

CLASS NAMES

Classes are groups of like data objects. All the near data objects from all modules (source files) in an entire program are collected together into a single block and given a CLASS name like "NDATA." This name is the handle by which the user may control the address where the data object will be placed by the linker.

NEAR OBJECTS:

NDATA = near data which is not initialized by the startup phase
NDATA0 = near data which is zeroed out by the startup phase

FAR OBJECTS:

FDATA = far data which is not initialized by the startup phase
FDATA0 = far data which is zeroed out by the startup phase

HUGE OBJECTS:

HDATA = huge data which is not initialized by the startup phase
HDATA0 = huge data which is zeroed out by the startup phase

XHUGE OBJECTS:

XDATA = xhuge data which is not initialized by the startup phase
XDATA0 = xhuge data which is zeroed out by the startup phase

IDATA OBJECTS:

IDATA = idata data which is not initialized by the startup phase
IDATA0 = idata data which is zeroed out by the startup phase

BDATA OBJECTS:

BDATA = bdata data which is not initialized by the startup phase
BDATA0 = bdata data which is zeroed out by the startup phase

NDATA AND NDATA0

The NOINIT #pragma

The C16x creates two varieties of each data class; NDATA and NDATA0 plus FDATA and FDATA0 etc... These should always be placed by the user at the same addresses as the basic NDATA and FDATA classes. The reason for the distinction is that the “0” suffixed classes are cleared to zero and initialized by STARTUP.A66 or START167.A66 before main() is reached. The CLRMEM constant in the startup file determines whether this clearing occurs or not.

START167.A66 Extract

```
; The following code is necessary to set RAM variables to 0 at
; start-up (RESET) of the C application program.
```

```
$IF (CLR_MEMORY = 1)
    EXTRN?C_CLRMEMSECSTART : WORD
Clr_Memory:
```

The initialization consists of either clearing to zero or the writing of start values given in a variable’s declaration, i.e.:

```
int far xvar = 0x02;
```

Objects destined for the FDATA and NDATA classes are those declared while there is a #pragma NOINIT control working. This is usually done where the data is held in non-volatile memory from the last time the program was run.

Example

```
MODULE = MAIN

int far var;           // zeroed before main()
int far var1 = 2;     // '2' written into variable before main()

#pragma NOINIT        // inhibit zeroing/initialization

int far novolvar;     // not zeroed before main()

#pragma INIT          // restore zeroing/initialization

int ordinary_var;
```

SECTIONS and CLASSES Produced as a Result of This

- nonvolvar ends up in SECTION ?FD?MAIN%FDATA
- var and var1 end up in SECTION ?FD0?MAIN%FDATA0

The precise address at which the classes are located is determined by the CLASSES and SECTIONS controls in the L166 options.

MODULES AND SECTIONS

SECTIONs allow the data from individual modules to be placed by the linker. The near constant data, NCONST class (or any other class), from a certain module can be placed at a user-defined address. This is especially useful for placing a look-up table in a FLASH EPROM, remote from the main program EPROM.

Example

A far integer is declared in a module MAIN.C:

```
int far testvar;
```

This creates a SECTION called ?FD0?MAIN%FDATA0. The construction of the section name is as follows:

```
?<classshortname>?<module name>%<classname>
```

The <classshortname> is an abbreviation for the full class name:

NDATA0	ND0
FDATA0	FD0
NDATA	ND
NCODE	PR
FCODE	PR
NCONST	NC
FCONST	FC

And so on...

Note that the section name generated for the executable code is always "PR." The conversion to NCODE and FCODE is made by the linker, depending on which memory model is used.

Example

A near integer is declared in a module, MODA:

```
int near testvar;
```

The resulting section is ?ND0?MODA%NDATA0.

To fix the classes, the CLASSES control is used:

EXEC.LIN Linker Input File:

```
main.obj, &
moda.obj &
to exec &
CLASSES(FDATA(0x8000),
        FDATA0(0x8000))
```

This fixes the far data (FDATA and FDATA0) classes at 0x8000. A further level of control over placement is possible using the SECTIONS command. This allows the addresses of a particular module's own data and code classes to be fixed within the host class's range:

Example

```
main.obj, &
moda.obj &
to exec &
CLASSES(FDATA(0x8000),
        FDATA0(0x8000))
SECTIONS(?FD0?MODA%FDATA0(0xA000))
```

This puts the far data class at 0x8000 but puts the far data objects from module MODA.C at 0xA000. L166 will then arrange far data objects from other modules so that 0xA000 remains free.

On the C167/5 with their 16MB address space, it is often helpful to split FDATA and FCODE classes across several ranges. This might be required if you have, for example, two ROM devices at different addresses. Commonly, this might be a 32k boot EPROM at 0x0000 and a 128k application FLASH EPROM at 0x40000. The CLASSES control can be used to allow the linker to fill both regions as such:

```
main.obj, &
moda.obj &
to exec &
CLASSES(FCODE(0-0x7FFFF, 0x40000-0x5FFFF),
        FDATA(0x8000),
        FDATA0(0x8000)) &
SECTIONS(?FD0?MODA%FDATA0(0xA000))
```

In the same way, FDATA can also be split. Be aware, though, that splitting the near code (NCODE) can be risky and you must make sure that you do not try to split it across a 64k segment boundary. Near function calls have no built-in segment number. This will cause your program to crash.

Copyright © 1997 Keil Software, Inc. All rights reserved.

In the USA:
Keil Software, Inc.
16990 Dallas Parkway, Suite 120
Dallas, TX 75248-1903
USA

Sales: 800-348-8051
Phone: 972-735-8052
FAX: 972-735-8055

E-mail: sales.us@keil.com
support.us@keil.com

Internet: <http://www.keil.com/>

In Europe:
Keil Elektronik GmbH
Bretonischer Ring 15
D-85630 Grasbrunn b. Munchen
Germany

Phone: (49) (089) 45 60 40 - 0
FAX: (49) (089) 46 81 62

E-mail: sales.intl@keil.com
support.intl@keil.com