to. It might be incompatible with the type of the actual abstract value. Conversion is needed during store and load operations in such a case.

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

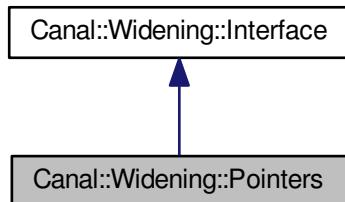
- lib/Pointer.h
- lib/Memory.cpp
- lib/Pointer.cpp

## 16.28 Canal::Widening::Pointers Class Reference

Inheritance diagram for Canal::Widening::Pointers:



Collaboration diagram for Canal::Widening::Pointers:



### Public Member Functions

- virtual void **widen** (const llvm::BasicBlock &wideningPoint, Domain &first, const Domain &second)

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

- lib/WideningPointers.h
- lib/WideningPointers.cpp

## 16.29 Canal::APIntUtils::SCompare Struct Reference

### Public Member Functions

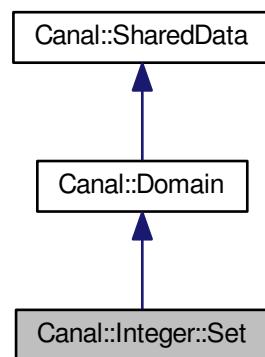
- bool **operator()** (const llvm::APInt &a, const llvm::APInt &b) const

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

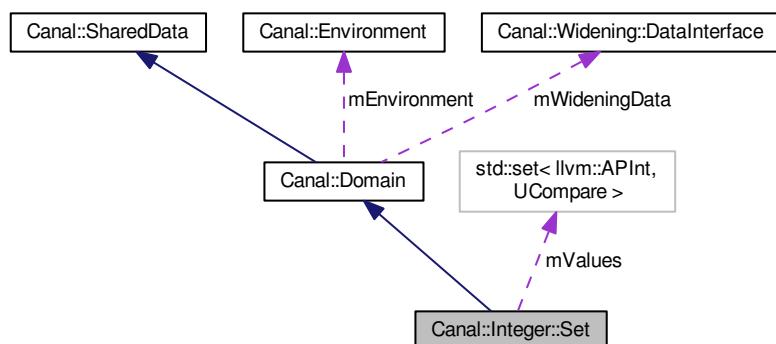
- lib/APIntUtils.h

## 16.30 Canal::Integer::Set Class Reference

Inheritance diagram for Canal::Integer::Set:



Collaboration diagram for Canal::Integer::Set:



### Public Member Functions

- `Set (const Environment &environment, unsigned bitWidth)`

- `Set (const Environment &environment, const llvm::APInt &constant)`

*Initializes to the lowest value.*
- `Set (const Set &value)`

*Initializes to the given value.*
- `Set (const Set &value)`

*Copy constructor.*
- `unsigned getBitWidth () const`
- `bool signedMin (llvm::APInt &result) const`
- `bool signedMax (llvm::APInt &result) const`
- `bool unsignedMin (llvm::APInt &result) const`
- `bool unsignedMax (llvm::APInt &result) const`
- `bool isConstant () const`
- `bool isTrue () const`

*Does this set represent single value?*
- `bool isFalse () const`

*Does the set represent signle bit that is set to 1?*
- `virtual Set * clone () const`

*Covariant return type.*
- `virtual size_t memoryUsage () const`

*Get memory usage (used byte count) of this abstract value.*
- `virtual std::string toString () const`

*Create a string representation of the abstract value.*
- `virtual void setZero (const llvm::Value *place)`
- `virtual bool operator== (const Domain &value) const`
- `virtual bool operator< (const Domain &value) const`
- `virtual bool operator> (const Domain &value) const`
- `virtual Set & join (const Domain &value)`
- `virtual void setBottom ()`

*Merge another value into this one.*
- `virtual Set & meet (const Domain &value)`
- `virtual bool isBottom () const`

*Is it the lowest value.*
- `virtual void setTop ()`

*Set to the lowest value.*
- `virtual bool isTop () const`

*Is it the highest value.*
- `virtual void setTop ()`

*Set it to the top value of lattice.*
- `virtual float accuracy () const`
- `virtual Set & add (const Domain &a, const Domain &b)`
- `virtual Set & sub (const Domain &a, const Domain &b)`
- `virtual Set & mul (const Domain &a, const Domain &b)`
- `virtual Set & udiv (const Domain &a, const Domain &b)`
- `virtual Set & sdiv (const Domain &a, const Domain &b)`

*Unsigned division.*
- `virtual Set & urem (const Domain &a, const Domain &b)`

*Signed division.*
- `virtual Set & srem (const Domain &a, const Domain &b)`
- `virtual Set & shl (const Domain &a, const Domain &b)`

- virtual Set & **lshr** (const Domain &a, const Domain &b)
- virtual Set & **ashr** (const Domain &a, const Domain &b)
- virtual Set & **and\_** (const Domain &a, const Domain &b)
- virtual Set & **or\_** (const Domain &a, const Domain &b)
- virtual Set & **xor\_** (const Domain &a, const Domain &b)
- virtual Set & **icmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual Set & **fcmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual Set & **trunc** (const Domain &value)
- virtual Set & **zext** (const Domain &value)
- virtual Set & **sext** (const Domain &value)
- virtual Set & **fptoui** (const Domain &value)
- virtual Set & **fptosi** (const Domain &value)
- virtual bool **isValue** () const
- virtual const llvm::IntegerType & **getValueType** () const

*Covariant return type.*

- virtual bool **hasValueExactSize** () const
- virtual Domain \* **getValueCell** (uint64\_t offset) const
- virtual void **mergeValueCell** (uint64\_t offset, const Domain &value)

## Public Attributes

- APIntUtils::USet **mValues**
- bool **mTop**
- unsigned **mBitWidth**

## Static Public Attributes

- static unsigned int **SET\_THRESHOLD** = 40

## Protected Member Functions

- Set & **applyOperation** (const Domain &a, const Domain &b, APIntUtils::Operation operation1, APIntUtils::OperationWithOverflow operation2)

## Additional Inherited Members

### 16.30.1 Member Function Documentation

#### 16.30.1.1 float Canal::Integer::Set::accuracy ( ) const [virtual]

Get accuracy of the abstract value (0 - 1). In finite-height lattices, it is determined by the position of the value in the lattice.

Accuracy 0 means that the value represents all possible values (top). Accuracy 1 means that the value represents the most precise and exact value (bottom).

Reimplemented from Canal::Domain.

**16.30.1.2 bool Canal::Integer::Set::operator== ( const Domain & value ) const [virtual]**

Implementing this is mandatory. Values are compared while computing the fixed point.

Implements Canal::Domain.

**16.30.1.3 void Canal::Integer::Set::setZero ( const llvm::Value \* place ) [virtual]**

Set value of this domain to represent zeroed memory. Needed for constants with zero initializer. This can only be called right after creating a domain.

Implements Canal::Domain.

**16.30.1.4 bool Canal::Integer::Set::signedMax ( llvm::APInt & result ) const**

Highest signed number represented by this abstract domain.

**Parameters**

<i>result</i>	Filled by the maximum value if it is known. Otherwise, the value is undefined.
---------------	--

**Returns**

True if the result is known and the parameter was set to correct value.

**16.30.1.5 bool Canal::Integer::Set::signedMin ( llvm::APInt & result ) const**

Lowest signed number represented by this abstract domain.

**Parameters**

<i>result</i>	Filled by the minimum value if it is known. Otherwise, the value is undefined.
---------------	--

**Returns**

True if the result is known and the parameter was set to correct value.

**16.30.1.6 bool Canal::Integer::Set::unsignedMax ( llvm::APInt & result ) const**

Highest unsigned number represented by this abstract domain.

**Parameters**

<i>result</i>	Filled by the maximum value if it is known. Otherwise, the value is undefined.
---------------	--

**Returns**

True if the result is known and the parameter was set to correct value.

### 16.30.1.7 bool Canal::Integer::Set::unsignedMin ( llvm::APInt & result ) const

Lowest unsigned number represented by this abstract domain.

#### Parameters

<i>result</i>	Filled by the minimum value if it is known. Otherwise, the value is undefined.
---------------	--

#### Returns

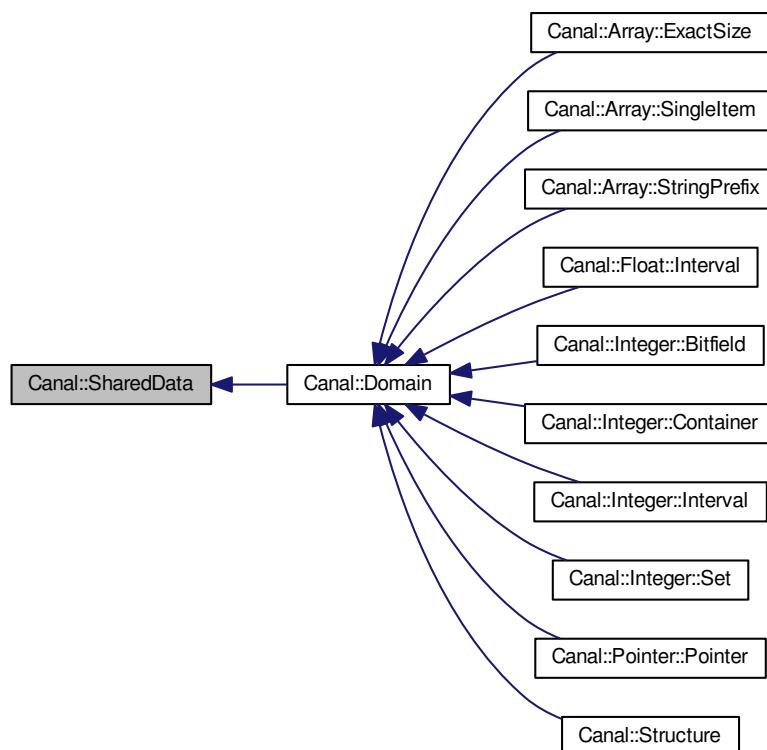
True if the result is known and the parameter was set to correct value.

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

- lib/IntegerSet.h
- lib/IntegerSet.cpp

## 16.31 Canal::SharedData Class Reference

Inheritance diagram for Canal::SharedData:



## Public Member Functions

- SharedData (const SharedData &)

## Public Attributes

- int **mRefCount**

### 16.31.1 Constructor & Destructor Documentation

#### 16.31.1.1 Canal::SharedData ( const SharedData & ) [inline]

When copying an object to a new one, reset the reference count.

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

- lib/SharedDataPointer.h

## 16.32 Canal::SharedDataPointer< T > Class Template Reference

## Public Member Functions

- **SharedDataPointer** (T \*ptr)
- **SharedDataPointer** (const SharedDataPointer< T > &ptr)
- SharedDataPointer< T > & **operator=** (SharedDataPointer< T > &ptr)
- SharedDataPointer< T > & **operator=** (T \*ptr)
- bool **operator==** (const SharedDataPointer< T > &ptr) const
- bool **operator!=** (const SharedDataPointer< T > &ptr) const
- const T & **operator\*** () const
- const T \* **operator->** () const
- bool **operator!** () const
- T \* **mutable\_** ()
- const T \* **data** () const

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

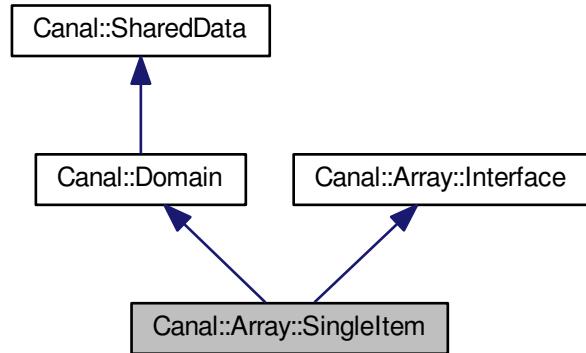
- lib/SharedDataPointer.h

## 16.33 Canal::Array::SingleItem Class Reference

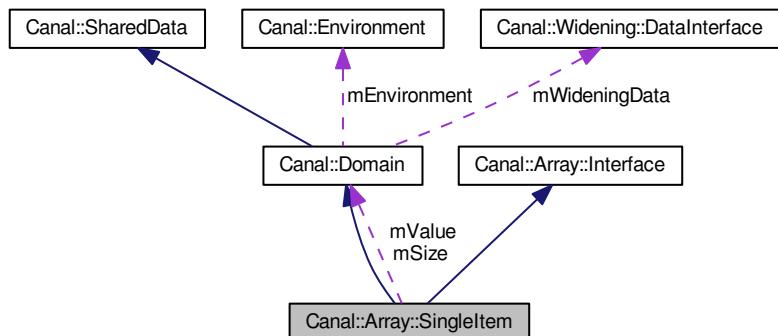
This array type is very imprecise.

```
#include <ArraySingleItem.h>
```

