CHAPTER 2

Directives

Topical Cross-reference for Directives. 22
Directives 25

Topical Cross-reference for Directives

Code Labels

ALIGN	EVEN
LABEL	ORG

Conditional Assembly

ELSE	ELSEIF	ELSEIF2
ENDIF	IF	IF2
IFB/IFNB	IFDEF/IFNDEF	IFDIF/IFDIFI
IFE	IFIDN/IFIDNI	

Conditional Control Flow

.BREAK	.CONTINUE	.ELSE
.ELSEIF	.ENDIF	.ENDW
.IF	.REPEAT	.UNTIL/
.UNTILCXZ	.WHILE	

Conditional Error

.ERR	.ERR2	.ERRB
.ERRDEF	.ERRDIF/.ERRDIFI	.ERRE
.ERRIDN/.ERRIDNI	.ERRNB	.ERRNDEF
.ERRNZ		

Data Allocation

ALIGN	BYTE/SBYTE	DWORD/SDWORD
EVEN	FWORD	LABEL
ORG	QWORD	REAL4
REAL8	REAL10	TBYTE
WORD/SWORD		

Equates

=

EQU

TEXTEQU

Listing Control

.CREF	.LIST	.LISTALL
.LISTIF	.LISTMACRO	.LISTMACROALL
.NOCREF	.NOLIST	.NOLISTIF
.NOLISTMACRO	PAGE	SUBTITLE
.TFCOND	TITLE	

Macros

ENDM	EXITM	GOTO
LOCAL	MACRO	PURGE

Miscellaneous

ASSUME	COMMENT	ECHO
END	INCLUDE	INCLUDELIB
OPTION	POPCONTEXT	PUSHCONTEXT
.RADIX		

Procedures

ENDP	INVOKE	PROC
PROTO	USES	

Processor

.186	.286	.286P
.287	.386	.386P
.387	.486	.486P
.8086	.8087	.NO87

Repeat Blocks

ENDM	FOR	FORC
GOTO	REPEAT	WHILE

Scope

COMM	EXTERN	EXTERNDEF
INCLUDELIB	PUBLIC	

Segment

.ALPHA	ASSUME	.DOSSEG
END	ENDS	GROUP
SEGMENT	.SEQ	

Simplified Segment

.CODE	.CONST	.DATA
.DATA?	.DOSSEG	.EXIT
.FARDATA	.FARDATA?	.MODEL
.STACK	.STARTUP	

String

CATSTR	INSTR
SIZESTR	SUBSTR

Structure and Record

ENDS	RECORD	STRUCT
TYPEDEF	UNION	

Directives

name = expression
Assigns the numeric value of *expression* to *name*. The symbol may be redefined later.

.186
Enables assembly of instructions for the 80186 processor; disables assembly of instructions introduced with later processors. Also enables 8087 instructions.

.286
Enables assembly of nonprivileged instructions for the 80286 processor; disables assembly of instructions introduced with later processors. Also enables 80287 instructions.

.286P
Enables assembly of all instructions (including privileged) for the 80286 processor; disables assembly of instructions introduced with later processors. Also enables 80287 instructions.

.287
Enables assembly of instructions for the 80287 coprocessor; disables assembly of instructions introduced with later coprocessors.

.386
Enables assembly of nonprivileged instructions for the 80386 processor; disables assembly of instructions introduced with later processors. Also enables 80387 instructions.

.386P
Enables assembly of all instructions (including privileged) for the 80386 processor; disables assembly of instructions introduced with later processors. Also enables 80387 instructions.

.387
Enables assembly of instructions for the 80387 coprocessor.

.486
Enables assembly of nonprivileged instructions for the 80486 processor.

.486P
Enables assembly of all instructions (including privileged) for the 80486 processor.

.8086
Enables assembly of 8086 instructions (and the identical 8088 instructions); disables assembly of instructions introduced with later processors. Also enables 8087 instructions. This is the default mode for processors.

.8087
Enables assembly of 8087 instructions; disables assembly of instructions introduced with later coprocessors. This is the default mode for coprocessors.

ALIGN [[*number*]]
Aligns the next variable or instruction on a byte that is a multiple of *number*.

.ALPHA
Orders segments alphabetically.

