# Stood

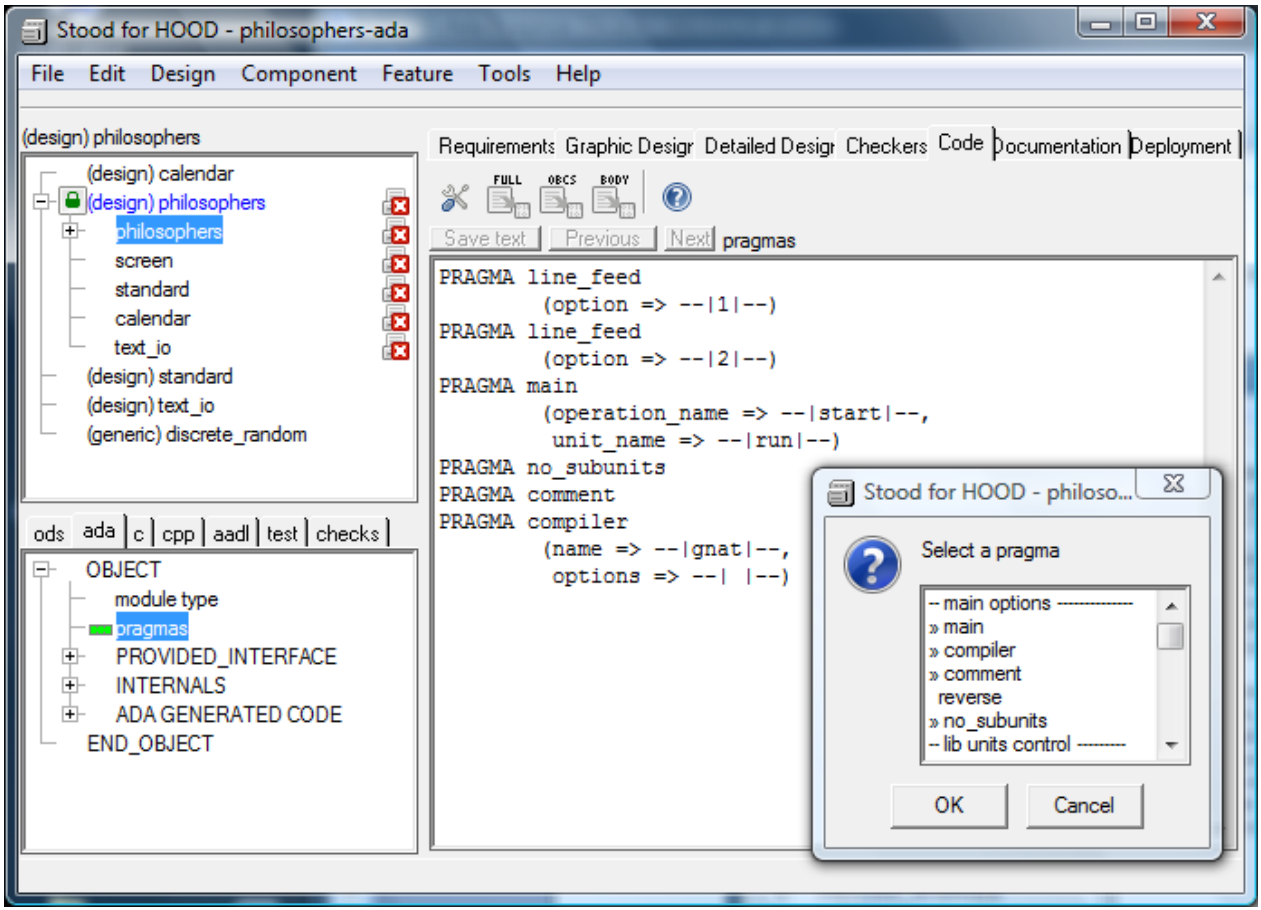
# Coding in Ada
# User Manual

Pierre Dissaux

# Contents

# 1 Introduction

This Technical Note describes briefly the 55 options (pragmas) that are available for the Ada code generator of Stood.




**Notes:**
- Global scope means that pragma effect applies to the whole Design. A global pragma should be set on the Root Component only, but will have the same effect if set on any Component.
- Local scope means that pragma effect only applies to the selected Componen.
- Restricted scope means that pragma effect only applies to the Feature of the selected Component that is specified in a pragma parameter.