Inheritance diagram for Canal::Array::SingleItem:



Collaboration diagram for Canal::Array::SingleItem:



## Public Member Functions

- `SingleItem (const Environment &environment, const llvm::SequentialType &type)`  
*Standard constructor.*
- `SingleItem (const Environment &environment, const llvm::SequentialType &type, const std::vector< Domain * > &values)`
- `SingleItem (const Environment &environment, const llvm::SequentialType &type, Domain *size)`
- `SingleItem (const SingleItem &value)`
- `virtual SingleItem * clone () const`

*Covariant return type.*

- virtual size\_t memoryUsage () const
 

*Get memory usage (used byte count) of this abstract value.*
- virtual std::string toString () const
 

*Create a string representation of the abstract value.*
- virtual void setZero (const llvm::Value \*place)
- virtual bool operator== (const Domain &value) const
- virtual bool **operator<** (const Domain &value) const
- virtual bool **operator>** (const Domain &value) const
- virtual SingleItem & join (const Domain &value)
 

*Computes a join of both values and array sizes.*
- virtual SingleItem & meet (const Domain &value)
 

*Computes a meet of both values and array sizes.*
- virtual bool isBottom () const
 

*Check if all items in the array are bottom.*
- virtual void setBottom ()
 

*Set all items in the array to bottom.*
- virtual bool isTop () const
 

*Check if all items in the array are top.*
- virtual void setTop ()
 

*Set all items in the array to top.*
- virtual float accuracy () const
- virtual SingleItem & **add** (const Domain &a, const Domain &b)
- virtual SingleItem & **fadd** (const Domain &a, const Domain &b)
- virtual SingleItem & **sub** (const Domain &a, const Domain &b)
- virtual SingleItem & **fsub** (const Domain &a, const Domain &b)
- virtual SingleItem & **mul** (const Domain &a, const Domain &b)
- virtual SingleItem & **fmul** (const Domain &a, const Domain &b)
- virtual SingleItem & **udiv** (const Domain &a, const Domain &b)
 

*Unsigned division.*
- virtual SingleItem & **sdiv** (const Domain &a, const Domain &b)
 

*Signed division.*
- virtual SingleItem & **fdiv** (const Domain &a, const Domain &b)
 

*Floating point division.*
- virtual SingleItem & **urem** (const Domain &a, const Domain &b)
- virtual SingleItem & **srem** (const Domain &a, const Domain &b)
- virtual SingleItem & **frem** (const Domain &a, const Domain &b)
- virtual SingleItem & **shl** (const Domain &a, const Domain &b)
- virtual SingleItem & **lshr** (const Domain &a, const Domain &b)
- virtual SingleItem & **ashr** (const Domain &a, const Domain &b)
- virtual SingleItem & **and\_** (const Domain &a, const Domain &b)
- virtual SingleItem & **or\_** (const Domain &a, const Domain &b)
- virtual SingleItem & **xor\_** (const Domain &a, const Domain &b)
- virtual SingleItem & **icmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual SingleItem & **fcmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual bool **isValue** () const
- virtual const
 

llvm::SequentialType & **getValueType** () const

*Covariant return type.*

- virtual bool **hasValueExactSize () const**
- virtual Domain \* **getValueAbstractSize () const**
- virtual Domain \* **getValueCell (uint64\_t offset) const**
- virtual void **mergeValueCell (uint64\_t offset, const Domain &value)**
- virtual std::vector< Domain \* > **getItem (const Domain &offset) const**
- virtual Domain \* **getItem (uint64\_t offset) const**
- virtual void **setItem (const Domain &offset, const Domain &value)**
- virtual void **setItem (uint64\_t offset, const Domain &value)**

## Public Attributes

- Domain \* **mValue**
- Domain \* **mSize**
- const llvm::SequentialType & **mType**

## Additional Inherited Members

### 16.33.1 Detailed Description

This array type is very imprecise.

The most trivial array type. It treats all array members as a single value. This means all the operations on the array are merged and used to move the single value up in its lattice.

### 16.33.2 Constructor & Destructor Documentation

#### 16.33.2.1 **Canal::Array::SingleItem::SingleItem ( const Environment & environment, const llvm::SequentialType & type, const std::vector< Domain \* > & values )**

##### Parameters

<i>values</i>	This class takes ownership of the values.
---------------	---

### 16.33.3 Member Function Documentation

#### 16.33.3.1 **float Canal::Array::SingleItem::accuracy ( ) const [virtual]**

Get accuracy of the abstract value (0 - 1). In finite-height lattices, it is determined by the position of the value in the lattice.

Accuracy 0 means that the value represents all possible values (top). Accuracy 1 means that the value represents the most precise and exact value (bottom).

Reimplemented from Canal::Domain.

#### 16.33.3.2 **std::vector< Domain \* > Canal::Array::SingleItem::getItem ( const Domain & offset ) const [virtual]**

Get the array items pointed by the provided offset. Returns internal array items that are owned by the array. Caller must not delete the items.

Implements Canal::Array::Interface.

#### 16.33.3.3 Domain \* Canal::Array::SingleItem::getItem ( uint64\_t offset ) const [virtual]

Get the array item pointed by the provided offset. Returns internal array item that is owned by the array. Caller must not delete the item.

##### Note

The uint64\_t offset variant exists because of the extractvalue instruction, which provides exact numeric offsets.

For future array domains it might be necessary to extend this method to return a list of values.

Implements Canal::Array::Interface.

#### 16.33.3.4 bool Canal::Array::SingleItem::operator== ( const Domain & value ) const [virtual]

Implementing this is mandatory. Values are compared while computing the fixed point.

Implements Canal::Domain.

#### 16.33.3.5 void Canal::Array::SingleItem::setItem ( const Domain & offset, const Domain & value ) [virtual]

##### Parameters

<i>value</i>	The method does not take the ownership of this memory. It copies the contents of the value instead.
--------------	---

Implements Canal::Array::Interface.

#### 16.33.3.6 void Canal::Array::SingleItem::setItem ( uint64\_t offset, const Domain & value ) [virtual]

##### Parameters

<i>value</i>	The method does not take the ownership of this memory. It copies the contents of the value instead.
--------------	---

##### Note

The uint64\_t offset variant exists because of the insertvalue instruction, which provides exact numeric offsets.

Implements Canal::Array::Interface.

#### 16.33.3.7 void Canal::Array::SingleItem::setZero ( const llvm::Value \* place ) [virtual]

Set value of this domain to represent zeroed memory. Needed for constants with zero initializer. This can only be called right after creating a domain.

Implements Canal::Domain.

### 16.33.4 Member Data Documentation

#### 16.33.4.1 Domain\* Canal::Array::SingleItem::mSize

Number of elements in the array. It is either a Constant or Integer::Container.

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

- lib/ArraySingleItem.h
- lib/ArraySingleItem.cpp

## 16.34 Canal::SlotTracker Class Reference

```
#include <SlotTracker.h>
```

### Public Types

- `typedef std::map< const llvm::MDNode *, unsigned >`  
`::iterator mdn_iterator`

*MDNode map iterators.*

### Public Member Functions

- `SlotTracker (const llvm::Module &module)`  
*Construct from a module.*
- `void setActiveFunction (const llvm::Function &function)`
- `int getLocalSlot (const llvm::Value &value)`
- `const llvm::Value * getLocalSlot (unsigned num)`
- `int getGlobalSlot (const llvm::Value &value)`  
*Get the slot number of a global value.*
- `const llvm::Value * getGlobalSlot (unsigned num)`
- `int getMetadataSlot (const llvm::MDNode &node)`  
*Get the slot number of a MDNode.*
- `mdn_iterator mdn_begin ()`
- `mdn_iterator mdn_end ()`
- `unsigned mdn_size () const`
- `bool mdn_empty () const`

### Protected Member Functions

- `void initialize ()`  
*This function does the actual initialization.*
- `void createFunctionSlot (const llvm::Value &value)`  
*Insert the specified Value\* into the slot table.*
- `void createModuleSlot (const llvm::GlobalValue &value)`  
*Insert the specified GlobalValue\* into the slot table.*

- void createMetadataSlot (const llvm::MDNode &node)  
*Insert the specified MDNode\* into the slot table.*
- void processModule ()
- void processFunction ()

### 16.34.1 Detailed Description

This class provides computation of slot numbers. Initial version was taken from LLVM source code (lib/VMCore/AsmWriter.cpp).

### 16.34.2 Member Function Documentation

#### 16.34.2.1 int Canal::SlotTracker::getLocalSlot ( const llvm::Value & value )

Get the slot number for a value that is local to a function. Return the slot number of the specified value in it's type plane. If something is not in the SlotTracker, return -1.

#### 16.34.2.2 void Canal::SlotTracker::processFunction ( ) [protected]

Add all of the functions arguments, basic blocks, and instructions.

#### 16.34.2.3 void Canal::SlotTracker::processModule ( ) [protected]

Add all of the module level global variables (and their initializers) and function declarations, but not the contents of those functions.

#### 16.34.2.4 void Canal::SlotTracker:: setActiveFunction ( const llvm::Function & function )

If you'd like to deal with a function instead of just a module, use this method to get its data into the SlotTracker.

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

- lib/SlotTracker.h
- lib/SlotTracker.cpp

## 16.35 Canal::State Class Reference

Abstract memory state.

```
#include <State.h>
```

### Public Member Functions

- **State** (const State &state)
- bool **operator==** (const State &state) const
- bool **operator!=** (const State &state) const
- void merge (const State &state)

- void mergeGlobal (const State &state)
 

*Merge everything.*
- void mergeFunctionBlocks (const State &state)
 

*Merge function blocks only.*
- void mergeForeignFunctionBlocks (const State &state, const llvm::Function &currentFunction)
- void addGlobalVariable (const llvm::Value &place, Domain \*value)
- void addFunctionVariable (const llvm::Value &place, Domain \*value)
- void **addGlobalBlock** (const llvm::Value &place, Domain \*value)
- void addFunctionBlock (const llvm::Value &place, Domain \*value)
 

*Adds a value created by alloca to the stack.*
- void **setReturnedValue** (Domain \*value)
- void **mergeToReturnedValue** (const Domain &value)
- const Domain \* **getReturnedValue** () const
- void **addVariableArgument** (const llvm::Instruction &place, Domain \*argument)
- const StateMap & **getGlobalVariables** () const
- StateMap & **getGlobalVariables** ()
- const StateMap & **getGlobalBlocks** () const
- StateMap & **getGlobalBlocks** ()
- const StateMap & **getFunctionVariables** () const
- StateMap & **getFunctionVariables** ()
- const StateMap & **getFunctionBlocks** () const
- StateMap & **getFunctionBlocks** ()
- const Domain \* **findVariable** (const llvm::Value &place) const
- const Domain \* **findBlock** (const llvm::Value &place) const
- bool **hasGlobalBlock** (const llvm::Value &place) const
- size\_t **memoryUsage** () const
 

*Get memory usage (used byte count) of this abstract state.*
- std::string **toString** (const llvm::Value &place, SlotTracker &slotTracker) const

### 16.35.1 Detailed Description

Abstract memory state.

Consists of function-level variables (also called registers) and stack memory, global variables and heap, and return value. All variables are in abstract domain.

### 16.35.2 Member Function Documentation

#### 16.35.2.1 void Canal::State::addFunctionVariable ( const llvm::Value & place, Domain \* value )

Adds a register-type value to the stack.

##### Parameters

<i>place</i>	Represents a place in the program where the function variable is assigned. Usually it is an instance of llvm::Instruction for a result of the instruction. It might also be an instance of llvm::Argument, which represents a function call parameter.
--------------	--

**See also**

To add a value created by alloca to the stack, use the method `addFunctionBlock`.

**16.35.2.2 void Canal::State::addGlobalVariable ( const llvm::Value & place, Domain \* value )****Parameters**

<i>place</i>	Represents a place in the program where the global variable is defined and assigned.
--------------	--

**16.35.2.3 const Domain \* Canal::State::findBlock ( const llvm::Value & place ) const**

Search both global and function blocks for a place. If the place is found, the block is returned. Otherwise NULL is returned.

**16.35.2.4 const Domain \* Canal::State::findVariable ( const llvm::Value & place ) const**

Search both global and function variables for a place. If the place is found, the variable is returned. Otherwise NULL is returned.

**16.35.2.5 void Canal::State::mergeForeignFunctionBlocks ( const State & state, const llvm::Function & currentFunction )**

Merge function memory blocks external to a function. This is used after a function call, where the modifications of the global state need to be merged to the state of the caller, but its local state is not relevant.