ASSUME *segregister*:*name* [[**,** *segregister*:*name*]]...
ASSUME *dataregister*:*type* [[**,** *dataregister*:*type*]]...
ASSUME *register*:**ERROR** [[**,** *register*:**ERROR**]]...
ASSUME [[*register*:]] **NOTHING** [[**,** *register*:**NOTHING**]]...
Enables error-checking for register values. After an **ASSUME** is put into effect, the assembler watches for changes to the values of the given registers. **ERROR** generates an error if the register is used. **NOTHING** removes register error-checking. You can combine different kinds of assumptions in one statement.

.BREAK [[**.IF** *condition*]]
Generates code to terminate a **.WHILE** or **.REPEAT** block if *condition* is true.

[[*name*]] **BYTE** *initializer* [[**,** *initializer*]]...
Allocates and optionally initializes a byte of storage for each *initializer*. Can also be used as a type specifier anywhere a type is legal.

name **CATSTR** [[*textitem1* [[**,** *textitem2*]]...]]
Concatenates text items. Each text item can be a literal string, a constant preceded by a **%**, or the string returned by a macro function.

.CODE [[*name*]]
When used with **.MODEL**, indicates the start of a code segment called *name* (the default segment name is _TEXT for tiny, small, compact, and flat models, or *module*_TEXT for other models).

COMM *definition* [[**,** *definition*]]...
Creates a communal variable with the attributes specified in *definition*. Each *definition* has the following form:

[[*langtype*]] [[**NEAR** | **FAR**]] *label*:*type*[[:*count*]]

The *label* is the name of the variable. The *type* can be any type specifier (**BYTE**, **WORD**, and so on) or an integer specifying the number of bytes. The *count* specifies the number of data objects (one is the default).

COMMENT *delimiter* [[*text*]]
[[*text*]]

[[*text*]] *delimiter* [[*text*]]

Treats all *text* between or on the same line as the delimiters as a comment.

.CONST
When used with **.MODEL**, starts a constant data segment (with segment name CONST). This segment has the read-only attribute.

.CONTINUE [[**.IF** *condition*]]
Generates code to jump to the top of a **.WHILE** or **.REPEAT** block if *condition* is true.

.CREF
Enables listing of symbols in the symbol portion of the symbol table and browser file.

.DATA
When used with **.MODEL**, starts a near data segment for initialized data (segment name _DATA).

.DATA?
When used with **.MODEL**, starts a near data segment for uninitialized data (segment name _BSS).

.DOSSEG
Orders the segments according to the MS-DOS segment convention: CODE first, then segments not in DGROUP, and then segments in DGROUP. The segments in DGROUP follow this order: segments not in BSS or STACK, then BSS segments, and finally STACK segments. Primarily used for ensuring CodeView support in MASM stand-alone programs. Same as **DOSSEG**.

DOSSEG
Identical to .**DOSSEG**, which is the preferred form.

DB
Can be used to define data like **BYTE**.

DD
Can be used to define data like **DWORD**.

DF
Can be used to define data like **FWORD**.

DQ
Can be used to define data like **QWORD**.

DT
Can be used to define data like **TBYTE**.

DW
Can be used to define data like **WORD**.

[[*name*]] **DWORD** *initializer* [[**,** *initializer*]]...
Allocates and optionally initializes a doubleword (4 bytes) of storage for each *initializer*. Can also be used as a type specifier anywhere a type is legal.

ECHO *message*
Displays *message* to the standard output device (by default, the screen). Same as **%OUT**.

.ELSE
See **.IF**.

ELSE
Marks the beginning of an alternate block within a conditional block. See **IF**.

ELSEIF
Combines **ELSE** and **IF** into one statement. See **IF**.

ELSEIF2
ELSEIF block evaluated on every assembly pass if **OPTION:SETIF2** is **TRUE**.

END [[*address*]]
Marks the end of a module and, optionally, sets the program entry point to *address*.

.ENDIF
See **.IF**.

ENDIF
See **IF**.

ENDM
Terminates a macro or repeat block. See **MACRO**, **FOR**, **FORC**, **REPEAT**, or **WHILE**.

name **ENDP**
Marks the end of procedure *name* previously begun with **PROC**. See **PROC**.

name **ENDS**
Marks the end of segment, structure, or union *name* previously begun with **SEGMENT**, **STRUCT**, **UNION**, or a simplified segment directive.

.ENDW
See **.WHILE**.

name **EQU** *expression*
Assigns numeric value of *expression* to *name*. The *name* cannot be redefined later.

name **EQU** <*text*>
Assigns specified *text* to *name*. The *name* can be assigned a different *text* later. See **TEXTEQU**.