# 2 Pragmas

## *2.1 Main options*

### 2.1.1. main

**Effect:**
Creates a main subprogram unit for the application. It uses the body file of root module to locate this new library unit. In consequence, it cannot be used for a terminal root module.
**Scope:**
Restricted.
**Parameters:**
- operation_name
name of a provided operation of current root unit
- unit_name
name of created subprogram unit
**Default:**
No main subprogram.
**Syntax:**

```
PRAGMA main
      (operation_name => start,
      unit_name => run)
```

### 2.1.2. compiler

**Effect:**
Controls generation of Ada Programing Environment specific features, like compilation commands. Compilation commands will be executed automatically after code generation.
**Scope:**
Global.
**Parameters:**
- name
name of selected compiling environment.
recognized values are "gnat, "aonix" (for ObjectAda), “alsys”, “tartan” and “tld”.
- options
valid command line options for the selected compiling environment.
**Default:**
Compilation command file contains only the list of produced files.
**Syntax:**

```
PRAGMA compiler
      (name => gnat,
       options => --| |--)
```

### 2.1.3. comment

**Effect:**
Inserts ODS textual descriptions as code comments.
**Scope:**
Global.
**Parameter:**
None.
**Default:**
Doesn't insert comments.
**Syntax:**
```
PRAGMA comment
```

### 2.1.4. reverse

**Effect:**
Inserts reverse coding tags into generated code, in order to be able to use "ada reversor". It may be set twice if both options are required.
**Scope:**
Global.
**Parameter:**
None.
**Default:**
No reverse coding tags are inserted.
**Syntax:**
```
PRAGMA reverse
```

### 2.1.5. no_subunits

**Effect:**
Avoids generating separate subunits for operations and Object Control Structures.
**Scope:**
Global
**Parameter:**
None.
**Default:**
Subunits are generated for each operation and Object Control Structure.
**Syntax:**
```
PRAGMA no_subunits
```

## 2.2 Library units management

### 2.2.1. except

**Effect:**
Don't generate source code for files.
**Scope:**
Local.
**Parameter:**
None.
**Default:**
Source code files are generated.
**Syntax:**
```
PRAGMA except
```

### 2.2.2. used

**Effect:**
Inserts a use clause after the with clause, each time current unit is required by another unit.
**Scope:**
Local (used unit).
**Parameter:**
None.
**Default:**
Doesn't insert use clauses.
**Syntax:**
```
PRAGMA used
```

### 2.2.3. using

**Effect:**
Inserts a use clause after the with clause, for the used unit specified as parameter.
**Scope:**
Local (using unit).
**Parameter:**
- used_unit
name of a used unit.
**Default:**
Doesn't insert use clauses.
**Syntax:**
```
PRAGMA using
       (used_unit => module_name)
```

### 2.2.4. nesting

**Effect:**
Lower level units are generated as subunits of the current unit. Specs and bodies are included into the body of the current parent unit.
**Scope:**
Local (nesting unit).
**Parameter:**
- separate_or_inside
specifies whether subunits body have to be generated as sepatate subunits (value: SEPARATE), or expended inside parent body (value: INSIDE).
**Default:**
All modules are library units.
**Syntax:**
```
PRAGMA nesting
       (separate_or_inside => SEPARATE)
```

### 2.2.5. ancillary

**Effect:**
Creates an ancillary library unit declaring separately the entity specified as parameter.
This can be used to solve some cyclic dependencies issues.
**Scope:**
Local.
**Parameters:**
- entity_name
should be typically the name a type or a constant provided by the current unit.
- unit_name
name of the ancillary unit to be created.
**Default:**
All the entities provided by a module are declared inside its spec.
**Syntax:**
```
PRAGMA ancillary
       (entity_name => resource,
        unit_name => target)
```

### 2.2.6. lib_unit_types

**Effect:**
Creates an additional library unit declaring all the types provided by the current unit.
**Scope:**
Local.
**Parameter:**
- unit_name
name of the additional unit to be created.