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

- lib/State.h
- lib/State.cpp

## 16.36 Canal::StateMap Class Reference

### Public Types

- `typedef Map::iterator iterator`
- `typedef Map::const_iterator const_iterator`
- `typedef Map::value_type value_type`
- `typedef Map::size_type size_type`
- `typedef Map::key_type key_type`

### Public Member Functions

- `StateMap (const StateMap &map)`
- `bool operator== (const StateMap &map) const`
- `iterator begin ()`
- `const_iterator begin () const`
- `iterator end ()`

- `const_iterator end () const`
- `size_type size () const`
- `void clear ()`
- `iterator find (const key_type &x)`
- `const_iterator find (const key_type &x) const`
- `std::pair< iterator, bool > insert (const value_type &x)`
- `void merge (const StateMap &map)`
- `void insert (const llvm::Value &place, Domain *value)`
- `size_t memoryUsage () const`

*Get memory usage (used byte count) of this state map.*

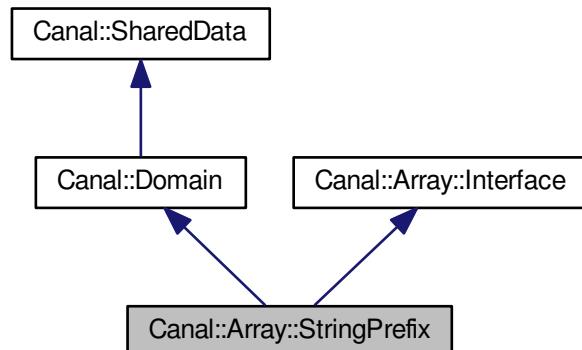
The documentation for this class was generated from the following files:

- `lib/StateMap.h`
- `lib/StateMap.cpp`

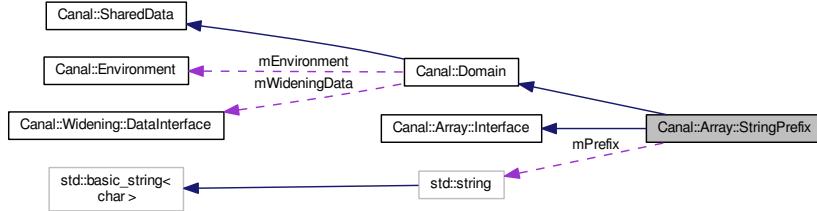
## 16.37 Canal::Array::StringPrefix Class Reference

```
#include <ArrayStringPrefix.h>
```

Inheritance diagram for Canal::Array::StringPrefix:



Collaboration diagram for Canal::Array::StringPrefix:



## Public Member Functions

- **StringPrefix** (const Environment &environment, const llvm::SequentialType &type)
 

*Standard constructor.*
- **StringPrefix** (const Environment &environment, const llvm::SequentialType &type, const std::vector<Domain \*> &values)
- **StringPrefix** (const Environment &environment, const llvm::SequentialType &type, Domain \*size)
- **StringPrefix** (const Environment &environment, const std::string &value)
- virtual StringPrefix \* clone () const
 

*Covariant return type.*
- virtual size\_t memoryUsage () const
 

*Get memory usage (used byte count) of this abstract value.*
- virtual std::string toString () const
 

*Create a string representation of the abstract value.*
- virtual void setZero (const llvm::Value \*place)
- virtual bool operator== (const Domain &value) const
- virtual bool operator< (const Domain &value) const
- virtual bool operator> (const Domain &value) const
- virtual StringPrefix & join (const Domain &value)
 

*Merge another value into this one.*
- virtual StringPrefix & **meet** (const Domain &value)
- virtual bool isBottom () const
 

*Is it the lowest value.*
- virtual void setBottom ()
 

*Set to the lowest value.*
- virtual bool isTop () const
 

*Is it the highest value.*
- virtual void setTop ()
 

*Set it to the top value of lattice.*
- virtual float accuracy () const
- virtual StringPrefix & **add** (const Domain &a, const Domain &b)
- virtual StringPrefix & **fadd** (const Domain &a, const Domain &b)
- virtual StringPrefix & **sub** (const Domain &a, const Domain &b)
- virtual StringPrefix & **fsub** (const Domain &a, const Domain &b)
- virtual StringPrefix & **mul** (const Domain &a, const Domain &b)

- virtual StringPrefix & **fmul** (const Domain &a, const Domain &b)
- virtual StringPrefix & udiv (const Domain &a, const Domain &b)
 

*Unsigned division.*
- virtual StringPrefix & sdiv (const Domain &a, const Domain &b)
 

*Signed division.*
- virtual StringPrefix & fdiv (const Domain &a, const Domain &b)
 

*Floating point division.*
- virtual StringPrefix & **urem** (const Domain &a, const Domain &b)
- virtual StringPrefix & **srem** (const Domain &a, const Domain &b)
- virtual StringPrefix & **frem** (const Domain &a, const Domain &b)
- virtual StringPrefix & **shl** (const Domain &a, const Domain &b)
- virtual StringPrefix & **lshr** (const Domain &a, const Domain &b)
- virtual StringPrefix & **ashr** (const Domain &a, const Domain &b)
- virtual StringPrefix & **and\_** (const Domain &a, const Domain &b)
- virtual StringPrefix & **or\_** (const Domain &a, const Domain &b)
- virtual StringPrefix & **xor\_** (const Domain &a, const Domain &b)
- virtual StringPrefix & **icmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual StringPrefix & **fcmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual bool **isValue** () const
- virtual const  
llvm::SequentialType & **getValueType** () const
 

*Covariant return type.*
- virtual bool **hasValueExactSize** () const
- virtual Domain \* **getValueAbstractSize** () const
- virtual Domain \* **getValueCell** (uint64\_t offset) const
- virtual void **mergeValueCell** (uint64\_t offset, const Domain &value)
- virtual std::vector< Domain \* > **getItem** (const Domain &offset) const
- virtual Domain \* **getItem** (uint64\_t offset) const
- virtual void **setItem** (const Domain &offset, const Domain &value)
- virtual void **setItem** (uint64\_t offset, const Domain &value)

## Public Attributes

- std::string **mPrefix**
- bool **mIsBottom**
- const llvm::SequentialType & **mType**

## Additional Inherited Members

### 16.37.1 Detailed Description

Array with exact size and limited length. It keeps all array members separately, not losing precision at all.

### 16.37.2 Constructor & Destructor Documentation

16.37.2.1 `Canal::Array::StringPrefix ( const Environment & environment, const Ilm::SequentialType & type, const std::vector< Domain * > & values )`

#### Parameters

<code>values</code>	This class takes ownership of the values.
---------------------	---

### 16.37.3 Member Function Documentation

16.37.3.1 `float Canal::Array::StringPrefix::accuracy ( ) const [virtual]`

Get accuracy of the abstract value (0 - 1). In finite-height lattices, it is determined by the position of the value in the lattice.

Accuracy 0 means that the value represents all possible values (top). Accuracy 1 means that the value represents the most precise and exact value (bottom).

Reimplemented from `Canal::Domain`.

16.37.3.2 `std::vector< Domain * > Canal::Array::StringPrefix::getItem ( const Domain & offset ) const [virtual]`

Get the array items pointed by the provided offset. Returns internal array items that are owned by the array. Caller must not delete the items.

Implements `Canal::Array::Interface`.

16.37.3.3 `Domain * Canal::Array::StringPrefix::getItem ( uint64_t offset ) const [virtual]`

Get the array item pointed by the provided offset. Returns internal array item that is owned by the array. Caller must not delete the item.

#### Note

The `uint64_t` offset variant exists because of the `extractvalue` instruction, which provides exact numeric offsets.

For future array domains it might be necessary to extend this method to return a list of values.

Implements `Canal::Array::Interface`.

16.37.3.4 `bool Canal::Array::StringPrefix::operator== ( const Domain & value ) const [virtual]`

Implementing this is mandatory. Values are compared while computing the fixed point.

Implements `Canal::Domain`.

16.37.3.5 `void Canal::Array::StringPrefix::setItem ( const Domain & offset, const Domain & value ) [virtual]`

**Parameters**

<i>value</i>	The method does not take the ownership of this memory. It copies the contents of the value instead.
--------------	---

Implements Canal::Array::Interface.

#### 16.37.3.6 void Canal::Array::StringPrefix::setItem ( uint64\_t offset, const Domain & value ) [virtual]

**Parameters**

<i>value</i>	The method does not take the ownership of this memory. It copies the contents of the value instead.
--------------	---

**Note**

The uint64\_t offset variant exists because of the insertvalue instruction, which provides exact numeric offsets.

Implements Canal::Array::Interface.

#### 16.37.3.7 void Canal::Array::StringPrefix::setZero ( const llvm::Value \* place ) [virtual]

Set value of this domain to represent zeroed memory. Needed for constants with zero initializer. This can only be called right after creating a domain.

Implements Canal::Domain.

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

- lib/ArrayStringPrefix.h
- lib/ArrayStringPrefix.cpp

## 16.38 Canal::StringStream Class Reference

```
#include <Utils.h>
```

### 16.38.1 Detailed Description

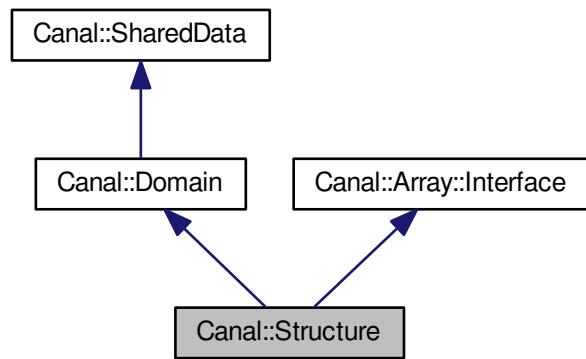
A raw\_string\_ostream that writes to an embedded std::string. This is a simple adaptor class.

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

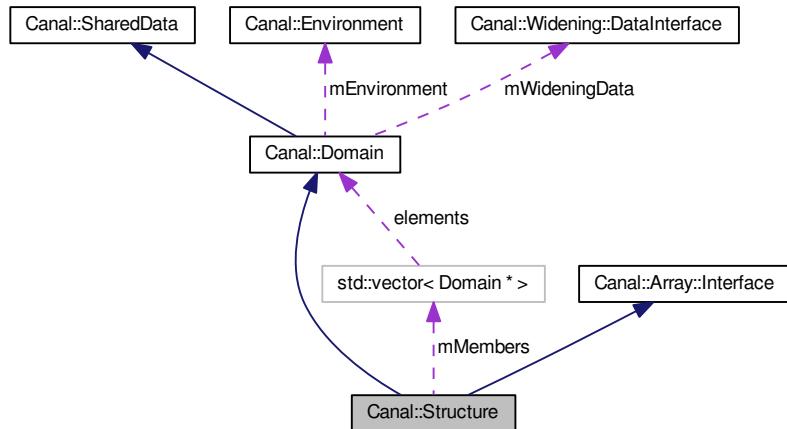
- lib/Utils.h

## 16.39 Canal::Structure Class Reference

Inheritance diagram for Canal::Structure:



Collaboration diagram for Canal::Structure:



### Public Member Functions

- **Structure** (const Environment &environment, const llvm::StructType &type, const std::vector< Domain \* > &members)
- **Structure** (const Structure &value)
- virtual Structure \* clone () const

- Covariant return type.*
- virtual size\_t memoryUsage () const  
*Get memory usage (used byte count) of this abstract value.*
  - virtual std::string toString () const  
*Create a string representation of the abstract value.*
  - virtual void setZero (const llvm::Value \*place)
  - virtual bool operator== (const Domain &value) const
  - virtual bool operator< (const Domain &value) const
  - virtual bool operator> (const Domain &value) const
  - virtual Structure & join (const Domain &value)
- Merge another value into this one.*
- virtual Structure & meet (const Domain &value)
  - virtual bool isBottom () const  
*Is it the lowest value.*
  - virtual void setBottom ()  
*Set to the lowest value.*
  - virtual bool isTop () const  
*Is it the highest value.*
  - virtual void setTop ()  
*Set it to the top value of lattice.*
  - virtual float accuracy () const
  - virtual bool isValue () const
  - virtual const llvm::StructType & getValueType () const
- Covariant return type.*
- virtual bool hasValueExactSize () const
  - virtual Domain \* getValueCell (uint64\_t offset) const
  - virtual void mergeValueCell (uint64\_t offset, const Domain &value)
  - virtual std::vector< Domain \* > getItem (const Domain &offset) const
  - virtual Domain \* getItem (uint64\_t offset) const
  - virtual void setItem (const Domain &offset, const Domain &value)
  - virtual void setItem (uint64\_t offset, const Domain &value)

## Public Attributes

- std::vector< Domain \* > **mMembers**
- const llvm::StructType & **mType**

## Additional Inherited Members

### 16.39.1 Member Function Documentation

#### 16.39.1.1 float Canal::Structure::accuracy ( ) const [virtual]

Get accuracy of the abstract value (0 - 1). In finite-height lattices, it is determined by the position of the value in the lattice.

