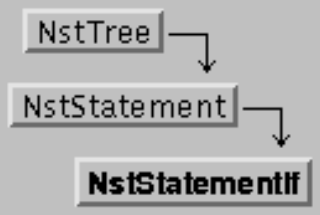
On-line documentation

class NstStatementIf : public NstStatement

The IF structure of Fortran

Inheritance:



Public Fields

- NstStatementList **else_part**
  The list of statements of the ELSE part.
- NstStatementList **then_part**
  The list of statements of the THEN part.

Public Methods

- NstStatement* **clone**(int deep = 0)
  Returns a clone of this object, recursive or not according to deep
- NstExpression* **condition**(NstExpression& new_expression)
  Changes the boolean expression of the IF statement.
- NstExpression* **condition**()
  Returns the boolean expression of the IF statement.
- void **copy**(const NstStatementIf& s, int deep = 0)
  Copies the object s into this object, recursively or not according to deep

On-line documentation (cont'd)

Documentation

The IF structure of Fortran.

This example creates a new \ctype{IF} statement with a statement \ctype{CONTINUE} in each part of the statement:

```
% cat test_if.cc
# include <libnstbase.H>
int main()
{
  NstBoolean vrai = TRUE;
  NstStatementContinue empty;
  NstStatementIf if_stat(vrai);
  if_stat.then_part.push_back(&empty);
  if_stat.else_part.push_back(&empty);
  cout << if_stat << endl;
}
% test_if
      if (.TRUE.) then
         continue
      else
         continue
      end if
```

- **NstStatementIf(NstExpression& e)**
  Creates a new IF statement
  Parameters:
  e – boolean expression of the IF.

- **NstExpression* condition()**
  Returns the boolean expression of the IF statement.

- **NstExpression* condition(NstExpression& new_expression)**
  Changes the boolean expression of the IF statement.

- **NstStatementList then_part**
  The list of statements of the THEN part.

- **NstStatementList else_part**
  The list of statements of the ELSE part.

On-line documentation (cont'd)

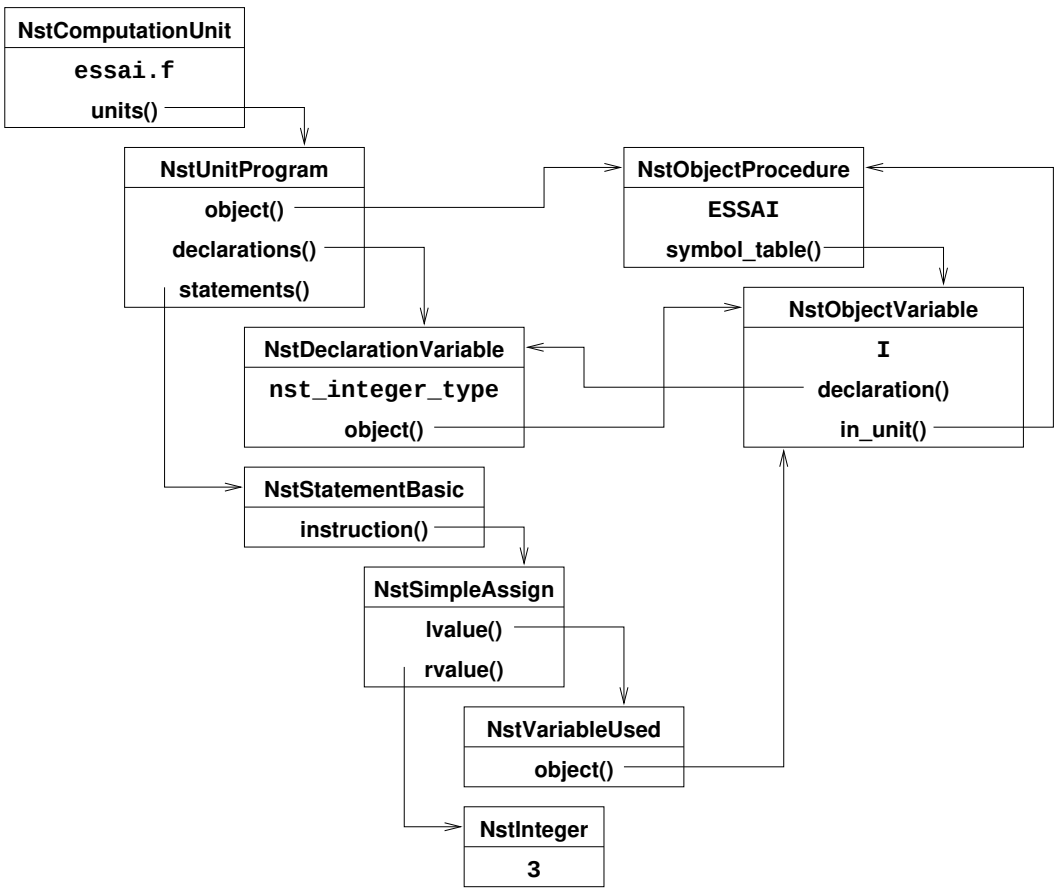
Base library

Classes and functions of the Base library.

- **The base Tree class.**
  Base class of the Base library.
- **Identifiers.**
  Identifiers in the Base library.
- **Computation unit (file).**
  Builds a Nestor structure from a source code.
- **Units**
  Classes of the Base library representing units.
- **Declarations.**
  Classes of the Base library representing declarations.
- **Statements.**
  Classes of the Base library representing statements.
- **Expressions.**
  Classes of the Base library representing expressions.
- **Functions.**
  Functions of the Base library.
- **Distributions.**
  Classes of the Base library describing distributions.

The base library of Nestor contains the definition of all the classes representing the differents syntactic elements of HPF 2.0, Fortran 90 and Fortran 77. The main purpose of this library is to provide the user an easy framework and programming tool to write source-to-source transformations of Fortran codes. Classes defined in the base library permit to create new object, i.e. new syntactic elements and then, to create a new source code from scratch. Secondly, a special class (NstComputationUnit) is used as an interface with the node library, creating a set of objects representing the parsed source code.

Examples: creating a program (cont'd)



Allen and Kennedy (cont'd)

```
!HPF$ INDEPENDENT, NEW(nst_0)
  do I = 1,N
!HPF$ INDEPENDENT
    do J = 2,N
      nst_0(J) = A(I,J+1,1)
    end do
    do J = 2,N
!HPF$ INDEPENDENT
      do K = 2,nst_0(J)
        A(I,J,K) = B(I,J,K)+A(I,J-1,K)
      end do
    end do
!HPF$ INDEPENDENT
    do J = 2,N
!HPF$ INDEPENDENT
      do K = 2,nst_0(J)
        C(I,J,K) = A(I,J-1,K-1)+D(I,J,K)
      end do
    end do
  end do
```

Examples: creating a program

```
#include <libnstbase.H>
void main()
{
  // Create the program
  NstIdentifier id_essai("essai");
  NstUnitProgram prog_essai(id_essai);

  // Declares the variable I in prog_essai
  NstIdentifier id_i("i");
  NstDeclarationVariable decl_i(prog_essai, id_i,
                                nst_integer_type);

  // Creates an assignment instruction
  NstVariableUsed use_i(*decl_i.object());
  NstInteger three(3);
  NstSimpleAssign assignment(use_i, three);

  // Creates the statement and add to the program
  NstStatementBasic stat_assign(assignment);
  prog_essai.statements.push_back(&stat_assign);

  cout << endl << prog_essai << endl;
}
```