**Default:**
All the types provided by a unit are declared inside its spec.
**Syntax:**
```
PRAGMA lib_unit_types
       (unit_name => module_TYPES)
```

### 2.2.7. not_child_unit

**Effect:**
Doesn't translate a derived class into an Ada95 child unit.
**Scope:**
Local (child unit).
**Parameter:**
None.
**Default:**
A derived class is translated into an Ada95 child unit.
**Syntax:**
```
PRAGMA not_child_unit
```

### 2.2.8. renames

**Effect:**
Renames references to current unit name by the unit name provided as parameter.
Enables the use of short unit names. Especially useful for hierarchical libraries.
Example: (pragma renames(ada.text_io) set on library text_io)

```
  with ada.text_io;
  package body P is
    text_io renames ada.text_io;
    text_io.new_line;
  end P;
```

**Scope:**
Local (renamed unit)
**Parameter:**
- unit_name
Actual unit name that must be recognized by the compiling environment.
**Default**:
Doesn't rename existing unit name.
**Syntax:**
```
PRAGMA renames
       (unit_name => unit)
```

### 2.2.9. with

**Effect:**
Renames references to specified generic units by the specified unit name.
Used by instances of generic units.
If defined in the generic unit, will be automatically set in all instances.
Example: (pragma with(integer_io,ada.text_io) set on generic library integer_io)

```
  with ada.text_io;
  package my_io is new ada.text_io.integer_io (Num => my_int);
```

**Scope:**
Local (generic unit or instanciated unit)
**Parameters:**
- unit_name
name of the generic unit that is instanciated
- withed_unit
name of the actual unit that must be recognized by the compiling environment.
**Default:**
Doesn't rename existing generic unit name.
**Syntax:**

```
PRAGMA with
       (unit_name => no,
        withed_unit => no)
```

### 2.2.10. init_bloc

**Effect:**
Uses init_op operation code to create an initialization bloc for current unit body.
Specified operation must be defined within the current unit, and should an internal one.
**Scope:**
Restricted.
**Parameter:**
- init_op
name of the operation of current unit to be used as initialization bloc.
**Default:**
No initialization bloc is produced.
**Syntax:**

```
PRAGMA init_bloc
       (init_op => initialize)
```

## 2.3 General

### 2.3.1. overwrite

**Effect:**
Uses automatic code generation for type declarations or Object Control Structure, even if relevant code sections of the ODS are not empty.
**Scope:**
Restricted.
**Parameter:**
- component
type name as declared within current unit or "obcs"
**Default:**
Automatic code generation is used only when relevant code sections of the ODS are empty.
**Syntax:**

```
PRAGMA overwrite
      (component => type_or_obcs)
```

### 2.3.2. no_Ada95

**Effect:**
Ada83 compatibility flag.
It has only an effect on automatically generated code.
**Scope:**
Global.
**Parameter:**
None.
**Default:**
Ada95 features may be used as needed by the code generator.
**Syntax:**

```
PRAGMA no_Ada95
```

### 2.3.3. if

**Effect:**
Adds conditional instructions for code testing with TLD environment.
A TLD “pragma if” is inserted for specified internal operation or data.
**Scope:**
Local.
**Parameters:**
- option
used as a parameter for the TLD pragma.
- composant
name of an internal operation or data.

**Default:**
Doesn't generate TLD conditional compilation pragmas.
**Syntax:**
```
PRAGMA if
       (option => comp_option,
        composant => operation_or_data )
```

### 2.3.4. if_op

**Effect:**
Alternate solution to add conditional instructions for code testing with TLD environment.
A TLD pragma if is inserted for specified internal operation.
**Scope:**
Current module.
**Parameters:**
- option
used as a parameter of TLD pragma.
- composant
name of an internal operation.
**Default:**
Doesn't generate TLD conditional compilation pragmas.
**Syntax:**
```
PRAGMA if_op
       (option => comp_option,
        composant => operation_name )
```

### 2.3.5. data_page

**Effect:**
When pragma compiler(tartan) is set, adds a pragma DATA_PAGE at the end of the current package declaration.
**Scope:**
Local.
**Parameter:**
- page
page name.
**Default:**
Doesn't insert Tartan pragma DATA_PAGE.
**Syntax:**
```
PRAGMA data_page
       (page => NO_PAGE)
```

## *2.4 Object Control Structures*

### 2.4.1. package_OBCS

**Effect:**
Object Control Structure of current unit is implemented as a package subunit.
**Scope:**
Local.
**Parameter:**
None.
**Default:**
Object Control Structure of current unit is implemented as a task.
**Syntax:**
```
PRAGMA package_OBCS
```

### 2.4.2. inside_OBCS

**Effect:**
Object Control Structure body of current unit is inserted inside the package body.
**Scope:**
Local.
**Parameter:**
None.
**Default:**
Object Control Structure body of current unit is a separate subunit.
**Syntax:**
```
PRAGMA inside_OBCS
```