Accuracy 0 means that the value represents all possible values (top). Accuracy 1 means that the value represents the most precise and exact value (bottom).

Reimplemented from Canal::Domain.

---

**16.39.1.2 std::vector< Domain \* > Canal::Structure::getItem ( const Domain & offset ) const [virtual]**

Get the array items pointed by the provided offset. Returns internal array items that are owned by the array. Caller must not delete the items.

Implements Canal::Array::Interface.

**16.39.1.3 Domain \* Canal::Structure::getItem ( uint64\_t offset ) const [virtual]**

Get the array item pointed by the provided offset. Returns internal array item that is owned by the array. Caller must not delete the item.

#### Note

The uint64\_t offset variant exists because of the extractvalue instruction, which provides exact numeric offsets.

For future array domains it might be necessary to extend this method to return a list of values.

Implements Canal::Array::Interface.

**16.39.1.4 bool Canal::Structure::operator== ( const Domain & value ) const [virtual]**

Implementing this is mandatory. Values are compared while computing the fixed point.

Implements Canal::Domain.

**16.39.1.5 void Canal::Structure::setItem ( const Domain & offset, const Domain & value ) [virtual]**

#### Parameters

<i>value</i>	The method does not take the ownership of this memory. It copies the contents of the value instead.
--------------	---

Implements Canal::Array::Interface.

**16.39.1.6 void Canal::Structure::setItem ( uint64\_t offset, const Domain & value ) [virtual]**

#### Parameters

<i>value</i>	The method does not take the ownership of this memory. It copies the contents of the value instead.
--------------	---

#### Note

The uint64\_t offset variant exists because of the insertvalue instruction, which provides exact numeric offsets.

Implements Canal::Array::Interface.

### 16.39.1.7 void Canal::Structure::setZero ( const llvm::Value \* place ) [virtual]

Set value of this domain to represent zeroed memory. Needed for constants with zero initializer. This can only be called right after creating a domain.

Implements Canal::Domain.

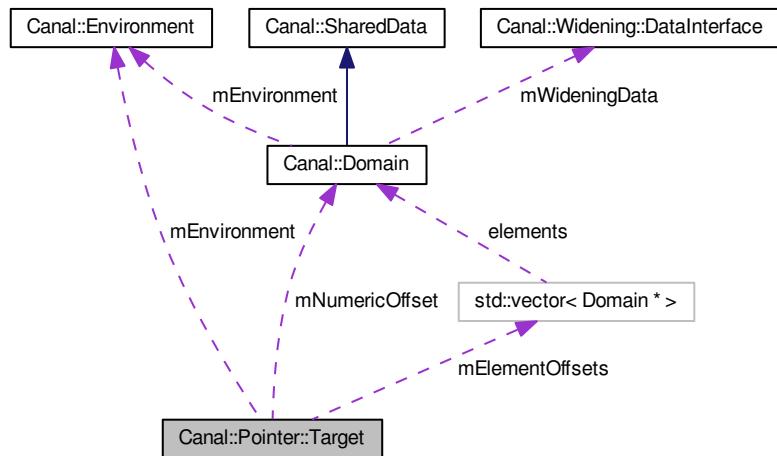
The documentation for this class was generated from the following files:

- lib/Structure.h
- lib/Structure.cpp

## 16.40 Canal::Pointer::Target Class Reference

```
#include <PointerTarget.h>
```

Collaboration diagram for Canal::Pointer::Target:



### Public Types

- enum **Type** { **Constant**, **Block**, **Function** }

### Public Member Functions

- `Target (const Environment &environment, Type type, const llvm::Value *target, const std::vector< Domain * > &elementOffsets, Domain *numericOffset)`
- `Target (const Target &target)`

*Copy constructor.*
- `~Target ()`

*Standard destructor.*

- bool **operator==** (const Target &target) const
- bool **operator!=** (const Target &target) const
- void merge (const Target &target)
 

*Merge another target into this one.*
- size\_t memoryUsage () const
 

*Get memory usage (used byte count) of this value.*
- std::string toString (SlotTracker &slotTracker) const
 

*Get a string representation of the target.*

## Public Attributes

- const Environment & **mEnvironment**
- Type **mType**

*Type of the target.*
- const llvm::Value \* **mTarget**
- std::vector< Domain \* > **mElementOffsets**
- Domain \* **mNumericOffset**

### 16.40.1 Detailed Description

Pointer target – where the pointer points to. Pointer can:

- be set to a fixed constant (memory area), such as 0xbaadfood
- point to a heap object (global block)
- point to a stack object (alloca, function block) Pointer can point to some offset in an array.

### 16.40.2 Constructor & Destructor Documentation

```
16.40.2.1 Canal::Pointer::Target ( const Environment & environment, Type type, const
                                   llvm::Value * target, const std::vector< Domain * > & elementOffsets, Domain * numericOffset
                                 )
```

Standard constructor.

#### Parameters

<i>type</i>	Type of the referenced memory.
<i>target</i>	Represents the target memory block or function. If type is Constant, it must be NULL. If type is Function, it must be a pointer to a function. Otherwise, it must be a valid pointer to an instruction. The instruction pointer is a key to State::mFunctionBlocks or State::mGlobalBlocks.
<i>offsets</i>	Offsets in the getelementptr style. The provided vector might be empty. The newly created pointer target becomes the owner of the objects in the vector.
<i>numericOffset</i>	Numerical offset that is used in addition to the getelementptr style offset and after they have been applied. It might be NULL, which indicates the offset 0. The target becomes the owner of the numerical offset when it's provided. This parameter is mandatory for pointers of Constant type, because it contains the constant.

### 16.40.3 Member Data Documentation

#### 16.40.3.1 std::vector<Domain\*> Canal::Pointer::Target::mElementOffsets

Array or struct offsets in the GetElementPtr style. This class owns the memory. If the vector is empty, mNumericOffset is considered. mElementOffsets and mNumericOffset are never used together.

#### 16.40.3.2 Domain\* Canal::Pointer::Target::mNumericOffset

A numeric offset. The value represents a number of bytes from the beginning of the object block. This class owns the memory. It might be NULL instead of 0.

#### 16.40.3.3 const llvm::Value\* Canal::Pointer::Target::mTarget

Valid when the target represents a memory block or function. For a memory block, this is a key to either State::mGlobalBlocks or State::mFunctionBlocks. The referenced llvm::Value instance is owned by the LLVM framework and not by this class.

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

- lib/PointerTarget.h
- lib/PointerTarget.cpp

## 16.41 Canal::APIntUtils::UCompare Struct Reference

### Public Member Functions

- bool **operator()** (const llvm::APInt &a, const llvm::APInt &b) const

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

- lib/APIntUtils.h

## 16.42 Canal::Pointer::Utils Class Reference

### Static Public Member Functions

- static void **addTarget** (Domain &pointer, Target::Type type, const llvm::Value \*place, const llvm::Value \*target, const std::vector< Domain \* > &offsets, Domain \*numericOffset)

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

- lib/PointerUtils.h
- lib/PointerUtils.cpp

## 16.43 Canal::VariableArguments Class Reference

### Public Member Functions

- VariableArguments ()  
*Standard constructor.*
- VariableArguments (const VariableArguments &arguments)  
*Copy constructor. Makes a deep copy of all arguments.*
- ~VariableArguments ()  
*Standard destructor. Deletes all arguments.*
- bool operator==(const VariableArguments &arguments) const
- void merge (const VariableArguments &arguments)  
*Merges the arguments per every instruction.*
- void addArgument (const llvm::Instruction &place, Domain \*argument)

#### 16.43.1 Member Function Documentation

##### 16.43.1.1 void Canal::VariableArguments::addArgument ( const llvm::Instruction & place, Domain \* argument )

Adds an argument at the end of the argument list for an instruction.

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

- lib/VariableArguments.h
- lib/VariableArguments.cpp



# Chapter 17

## Tool Class Index

### 17.1 Class Hierarchy

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

Arguments . . . . .	121
Command . . . . .	123
CommandBreak . . . . .	125
CommandCd . . . . .	126
CommandContinue . . . . .	128
CommandDump . . . . .	129
CommandFile . . . . .	131
CommandFinish . . . . .	132
CommandHelp . . . . .	134
CommandInfo . . . . .	135
CommandPrint . . . . .	137
CommandPwd . . . . .	139
CommandQuit . . . . .	140
CommandRun . . . . .	142
CommandSet . . . . .	144
CommandShell . . . . .	146
CommandShow . . . . .	147
CommandStart . . . . .	149
CommandStep . . . . .	150
Commands . . . . .	143
FunctionDetailedInfo . . . . .	152
FunctionEntry . . . . .	152
FunctionInfo . . . . .	153
IteratorCallback[external]	
IteratorCallback . . . . .	154
LoopTree . . . . .	155
State . . . . .	155

### 17.2 Class List

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

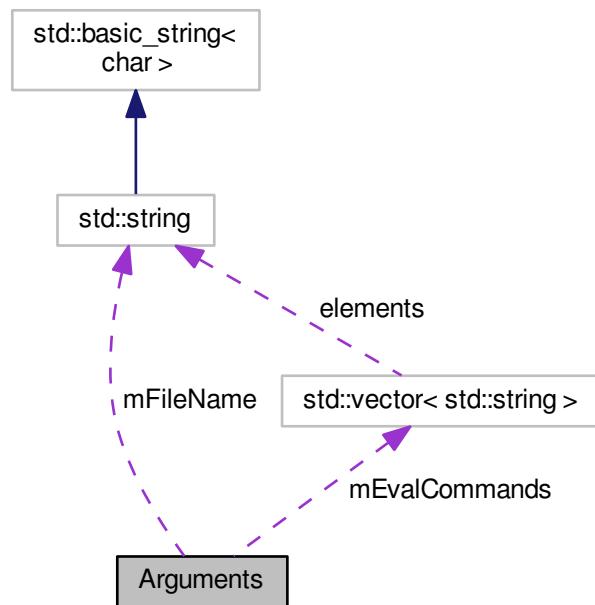
Arguments . . . . .	121
Command . . . . .	123
CommandBreak . . . . .	125
CommandCd . . . . .	126
CommandContinue . . . . .	128
CommandDump . . . . .	129
CommandFile . . . . .	131
CommandFinish . . . . .	132
CommandHelp . . . . .	134
CommandInfo . . . . .	135
CommandPrint . . . . .	137
CommandPwd . . . . .	139
CommandQuit . . . . .	140
CommandRun . . . . .	142
Commands . . . . .	143
CommandSet . . . . .	144
CommandShell . . . . .	146
CommandShow . . . . .	147
CommandStart . . . . .	149
CommandStep . . . . .	150
FunctionDetailedInfo . . . . .	152
FunctionEntry . . . . .	152
FunctionInfo . . . . .	153
IteratorCallback . . . . .	154
LoopTree . . . . .	155
State . . . . .	155

# Chapter 18

## Tool Class Documentation

### 18.1 Arguments Class Reference

Collaboration diagram for Arguments:



#### Public Attributes

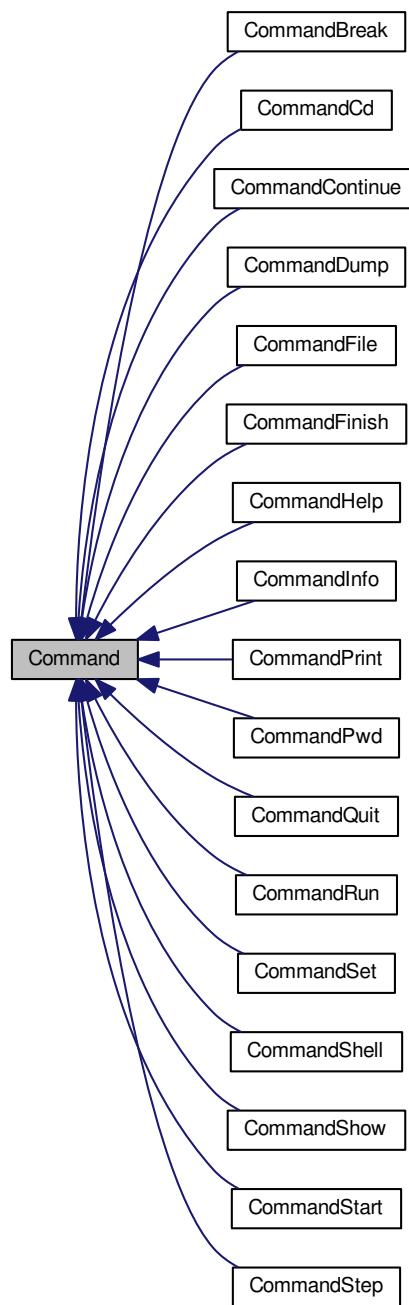
- `std::vector< std::string > mEvalCommands`
- `std::string mFileName`

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

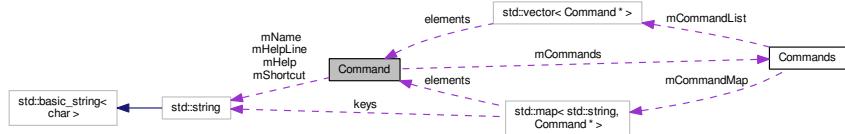
- tool/Canal.cpp

## 18.2 Command Class Reference

Inheritance diagram for Command:



Collaboration diagram for Command:



## Public Member Functions

- **Command** (const std::string &name, const std::string &shortcut, const std::string &helpLine, const std::string &help, Commands &commands)
- virtual ~Command ()  
*Standard virtual destructor.*
- const std::string & **getName** () const
- const std::string & **getShortcut** () const
- const std::string & **getHelpLine** () const
- const std::string & **getHelp** () const
- virtual std::vector< std::string > **getCompletionMatches** (const std::vector< std::string > &args, int pointArg, int pointArgOffset) const
- virtual void **run** (const std::vector< std::string > &args)=0

*Perform the command.*

## Protected Attributes

- std::string **mName**
- std::string **mShortcut**
- std::string **mHelpLine**
- std::string **mHelp**
- Commands & **mCommands**

### 18.2.1 Member Function Documentation

#### 18.2.1.1 virtual void Command::run ( const std::vector< std::string > & args ) [pure virtual]

Perform the command.