### 2.4.3. hidden_OBCS

**Effect:**
Inserts Object Control Structure task or package spec inside current unit body.
**Scope:**
Local.
**Parameter:**
None.
**Default:**
Inserts Object Control Structure task or package spec inside current unit spec.
**Syntax:**
```
PRAGMA hidden_OBCS
```

### 2.4.4. server_task

**Effect:**
Creates an additional task for the specified protocol constraint operation.
This server task may call a relevant Object Control Structure entry, or the Operation body directly.
**Scope:**
Restricted.
**Parameters:**
- operation_name:
name of a protocol constraint operation provided by current unit.
- task_name:
name of created server task.
- entry_call:
server task will call an Object Control Structure entry (value=OBCS)
or the Operation body directly (value=OPCS).
**Default:**
Protocol constraint operations are directly managed by the Object Control Structure.
**Syntax:**
```
PRAGMA server_task
       (operation_name => entry,
        task_name => task,
        entry_call => OBCS)
```

### 2.4.5. address_clause

**Effect:**
Inserts an Ada address clause to specified task entry or data.
A context clause is also added to get visibility on standard package SYSTEM.
**Scope:**
Restricted.
**Parameters:**
- data_or_entry:
name of the operation or data item.
- address:
name of a constant or hex value.
**Default:**
No address clause.
**Syntax:**
```
PRAGMA address_clause
       (data_or_entry => name,
        address => const_or_value)
```

### 2.4.6. storage_size

**Effect:**
Implements current Object Control Structure as an instance of a task type of specified name and storage_size attribute.
**Scope:**
Local.
**Parameters:**
- task_type_name:
name of task type to be created.
- size:
valid parameter for a storage_size attribute.
**Default:**
The ObjectControl Structure is an instance of an anonymous task type (or a package).
**Syntax:**

```
PRAGMA storage_size
       (task_type_name => task_type,
        size => value)
```

### 2.4.7. task_type

**Effect:**
Similar to pragma storage_size, except that an additional parameter is used to rename the task instance.
**Scope:**
Local.
**Parameters:**
- task_type_name:
name of task type to be created.
- size:
valid parameter for a storage_size attribute.
- task_name:
Name of the task instance.
**Default:**
The ObjectControl Structure is an instance of an anonymous task type (or a package).
**Syntax:**

```
PRAGMA task_type
       (task_type_name => task_type,
        size => no,
        task_name => OBCS)
```

### 2.4.8. interrupt_vector

**Effect:**
Inserts an Ada pragma INTERRUPT_HANDLER and an address clause for each operation provided by current unit and holding a BY_IT execution request constraint.
**Scope:**
Local.
**Parameter:**
None.
**Default:**
No Ada pragma nor address clause is generated for operations constrained BY_IT.
**Syntax:**
```
PRAGMA interrupt_vector
```

### 2.4.9. hide_IT

**Effect:**
Hides all the provided operations holding a BY_IT execution request constraint, within the body of the corresponding terminal unit, in order to avoid direct use from other units.
**Scope:**
Global.
**Parameter:**
None.
**Default:**
Provided operations holding a BY_IT constraint are inserted within the spec of the current unit.
**Syntax:**
```
PRAGMA hide_IT
```

## *2.5 Operations*

### 2.5.1. inside_op

**Effect:**
Inserts specified operation(s) body inside the body of current unit.
**Scope:**
Restricted.
**Parameter:**
- operation_name:
unique operation name or "*" to specify all operations of current unit.
**Default:**
Inserts operation body inside a separate subunit.
**Syntax:**
```
PRAGMA inside_op
      (operation_name => --|*|--)
```

### 2.5.2. separate_op

**Effect:**
Inserts the specified operation body inside a separate subunit.
Useful when pragma inside_op("*") has also been set.
**Scope:**
Restricted.
**Parameter:**
- operation_name:
unique operation name of current unit.
**Default:**
No effect on the pragma inside_op("*").
**Syntax:**

```
PRAGMA separate_op
       (operation_name => --|*|--)
```

### 2.5.3. interface

**Effect:**
Doesn't generate a body for specified operation(s) of current unit.
**Scope:**
Restricted.
**Parameter:**
- operation_name:
unique operation name or "*" to specify all operations in current unit.
**Default:**
Generates a body for each operation of current unit.
**Syntax:**

```
PRAGMA interface
       (operation_name => --|*|--)
```

### 2.5.4. null

**Effect:**
Specifies the statements to be inserted automatically when operation code section has been left empty.
**Scope:**
Global.
**Parameter:**
- contents:
valid Ada statement.
**Default:**
a null statement is inserted automatically when operation code section has been left empty.
**Syntax:**

```
PRAGMA null
      (contents => --|null;|--)
```

### 2.5.5. no_constraints

**Effect:**
Ignores operation protocol execution request constraints when generating code.
**Scope:**
Local.
**Parameter:**
None.
**Default:**
Uses operation constraints information to generate appropriate code.
**Syntax:**

```
PRAGMA no_constraints
```

### 2.5.6. operator

**Effect:**
Specified operation of current unit is generated as an Ada operator.
**Scope:**
Restricted.
**Parameter:**
- operation_name:
name of an operation of current unit.
**Default:**
Operations are not operators.
**Syntax:**

```
PRAGMA operator
      (operation_name => operator)
```

### 2.5.7. generic_op

**Effect:**
Indicates that the operation code extension section contains statements that must be inserted before the operation declaration.
This is useful for declaring generic subprograms.
**Scope:**
Restricted.
**Parameter:**
- operation_name
Name of an operation of current unit.
**Default:**
Contents of the operation code extension section (if any) is inserted after the operation declaration.

**Syntax:**

```
PRAGMA generic_op
      (operation_name => op)
```

### 2.5.8. instance_op

**Effect:**
Indicates that the operation code extension section contains statements that must be inserted instead of the operation declaration.
This is useful for declaring instances of generic subprograms.
**Scope:**
Restricted.
**Parameter:**
- operation_name:
Name of an operation of current unit.
**Default:**
Contents of the operation code extension section (if any) is inserted after the operation declaration.
**Syntax:**

```
PRAGMA instance_op
      (operation_name => op)
```

### 2.5.9. extension

**Effect:**
Indicates that the operation code extension section contains statements that must be inserted immediately after of the operation declaration, and without separator.
This is useful for renaming subprograms.
**Scope:**
Restricted.
**Parameter:**
- operation_name:
Name of an operation of current unit.
**Default:**
Contents of the operation code extension section (if any) is inserted after the operation declaration and is separated by a semicolon.
**Syntax:**

```
PRAGMA extension
      (operation_name => op)
```

## 2.6 Types and constants

### 2.6.1. no_use_type

**Effect:**
Doesn't automatically insert Ada95 use type clauses when specified type is required.
**Scope:**
Restricted.
**Parameter:**
- type_name:
name of a type of current unit, or "*" to specify all types in current unit.
**Default:**
Automatically generates a use type clause for Ada95 code when a type is required.
**Syntax:**
```
PRAGMA no_use_type
       (type_name => --|*|--)
```

### 2.6.2. const_enum

**Effect:**
Creates constants to rename enumeration elements inside current parent unit spec.
**Scope:**
Restricted.
**Parameter:**
- type_name:
name of a type of current unit, or "*" to specify all types in current unit.
**Default:**
Creates functions to rename enumeration elements inside current parent unit spec.
**Syntax:**
```
PRAGMA const_enum
       (type_name => --|*|--)
```

### 2.6.3. derived_typing

**Effect:**
Uses a derived type to rename a type defined at lower level in the hierarchy.
**Scope:**
Restricted (parent type).
**Parameter:**
- type_name:
name of a type of current parent unit, or "*" to specify all types in current parent unit.
**Default:**
Uses a subtype to rename a type defined at lower level in the hierarchy.
**Syntax:**

```
PRAGMA derived_typing
      (type_name => --|*|--)
```

### 2.6.4. subtyping

**Effect:**
Uses a subtype to rename a type defined at lower level in the hierarchy.
Useful when pragma derived_typing("*") is set.
**Scope:**
Restricted (parent type).
**Parameter:**
- type_name:
name of a type of current parent unit.
**Default:**
Uses a derived type to rename a type defined at lower level in the hierarchy when pragma derived_typing is set.
**Syntax:**

```
PRAGMA subtyping
      (type_name => --|*|--)
```

### 2.6.5. rename_constants

**Effect:**
Uses a renaming declaration to rename all constants defined at lower level in the hierarchy.
**Scope:**
Global.
**Parameter:**
None
**Default:**
Creates new local constants to rename constants defined at lower level in the hierarchy.
**Syntax :**

```
PRAGMA rename_constants
```

### 2.6.6. discriminant

**Effect:**
Uses specified attribute to create a discriminant for specified type.
**Scope:**
Restricted.
**Parameters:**
- type_name:
name of a structured type of current terminal unit.
- attribute_name:
name of an attribute of specified type.