#### Parameters

args	First argument is the name of the command.
------	--

Implemented in CommandPrint, CommandSet, CommandDump, CommandCd, CommandFile, CommandHelp, CommandInfo, CommandShow, CommandBreak, CommandContinue, CommandFinish, CommandPwd, CommandQuit, CommandRun, CommandShell, CommandStart, and CommandStep.

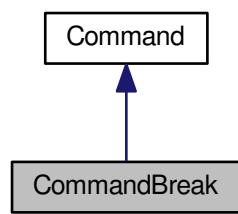
The documentation for this class was generated from the following files:

- tool/Command.h

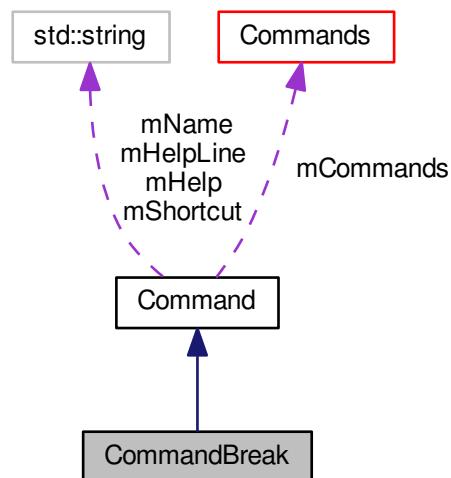
- tool/Command.cpp

## 18.3 CommandBreak Class Reference

Inheritance diagram for CommandBreak:



Collaboration diagram for CommandBreak:



### Public Member Functions

- **CommandBreak** (Commands &commands)
- virtual void **run** (const std::vector< std::string > &args)

*Perform the command.*

## Additional Inherited Members

### 18.3.1 Member Function Documentation

18.3.1.1 `void CommandBreak::run ( const std::vector< std::string > & args ) [virtual]`

Perform the command.

#### Parameters

<code>args</code>	First argument is the name of the command.
-------------------	--

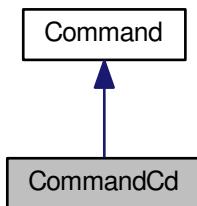
Implements Command.

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

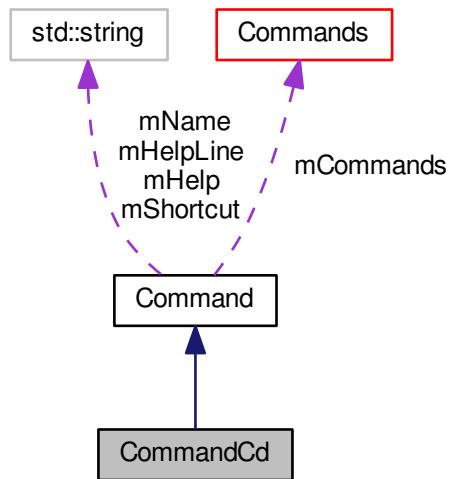
- tool/CommandBreak.h
- tool/CommandBreak.cpp

## 18.4 CommandCd Class Reference

Inheritance diagram for CommandCd:



Collaboration diagram for CommandCd:



## Public Member Functions

- **CommandCd** (`Commands &commands`)
- `virtual std::vector< std::string > getCompletionMatches (const std::vector< std::string > &args, int pointArg, int pointArgOffset) const`
- `virtual void run (const std::vector< std::string > &args)`

*Perform the command.*

## Additional Inherited Members

### 18.4.1 Member Function Documentation

#### 18.4.1.1 `void CommandCd::run ( const std::vector< std::string > & args ) [virtual]`

Perform the command.

##### Parameters

<code>args</code>	First argument is the name of the command.
-------------------	--

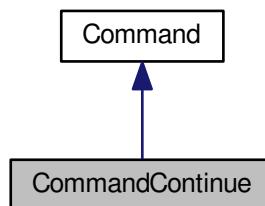
Implements `Command`.

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

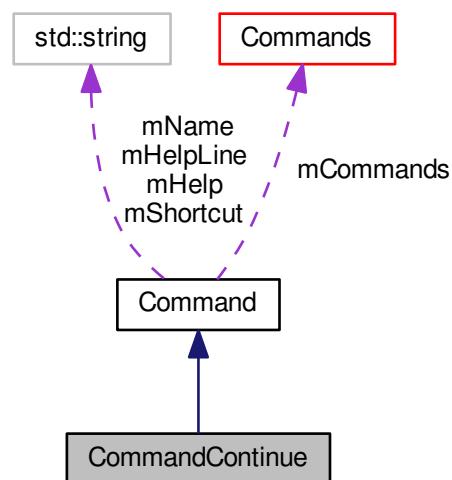
- `tool/CommandCd.h`
- `tool/CommandCd.cpp`

## 18.5 CommandContinue Class Reference

Inheritance diagram for CommandContinue:



Collaboration diagram for CommandContinue:



### Public Member Functions

- **CommandContinue (Commands &commands)**
- **virtual void run (const std::vector< std::string > &args)**

*Perform the command.*

## Additional Inherited Members

### 18.5.1 Member Function Documentation

#### 18.5.1.1 void CommandContinue::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<i>args</i>	First argument is the name of the command.
-------------	--

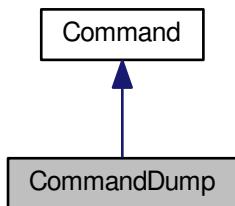
Implements Command.

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

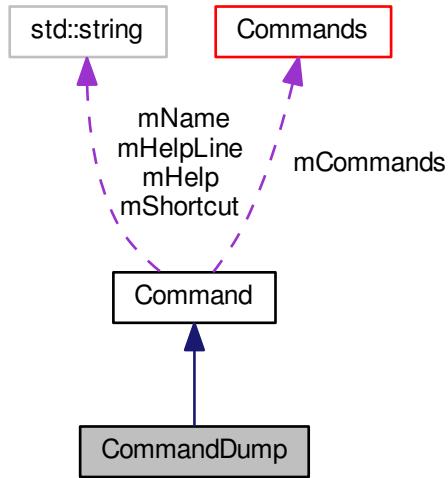
- tool/CommandContinue.h
- tool/CommandContinue.cpp

## 18.6 CommandDump Class Reference

Inheritance diagram for CommandDump:



Collaboration diagram for CommandDump:



## Public Member Functions

- **CommandDump** (Commands &commands)
- virtual std::vector< std::string > **getCompletionMatches** (const std::vector< std::string > &args, int pointArg, int pointArgOffset) const
- virtual void **run** (const std::vector< std::string > &args)

*Perform the command.*

## Additional Inherited Members

### 18.6.1 Member Function Documentation

#### 18.6.1.1 void CommandDump::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<i>args</i>	First argument is the name of the command.
-------------	--

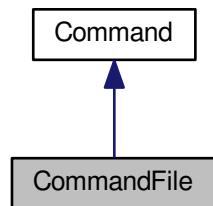
Implements Command.

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

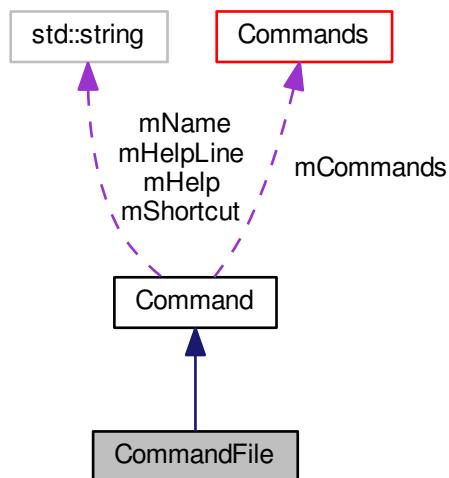
- tool/CommandDump.h
- tool/CommandDump.cpp

## 18.7 CommandFile Class Reference

Inheritance diagram for CommandFile:



Collaboration diagram for CommandFile:



### Public Member Functions

- **CommandFile (Commands &commands)**
- virtual std::vector< std::string > **getCompletionMatches** (const std::vector< std::string > &args, int pointArg, int pointArgOffset) const
- virtual void **run** (const std::vector< std::string > &args)

*Perform the command.*

## Additional Inherited Members

### 18.7.1 Member Function Documentation

#### 18.7.1.1 void CommandFile::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<i>args</i>	First argument is the name of the command.
-------------	--

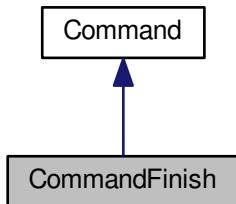
Implements Command.

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

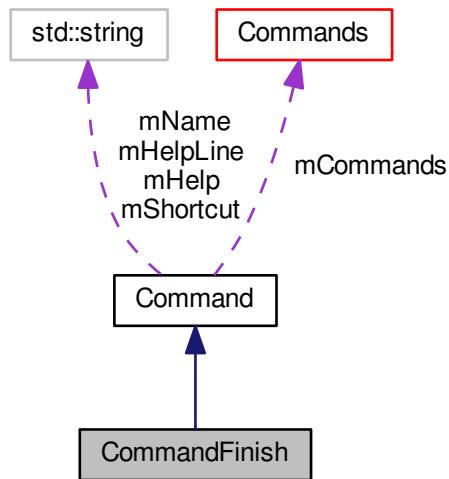
- tool/CommandFile.h
- tool/CommandFile.cpp

## 18.8 CommandFinish Class Reference

Inheritance diagram for CommandFinish:



Collaboration diagram for CommandFinish:



## Public Member Functions

- **CommandFinish** (Commands &commands)
- virtual void run (const std::vector< std::string > &args)  
*Perform the command.*

## Additional Inherited Members

### 18.8.1 Member Function Documentation

#### 18.8.1.1 void CommandFinish::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<i>args</i>	First argument is the name of the command.
-------------	--

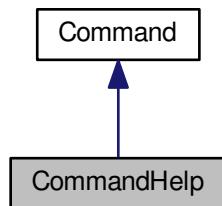
Implements Command.

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

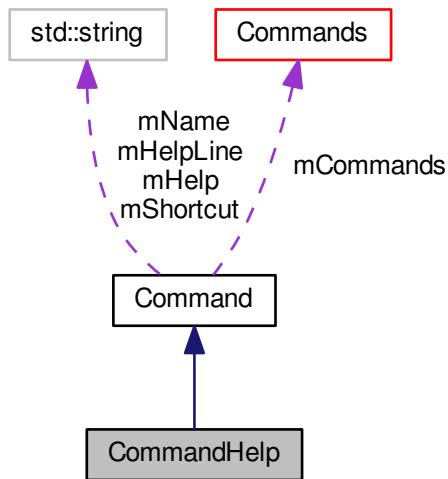
- tool/CommandFinish.h
- tool/CommandFinish.cpp

## 18.9 CommandHelp Class Reference

Inheritance diagram for CommandHelp:



Collaboration diagram for CommandHelp:



### Public Member Functions

- **CommandHelp (Commands &commands)**
- virtual std::vector< std::string > **getCompletionMatches** (const std::vector< std::string > &args, int pointArg, int pointArgOffset) const
- virtual void **run** (const std::vector< std::string > &args)

*Perform the command.*

## Additional Inherited Members

### 18.9.1 Member Function Documentation

#### 18.9.1.1 void CommandHelp::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<i>args</i>	First argument is the name of the command.
-------------	--

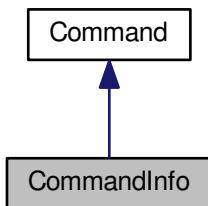
Implements Command.