**Default:**
All attributes are used to create record elements.
**Syntax:**
```
PRAGMA discriminant
      (type_name => type,
      attribute_name => --|id|--)
```

## *2.7 Layout*

### 2.7.1. no_indent

**Effect:**
Doesn't automatically indent generated code lines.
**Scope:**
Global.
**Parameter:**
None.
**Default:**
Automatically indents generated code lines.
**Syntax:**
```
PRAGMA no_indent
```

### 2.7.2. indent_width

**Effect:**
Specifies the number of space characters to be use for indentation level.
**Scope:**
Global.
**Parameter:**
Width:
Number of space characters.
**Default:**
Inserts 2 space characters for each indentation level.
**Syntax:**
```
PRAGMA indent_width
      (width => 2)
```

### 2.7.3. no_end_op

**Effect:**
Doesn't automatically insert a end statement in operation body.
It is thus supposed to have been entered manually inside OPCS code.

Useful when generating code after a previous reverse engineering.
**Scope:**
Global.
**Parameter:**
None.
**Default:**
Automatically inserts a end statement in each subprogram body.
**Syntax:**
```
PRAGMA no_end_op
```

### 2.7.4. cut_line

**Effect:**
Cuts specified subprogram declaration, in order to insert each parameter type and default value on a new line.
Useful when the compiler doesn't accept long lines, and to improve readability.
**Scope:**
Restricted.
**Parameter:**
- operation_name:
name of an operation of current unit, or "*" to specify all operations in current unit.
**Default:**
Parameters name, mode, type and default value are inserted on the same line.
**Syntax:**
```
PRAGMA cut_line
      (operation_name => --|*|--)
```

### 2.7.5. line_feed

**Effect:**
Adds new lines to generated code.
May be used twice with both options.
**Scope:**
Global.
**Parameter:**
- option:
adds a blank line after some automatically generated statements (value = 1).
inserts each subprogram or instance parameter on a separate line (value = 2).
**Default:**
Compact code.
**Syntax:**
```
PRAGMA line_feed
      (option => 1)
```

### 2.7.6. lower_case

**Effect:**
Writes operation parameter modes in lower case.
**Scope:**
Global.
**Parameter:**
None.
**Default:**
Writes operation parameter modes in upper case.
**Syntax:**
```
PRAGMA lower_case
```

### 2.7.7. trace

**Effect:**
Writes debugging information into the extraction messages log file.
**Scope:**
Global.
**Parameter:**
None.
**Default:**
Writes minimal information into the extraction messages log file.
**Syntax:**
```
PRAGMA trace
```

### 2.7.8. line_width

**Effect:**
Wraps comment lines at specified length.
**Scope:**
Global.
**Parameter:**
- length:
maximum length for comment lines.
**Default:**
Wraps comment lines around 80 characters.
**Syntax:**
```
PRAGMA line_width
      (length => 80)
```

### 2.7.9. sep_lines

**Effect:**
Adds a separation line at the beginning and the end of each comment.
**Scope:**
Global.
**Parameter:**
None
**Default:**
No separation line.
**Syntax:**
```
PRAGMA sep_lines
```

### 2.7.10. verbose

**Effect:**
Adds more comments.
**Scope:**
Global.
**Parameter:**
- option:
For future use. The only implemented option is for value = 1.
**Default:**
Main comments only.
**Syntax:**
```
PRAGMA verbose
      (option => 1)
```

### 2.7.11. more_headers

**Effect:**
Includes specific headers for separate subunits.
**Scope:**
Global.
**Parameter:**
None
**Default:**
No specific headers for separate subunits.
**Syntax:**
```
PRAGMA more_headers
```


www.ellidiss.com
stood@ellidiss.com

TNI Europe Limited
Triad House
Mountbatten Court
Worall Street
Congleton
Cheshire
CW12 1AG
UK

+44 1260 291 449

Ellidiss Technologies

24 quai de la douane
29200 Brest

Brittany

France

+33 298 451 870