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

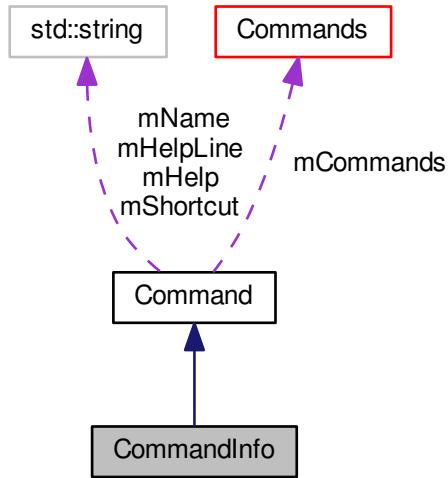
- tool/CommandHelp.h
- tool/CommandHelp.cpp

## 18.10 CommandInfo Class Reference

Inheritance diagram for CommandInfo:



Collaboration diagram for CommandInfo:



## Public Member Functions

- **CommandInfo** (Commands &commands)
- virtual std::vector< std::string > **getCompletionMatches** (const std::vector< std::string > &args, int pointArg, int pointArgOffset) const
- virtual void **run** (const std::vector< std::string > &args)

*Perform the command.*

## Additional Inherited Members

### 18.10.1 Member Function Documentation

#### 18.10.1.1 void CommandInfo::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<i>args</i>	First argument is the name of the command.
-------------	--

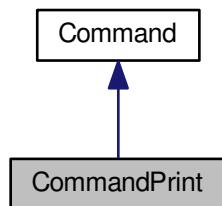
Implements Command.

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

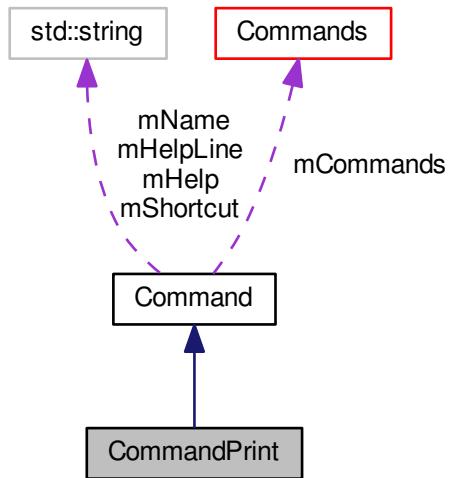
- tool/CommandInfo.h
- tool/CommandInfo.cpp

## 18.11 CommandPrint Class Reference

Inheritance diagram for CommandPrint:



Collaboration diagram for CommandPrint:



### Public Member Functions

- `CommandPrint (Commands &commands)`  
*Standard constructor.*
- `virtual std::vector< std::string > getCompletionMatches (const std::vector< std::string > &args, int pointArg, int pointArgOffset) const`
- `virtual void run (const std::vector< std::string > &args)`

*Prints current values of variables that were requested by arguments.*

## Additional Inherited Members

### 18.11.1 Member Function Documentation

**18.11.1.1 std::vector< std::string > CommandPrint::getCompletionMatches ( const std::vector< std::string > & args, int pointArg, int pointArgOffset ) const [virtual]**

#### Note

Implements Command::getCompletionMatches().

Reimplemented from Command.

**18.11.1.2 void CommandPrint::run ( const std::vector< std::string > & args ) [virtual]**

Prints current values of variables that were requested by arguments.

"@" means all global values are printed. "%" means all local variables of current function are printed.

#### Parameters

<i>args</i>	First argument is the name of the command.
-------------	--

#### Note

Implements Command::run().

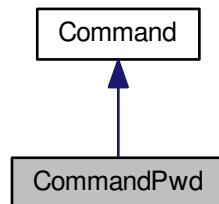
Implements Command.

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

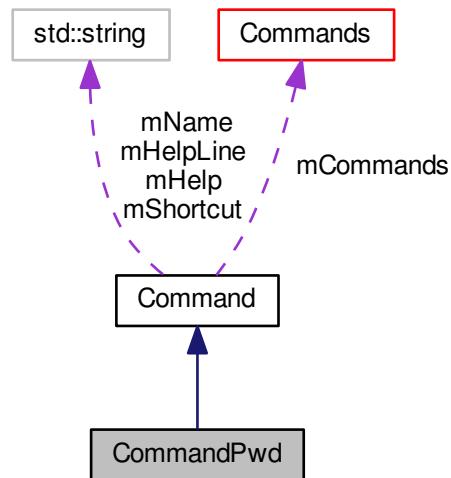
- tool/CommandPrint.h
  
- tool/CommandPrint.cpp

## 18.12 CommandPwd Class Reference

Inheritance diagram for CommandPwd:



Collaboration diagram for CommandPwd:



### Public Member Functions

- **CommandPwd (Commands &commands)**
- virtual void run (const std::vector< std::string > &args)

*Perform the command.*

## Additional Inherited Members

### 18.12.1 Member Function Documentation

#### 18.12.1.1 void CommandPwd::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<i>args</i>	First argument is the name of the command.
-------------	--

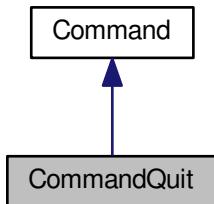
Implements Command.

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

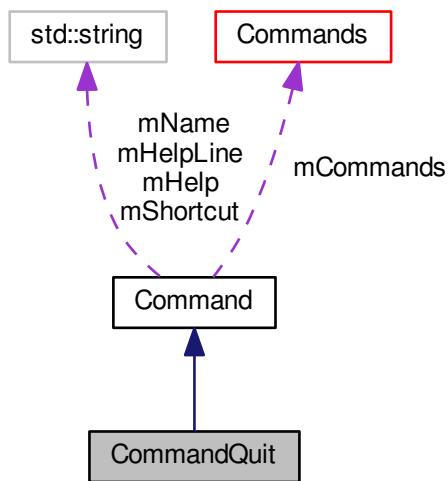
- tool/CommandPwd.h
- tool/CommandPwd.cpp

## 18.13 CommandQuit Class Reference

Inheritance diagram for CommandQuit:



Collaboration diagram for CommandQuit:



## Public Member Functions

- **CommandQuit** (Commands &commands)
- virtual void run (const std::vector< std::string > &args)  
*Perform the command.*

## Additional Inherited Members

### 18.13.1 Member Function Documentation

#### 18.13.1.1 void CommandQuit::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<i>args</i>	First argument is the name of the command.
-------------	--

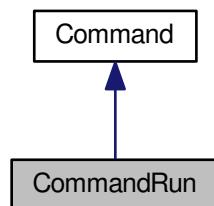
Implements Command.

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

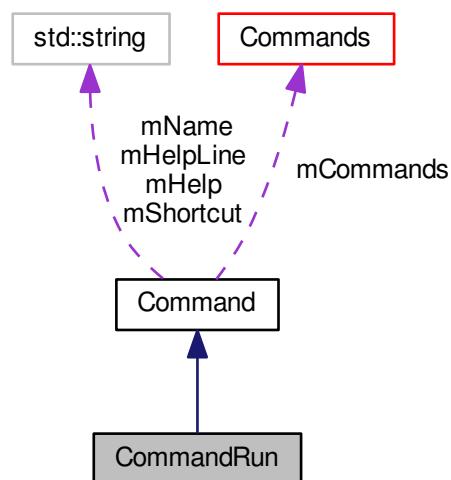
- tool/CommandQuit.h
- tool/CommandQuit.cpp

## 18.14 CommandRun Class Reference

Inheritance diagram for CommandRun:



Collaboration diagram for CommandRun:



### Public Member Functions

- **CommandRun (Commands &commands)**
- virtual void run (const std::vector< std::string > &args)

*Perform the command.*

## Additional Inherited Members

### 18.14.1 Member Function Documentation

#### 18.14.1.1 void CommandRun::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<code>args</code>	First argument is the name of the command.
-------------------	--

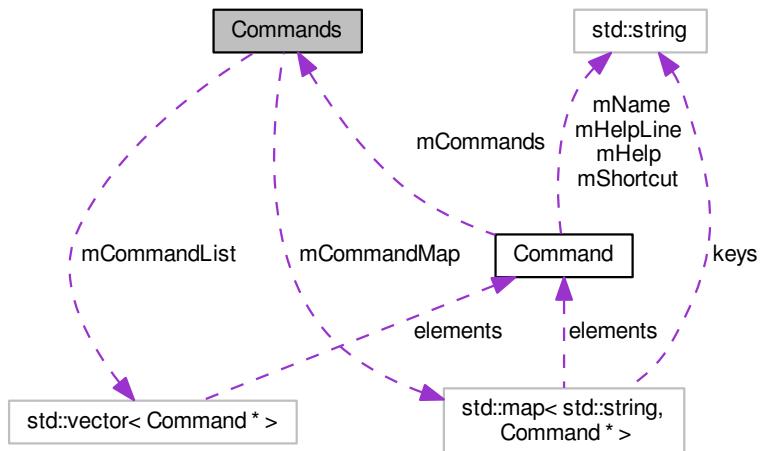
Implements Command.

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

- tool/CommandRun.h
- tool/CommandRun.cpp

## 18.15 Commands Class Reference

Collaboration diagram for Commands:



## Public Types

- `typedef std::map< std::string, Command * > CommandMap`

## Public Member Functions

- std::vector< std::string > **getCompletionMatches** (const std::string &text, int point) const
- std::vector< std::string > **getCommandMatches** (const std::string &command) const
- void **executeLine** (const std::string &line)
- Command \* **getCommand** (const std::string &name)
- const Command \* **getCommand** (const std::string &name) const
- State \* **getState** ()
- const State \* **getState** () const
- void **createState** (llvm::Module \*module)

## Public Attributes

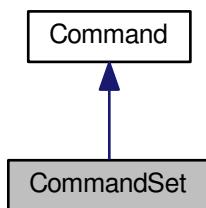
- std::vector< Command \* > **mCommandList**
- CommandMap **mCommandMap**

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

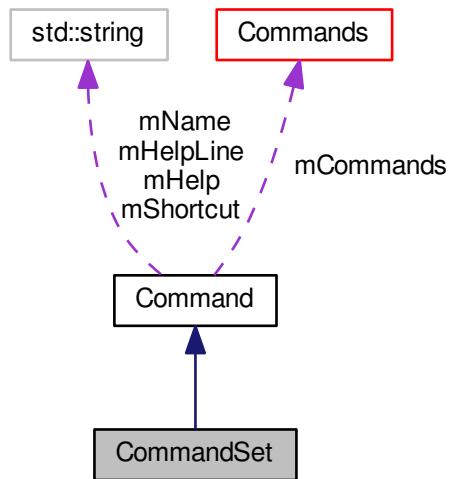
- tool/Commands.h
- tool/Commands.cpp

## 18.16 CommandSet Class Reference

Inheritance diagram for CommandSet:



Collaboration diagram for CommandSet:



## Public Member Functions

- **CommandSet (Commands &commands)**
- virtual std::vector< std::string > **getCompletionMatches** (const std::vector< std::string > &args, int pointArg, int pointArgOffset) const
- virtual void **run** (const std::vector< std::string > &args)

*Perform the command.*

## Additional Inherited Members

### 18.16.1 Member Function Documentation

#### 18.16.1.1 void CommandSet::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<i>args</i>	First argument is the name of the command.
-------------	--

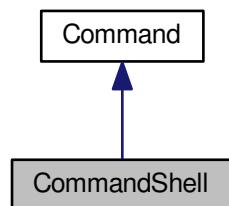
Implements Command.

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

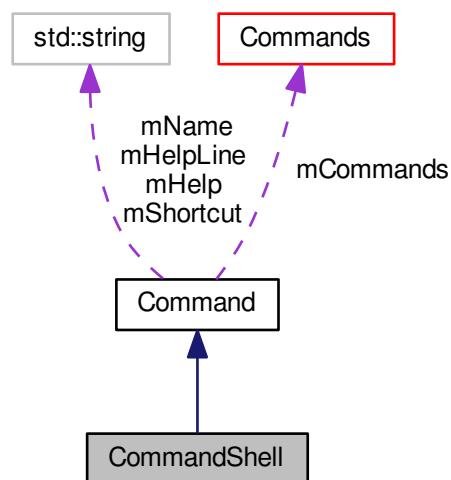
- tool/CommandSet.h
- tool/CommandSet.cpp

## 18.17 CommandShell Class Reference

Inheritance diagram for CommandShell:



Collaboration diagram for CommandShell:



### Public Member Functions

- **CommandShell** (Commands &commands)
- virtual void run (const std::vector< std::string > &args)

*Perform the command.*

## Additional Inherited Members

### 18.17.1 Member Function Documentation

#### 18.17.1.1 void CommandShell::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<i>args</i>	First argument is the name of the command.
-------------	--

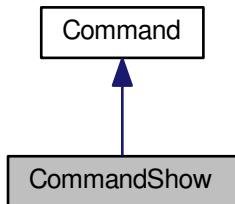
Implements Command.

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

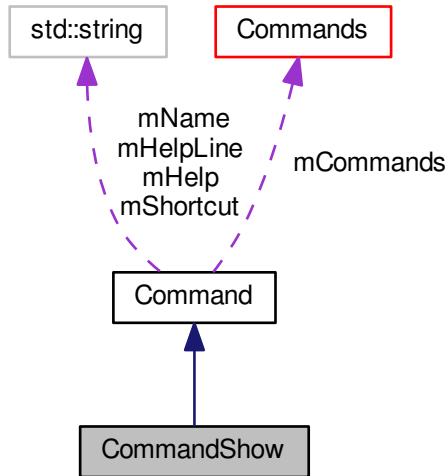
- tool/CommandShell.h
- tool/CommandShell.cpp

## 18.18 CommandShow Class Reference

Inheritance diagram for CommandShow:



Collaboration diagram for CommandShow:



## Public Member Functions

- **CommandShow (Commands &commands)**
- virtual std::vector< std::string > **getCompletionMatches** (const std::vector< std::string > &args, int pointArg, int pointArgOffset) const
- virtual void **run** (const std::vector< std::string > &args)

*Perform the command.*

## Additional Inherited Members

### 18.18.1 Member Function Documentation

#### 18.18.1.1 void CommandShow::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<i>args</i>	First argument is the name of the command.
-------------	--

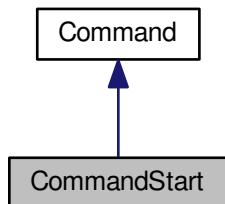
Implements Command.

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

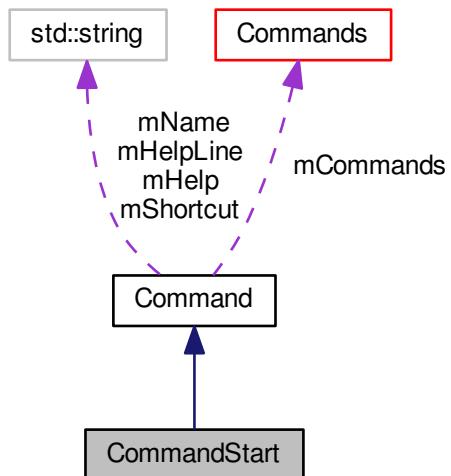
- tool/CommandShow.h
- tool/CommandShow.cpp

## 18.19 CommandStart Class Reference

Inheritance diagram for CommandStart:



Collaboration diagram for CommandStart:



### Public Member Functions

- **CommandStart (Commands &commands)**
- **virtual void run (const std::vector< std::string > &args)**

*Perform the command.*

## Additional Inherited Members

### 18.19.1 Member Function Documentation

#### 18.19.1.1 void CommandStart::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<i>args</i>	First argument is the name of the command.
-------------	--

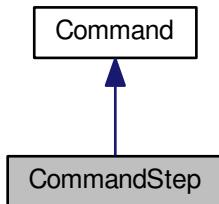
Implements Command.

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

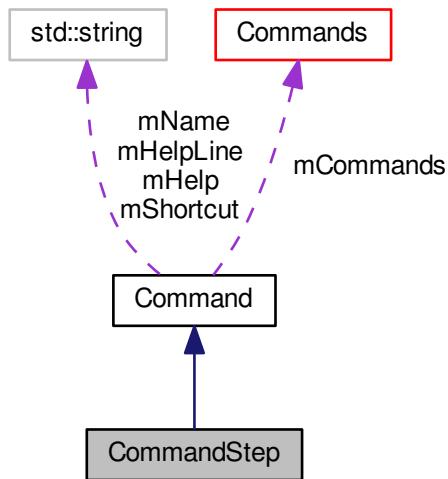
- tool/CommandStart.h
- tool/CommandStart.cpp

## 18.20 CommandStep Class Reference

Inheritance diagram for CommandStep:



Collaboration diagram for CommandStep:



## Public Member Functions

- **CommandStep** (Commands &commands)
- virtual void run (const std::vector< std::string > &args)  
*Perform the command.*

## Additional Inherited Members

### 18.20.1 Member Function Documentation

#### 18.20.1.1 void CommandStep::run ( const std::vector< std::string > & args ) [virtual]

Perform the command.

##### Parameters

<code>args</code>	First argument is the name of the command.
-------------------	--

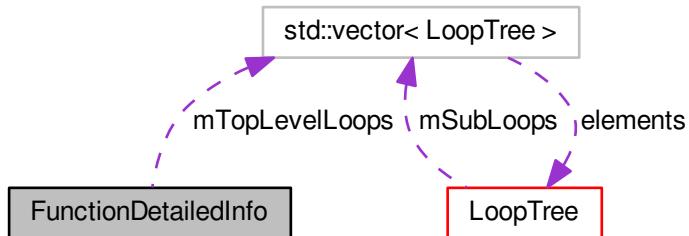
Implements Command.

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

- tool/CommandStep.h
- tool/CommandStep.cpp

## 18.21 FunctionDetailedInfo Struct Reference

Collaboration diagram for FunctionDetailedInfo:



### Public Member Functions

- virtual bool **runOnFunction** (llvm::Function &function)
- virtual void **getAnalysisUsage** (llvm::AnalysisUsage &analysisUsage) const

### Public Attributes

- llvm::Function \* **mFunction**
- std::vector< LoopTree > **mTopLevelLoops**

### Static Public Attributes

- static char **ID** = 0

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

- tool/CommandInfo.cpp

## 18.22 FunctionEntry Struct Reference

### Public Attributes

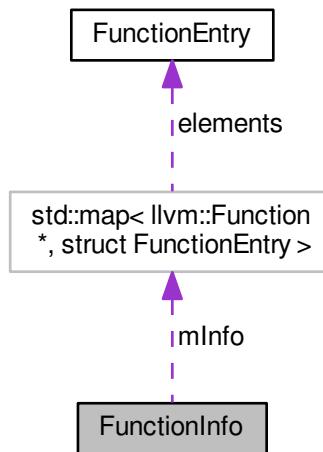
- size\_t **mNaturalLoopCount**
- unsigned **mCallsCount**
- unsigned **mCalledCount**

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

- tool/CommandInfo.cpp

## 18.23 FunctionInfo Class Reference

Collaboration diagram for FunctionInfo:



### Public Member Functions

- virtual bool **runOnFunction** (llvm::Function &function)
- virtual void **getAnalysisUsage** (llvm::AnalysisUsage &analysisUsage) const

### Public Attributes

- `std::map< llvm::Function *, struct FunctionEntry > mInfo`

### Static Public Attributes

- static char **ID** = 0

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

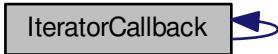
- tool/CommandInfo.cpp

## 18.24 IteratorCallback Class Reference

Inheritance diagram for IteratorCallback:



Collaboration diagram for IteratorCallback:



### Public Member Functions

- virtual void **onFixpointReached** ()
- virtual void **onFunctionEnter** (Canal::Interpreter::Function &function)
- virtual void **onBasicBlockEnter** (Canal::Interpreter::BasicBlock &basicBlock)
- virtual void **onInstructionExit** (const llvm::Instruction &instruction)
- bool **isFixpointReached** () const
- bool **isFunctionEnter** ()
- bool **isBasicBlockEnter** ()

### Protected Attributes

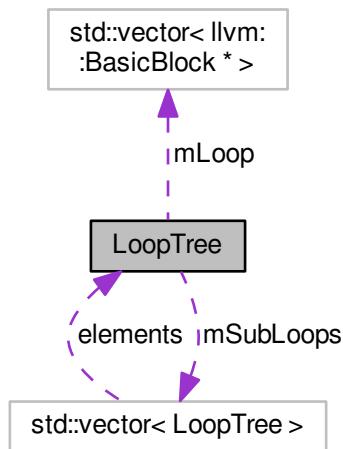
- bool **mFixpointReached**
- bool **mFunctionEnter**
- bool **mBasicBlockEnter**

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

- tool/IteratorCallback.h
- tool/IteratorCallback.cpp

## 18.25 LoopTree Class Reference

Collaboration diagram for LoopTree:



### Public Member Functions

- **LoopTree** (`llvm::Loop &loop`)
- `std::string toString` (`Canal::SlotTracker &slotTracker`) const

### Public Attributes

- `std::vector< llvm::BasicBlock * > mLoop`
- `std::vector< LoopTree > mSubLoops`

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

- `tool/CommandInfo.cpp`

## 18.26 State Class Reference

### Public Member Functions

- **State** (`llvm::Module *module`)
- `Canal::Interpreter::Interpreter & getInterpreter ()`
- `const Canal::Interpreter::Interpreter & getInterpreter () const`
- `const Canal::Environment & getEnvironment () const`

- const llvm::Module & **getModule** () const
- llvm::Module & **getModule** ()
- Canal::SlotTracker & **getSlotTracker** () const
- const IteratorCallback & **getIteratorCallback** () const
- bool **isInterpreting** () const
- void **start** ()
- void **run** ()
- void **step** (int count)
- void **finish** ()
- void **addFunctionBreakpoint** (const std::string &functionName)

### Protected Member Functions

- bool **reachedBreakpoint** ()

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

- tool/State.h
- tool/State.cpp

## **Chapter 19**

### **Known Bugs**

Pointers should have the possibility to be set to top.



# **Chapter 20**

## **Wishlist**

### **20.1 Callbacks Interface**

### **20.2 Models**

Models of functions and modules. Model of environment.

### **20.3 Support of Multiple Platforms**

Support Microsoft Windows and Mac OS X natively.

### **20.4 Automatic Tests**

Unit tests and integration tests.

### **20.5 Graphical User Interface**

Extend Eclipse to provide an user interface to Canal.



# Bibliography

- [1] Patrick Cousot and Radhia Cousot. Static Determination of Dynamic Properties of Programs. In *Proceedings of the Second International Symposium on Programming*, 1976.
- [2] Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, 1977.
- [3] Patrick Cousot and Radhia Cousot. Systematic Design of Program Analysis Frameworks. In *POPL '79: Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages*, 1979.
- [4] Patrick Cousot and Radhia Cousot. Compositional Separate Modular Static Analysis of Programs by Abstract Interpretation. In *SSGRR '01: Proceedings of the Second International Conference on Advances in Infrastructure for E-Business, E-Science and E-Education on the Internet*, 2001.
- [5] Seth Hallem, Benjamin Chelf, Yichen Xie, and Dawson Engler. A System and Language for Building System-Specific, Static Analyses. In *PLDI '02: Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation*, 2002.
- [6] Patrick Cousot and Radhia Cousot. Modular Static Program Analysis. In *CC '02: International Conference on Compiler Construction*, 2002.
- [7] Brian Albert Davey and Hilary Ann Priestley. Introduction to Lattices and Order. 2nd ed. Cambridge University Press, 2002.
- [8] Roland Backhouse, Roy Crole, and Jeremy Gibbons, eds. Algebraic and Coalgebraic Methods in the Mathematics of Program Construction. Springer-Verlag, 2002.
- [9] Antoine Miné. Weakly relational numerical abstract domains. Ph.D report. 2004.
- [10] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *CGO '04: Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, 2004.
- [11] Laurent Mauborgne and Xavier Rival. Trace Partitioning in Abstract Interpretation Based Static Analyzers. In *ESOP '05: European Symposium on Programming*, 2005.
- [12] David Monniaux. The parallel implementation of the Astrée static analyzer. 2005.
- [13] Antoine Miné. Field-Sensitive Value Analysis of Embedded C Programs with Union Types and Pointer Arithmetics. In *LCTES '06: Proceedings of the 2006 ACM SIGPLAN/SIGBED conference on Language, compilers, and tool support for embedded systems*, 2006.
- [14] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. Combination of Abstractions in the ASTRÉE Static Analyzer. In *ASIAN '06: 11th Annual Asian Computing Science Conference*, 2006.

- [15] Antoine Miné. Static Analysis of Run-time Errors in Embedded Critical Parallel C Programs. In *ESOP '11: Proceedings of The 20th European Symposium on Programming*, 2011.
- [16] Antoine Miné. Abstract Domains for Bit-Level Machine Integer and Floating-point Operations. In *WING '12: Proceedings of The 4th International Workshop on Invariant Generation*, 2012.  
Boxes: A Symbolic Abstract Domain of Boxes Arie Gurfinkel and Sagar Chaki  
SubPolyhedra: A family of numerical abstract domains for the (more) scalable inference of linear inequalities Vincent Lviron1 , Francesco Logozzo  
The Calculational Design of a Generic Abstract Interpreter Patrick COUSOT  
SAS 2012 The 19th International Static Analysis Symposium 11-13 September 2012, Deauville, France  
Patrick Cousot, Radhia Cousot and Laurent Mauborgne. A Scalable Segmented Decision Tree Abstract Domain.
- [17] Jianzhou Zhao, Santosh Nagarakatte, Milo M. K. Martin, and Steve Zdancewic. Formalizing the LLVM Intermediate Representation for Verified Program Transformations. In *POPL '12: Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2012.

# Index

accuracy  
    Canal::Array::ExactSize, 58  
    Canal::Array::SingleItem, 96  
    Canal::Array::StringPrefix, 105  
    Canal::Domain, 54  
    Canal::Float::Interval, 71  
    Canal::Integer::Bitfield, 42  
    Canal::Integer::Container, 48  
    Canal::Integer::Interval, 67  
    Canal::Integer::Set, 90  
    Canal::Structure, 108  
addArgument  
    Canal::VariableArguments, 113  
addFunctionVariable  
    Canal::State, 100  
addGlobalVariable  
    Canal::State, 101  
addTarget  
    Canal::Pointer::Pointer, 85  
Arguments, 117  
  
br  
    Canal::Operations, 79  
  
Canal::APIntUtils::SCompare, 87  
Canal::APIntUtils::UCompare, 112  
Canal::Array::ExactSize, 55  
    accuracy, 58  
    ExactSize, 58  
    getItem, 58  
    operator==, 58  
    setItem, 58, 59  
    setZero, 59  
Canal::Array::Interface, 61  
    getItem, 61  
    getValue, 62  
    setItem, 62  
Canal::Array::SingleItem, 93  
    accuracy, 96  
    getItem, 96, 97  
    mSize, 98  
    operator==, 97  
    setItem, 97  
    setZero, 97  
    SingleItem, 96  
  
Canal::Array::StringPrefix, 102  
    accuracy, 105  
    getItem, 105  
    operator==, 105  
    setItem, 105, 106  
    setZero, 106  
    StringPrefix, 105  
Canal::Constructors, 45  
    create, 46  
Canal::Domain, 51  
    accuracy, 54  
    Domain, 54  
    operator==, 54  
    setZero, 54  
Canal::Environment, 54  
Canal::Float::Interval, 69  
    accuracy, 71  
    operator==, 71  
    setZero, 71  
Canal::FunctionModel, 60  
Canal::FunctionModelManager, 60  
Canal::Integer::Bitfield, 39  
    accuracy, 42  
    getBitValue, 42  
    icmp, 42  
    mOnes, 44  
    mZeroes, 44  
    operator==, 43  
    setBitValue, 43  
    setZero, 43  
    signedMax, 43  
    signedMin, 43  
    unsignedMax, 44  
    unsignedMin, 44  
Canal::Integer::Container, 46  
    accuracy, 48  
    operator==, 48  
    setZero, 49  
Canal::Integer::Interval, 64  
    accuracy, 67  
    Interval, 67  
    isConstant, 67  
    mEmpty, 69  
    operator==, 67

setZero, 67  
 signedMax, 68  
 signedMin, 68  
 unsignedMax, 68  
 unsignedMin, 68  
 Canal::Integer::Set, 88  
     accuracy, 90  
     operator==, 90  
     setZero, 91  
     signedMax, 91  
     signedMin, 91  
     unsignedMax, 91  
     unsignedMin, 91  
 Canal::Interpreter::BasicBlock, 39  
 Canal::Interpreter::Function, 59  
     initializeInputState, 60  
 Canal::Interpreter::Interpreter, 63  
     Interpreter, 64  
 Canal::Interpreter::Iterator, 72  
     interpretInstruction, 72  
 Canal::Interpreter::IteratorCallback, 72  
 Canal::Interpreter::Module, 74  
 Canal::Interpreter::OperationsCallback, 82  
     onFunctionCall, 83  
 Canal::Operations, 75  
     br, 79  
     fadd, 79  
     fdiv, 79  
     fsub, 79  
     indirectbr, 79  
     invoke, 79  
     resume, 79  
     ret, 79  
     sdiv, 79  
     switch\_, 80  
     udiv, 80  
     unreachable, 80  
     variableOrConstant, 80  
 Canal::OperationsCallback, 81  
     onFunctionCall, 81  
 Canal::Pointer::Pointer, 83  
     addTarget, 85  
     getElementPtr, 85  
     load, 85  
     mTargets, 86  
     mType, 86  
     operator==, 86  
     Pointer, 85  
     setZero, 86  
 Canal::Pointer::Target, 110  
     mElementOffsets, 112  
     mNumericOffset, 112  
     mTarget, 112  
     Target, 111  
     Canal::Pointer::Utils, 112  
     Canal::SharedData, 92  
         SharedData, 93  
     Canal::SharedDataPointer< T >, 93  
     Canal::SlotTracker, 98  
         getLocalSlot, 99  
         processFunction, 99  
         processModule, 99  
         setActiveFunction, 99  
 Canal::State, 99  
     addFunctionVariable, 100  
     addGlobalVariable, 101  
     findBlock, 101  
     findVariable, 101  
     mergeForeignFunctionBlocks, 101  
 Canal::StateMap, 101  
 Canal::StringStream, 106  
 Canal::Structure, 107  
     accuracy, 108  
     getItem, 108, 109  
     operator==, 109  
     setItem, 109  
     setZero, 109  
 Canal::VariableArguments, 113  
     addArgument, 113  
 Canal::Widening::DataInterface, 49  
 Canal::Widening::DataIterationCount, 50  
 Canal::Widening::Interface, 63  
 Canal::Widening::Manager, 73  
 Canal::Widening::NumericalInfinity, 75  
 Canal::Widening::Pointers, 87  
 Command, 119  
     run, 120  
 CommandBreak, 121  
     run, 122  
 CommandCd, 122  
     run, 123  
 CommandContinue, 124  
     run, 125  
 CommandDump, 125  
     run, 126  
 CommandFile, 127  
     run, 128  
 CommandFinish, 128  
     run, 129  
 CommandHelp, 130  
     run, 131  
 CommandInfo, 131  
     run, 132  
 CommandPrint, 133  
     getCompletionMatches, 134  
     run, 134  
 CommandPwd, 135  
     run, 136

CommandQuit, 136  
    run, 137  
CommandRun, 138  
    run, 139  
CommandSet, 140  
    run, 141  
CommandShell, 142  
    run, 143  
CommandShow, 143  
    run, 144  
CommandStart, 145  
    run, 146  
CommandStep, 146  
    run, 147  
Commands, 139  
create  
    Canal::Constructors, 46

Domain  
    Canal::Domain, 54

ExactSize  
    Canal::Array::ExactSize, 58

fadd  
    Canal::Operations, 79

fdiv  
    Canal::Operations, 79

findBlock  
    Canal::State, 101

findVariable  
    Canal::State, 101

fsub  
    Canal::Operations, 79

FunctionDetailedInfo, 148

FunctionEntry, 148

FunctionInfo, 149

getBitValue  
    Canal::Integer::Bitfield, 42

getCompletionMatches  
    CommandPrint, 134

getElementPtr  
    Canal::Pointer::Pointer, 85

getItem  
    Canal::Array::ExactSize, 58  
    Canal::Array::Interface, 61  
    Canal::Array::SingleItem, 96, 97  
    Canal::Array::StringPrefix, 105  
    Canal::Structure, 108, 109

getLocalSlot  
    Canal::SlotTracker, 99

getValue  
    Canal::Array::Interface, 62

icmp  
    Canal::Integer::Bitfield, 42

indirectbr  
    Canal::Operations, 79

initializeInputState  
    Canal::Interpreter::Function, 60

interpretInstruction  
    Canal::Interpreter::Iterator, 72

Interpreter  
    Canal::Interpreter::Interpreter, 64

Interval  
    Canal::Integer::Interval, 67

invoke  
    Canal::Operations, 79

isConstant  
    Canal::Integer::Interval, 67

IteratorCallback, 150

load  
    Canal::Pointer::Pointer, 85

LoopTree, 151

mElementOffsets  
    Canal::Pointer::Target, 112

mEmpty  
    Canal::Integer::Interval, 69

mNumericOffset  
    Canal::Pointer::Target, 112

mOnes  
    Canal::Integer::Bitfield, 44

mSize  
    Canal::Array::SingleItem, 98

mTarget  
    Canal::Pointer::Target, 112

mTargets  
    Canal::Pointer::Pointer, 86

mType  
    Canal::Pointer::Pointer, 86

mZeroes  
    Canal::Integer::Bitfield, 44

MemoryOverlay, 74

mergeForeignFunctionBlocks  
    Canal::State, 101

onFunctionCall  
    Canal::Interpreter::OperationsCallback, 83  
    Canal::OperationsCallback, 81

operator==  
    Canal::Array::ExactSize, 58  
    Canal::Array::SingleItem, 97  
    Canal::Array::StringPrefix, 105  
    Canal::Domain, 54  
    Canal::Float::Interval, 71  
    Canal::Integer::Bitfield, 43

Canal::Integer::Container, 48  
 Canal::Integer::Interval, 67  
 Canal::Integer::Set, 90  
 Canal::Pointer::Pointer, 86  
 Canal::Structure, 109  
**Pointer**  
 Canal::Pointer::Pointer, 85  
**processFunction**  
 Canal::SlotTracker, 99  
**processModule**  
 Canal::SlotTracker, 99  
**resume**  
 Canal::Operations, 79  
**ret**  
 Canal::Operations, 79  
**run**  
 Command, 120  
 CommandBreak, 122  
 CommandCd, 123  
 CommandContinue, 125  
 CommandDump, 126  
 CommandFile, 128  
 CommandFinish, 129  
 CommandHelp, 131  
 CommandInfo, 132  
 CommandPrint, 134  
 CommandPwd, 136  
 CommandQuit, 137  
 CommandRun, 139  
 CommandSet, 141  
 CommandShell, 143  
 CommandShow, 144  
 CommandStart, 146  
 CommandStep, 147  
**sdiv**  
 Canal::Operations, 79  
**setActiveFunction**  
 Canal::SlotTracker, 99  
**setBitValue**  
 Canal::Integer::Bitfield, 43  
**setItem**  
 Canal::Array::ExactSize, 58, 59  
 Canal::Array::Interface, 62  
 Canal::Array::SingleItem, 97  
 Canal::Array::StringPrefix, 105, 106  
 Canal::Structure, 109  
**setZero**  
 Canal::Array::ExactSize, 59  
 Canal::Array::SingleItem, 97  
 Canal::Array::StringPrefix, 106  
 Canal::Domain, 54  
 Canal::Float::Interval, 71  
 Canal::Integer::Bitfield, 43  
 Canal::Integer::Container, 49  
 Canal::Integer::Interval, 67  
 Canal::Integer::Set, 91  
 Canal::Pointer::Pointer, 86  
 Canal::Structure, 109  
**SharedData**  
 Canal::SharedData, 93  
**signedMax**  
 Canal::Integer::Bitfield, 43  
 Canal::Integer::Interval, 68  
 Canal::Integer::Set, 91  
**signedMin**  
 Canal::Integer::Bitfield, 43  
 Canal::Integer::Interval, 68  
 Canal::Integer::Set, 91  
**SingleItem**  
 Canal::Array::SingleItem, 96  
**State**, 151  
**StringPrefix**  
 Canal::Array::StringPrefix, 105  
**switch\_**  
 Canal::Operations, 80  
**Target**  
 Canal::Pointer::Target, 111  
**udiv**  
 Canal::Operations, 80  
**unreachable**  
 Canal::Operations, 80  
**unsignedMax**  
 Canal::Integer::Bitfield, 44  
 Canal::Integer::Interval, 68  
 Canal::Integer::Set, 91  
**unsignedMin**  
 Canal::Integer::Bitfield, 44  
 Canal::Integer::Interval, 68  
 Canal::Integer::Set, 91  
**variableOrConstant**  
 Canal::Operations, 80