.ERR [[*message*]]
Generates an error.

.ERR2 ⟦*message*⟧
.ERR block evaluated on every assembly pass if **OPTION:SETIF2** is **TRUE.**

.ERRB <*textitem*> ⟦**,** *message*⟧
Generates an error if *textitem* is blank.

.ERRDEF *name* ⟦**,** *message*⟧
Generates an error if *name* is a previously defined label, variable, or symbol.

.ERRDIF⟦**I**⟧ <*textitem1*>**,** <*textitem2*> ⟦**,** *message*⟧
Generates an error if the text items are different. If **I** is given, the comparison is case insensitive.

.ERRE *expression* ⟦**,** *message*⟧
Generates an error if *expression* is false (0).

.ERRIDN⟦**I**⟧ <*textitem1*>**,** <*textitem2*> ⟦**,** *message*⟧
Generates an error if the text items are identical. If **I** is given, the comparison is case insensitive.

.ERRNB <*textitem*> ⟦**,** *message*⟧
Generates an error if *textitem* is not blank.

.ERRNDEF *name* ⟦**,** *message*⟧
Generates an error if *name* has not been defined.

.ERRNZ *expression* ⟦**,** *message*⟧
Generates an error if *expression* is true (nonzero).

EVEN
Aligns the next variable or instruction on an even byte.

.EXIT ⟦*expression*⟧
Generates termination code. Returns optional *expression* to shell.

EXITM ⟦*textitem*⟧
Terminates expansion of the current repeat or macro block and begins assembly of the next statement outside the block. In a macro function, *textitem* is the value returned.

EXTERN ⟦*langtype*⟧ *name* ⟦**(***altid***)**⟧ **:***type* ⟦**,** ⟦*langtype*⟧ *name* ⟦**(***altid***)**⟧ **:***type*⟧...
Defines one or more external variables, labels, or symbols called *name* whose type is *type*. The *type* can be **ABS**, which imports *name* as a constant. Same as **EXTRN**.

EXTERNDEF ⟦*langtype*⟧ *name***:***type* ⟦**,** ⟦*langtype*⟧ *name***:***type*⟧...
Defines one or more external variables, labels, or symbols called *name* whose type is *type*. If *name* is defined in the module, it is treated as **PUBLIC**. If *name* is referenced in the module, it is treated as **EXTERN**. If *name* is not referenced, it is ignored. The *type* can be **ABS**, which imports *name* as a constant. Normally used in include files.

EXTRN
See **EXTERN**.

.FARDATA [[*name*]]
When used with **.MODEL**, starts a far data segment for initialized data (segment name FAR_DATA or *name*).

.FARDATA? [[*name*]]
When used with **.MODEL**, starts a far data segment for uninitialized data (segment name FAR_BSS or *name*).

FOR parameter [[**:REQ** | **:=**default]] **,** <argument [[**,** argument]]...>
statements
ENDM
Marks a block that will be repeated once for each *argument*, with the current *argument* replacing *parameter* on each repetition. Same as **IRP**.

FORC
*parameter***,** <*string*> *statements*
ENDM
Marks a block that will be repeated once for each character in *string*, with the current character replacing *parameter* on each repetition. Same as **IRPC**.

[[*name*]] **FWORD** *initializer* [[**,** *initializer*]]...
Allocates and optionally initializes 6 bytes of storage for each *initializer*. Also can be used as a type specifier anywhere a type is legal.

GOTO *macrolabel*
Transfers assembly to the line marked **:***macrolabel*. **GOTO** is permitted only inside **MACRO**, **FOR**, **FORC**, **REPEAT**, and **WHILE** blocks. The label must be the only directive on the line and must be preceded by a leading colon.

name **GROUP** *segment* [[**,** *segment*]]...
Add the specified *segments* to the group called *name*.

.IF *condition1*
statements
[[**.ELSEIF** condition2
statements]]
[[**.ELSE**
statements]]
.ENDIF
Generates code that tests *condition1* (for example, AX > 7) and executes the *statements* if that condition is true. If an **.ELSE** follows, its statements are executed if the original condition was false. Note that the conditions are evaluated at run time.

IF expression1
ifstatements
[[**ELSEIF** *expression2*
elseifstatements]]
[[**ELSE**
elsestatements]]
ENDIF

Grants assembly of *ifstatements* if *expression1* is true (nonzero) or *elseifstatements* if *expression1* is false (0) and *expression2* is true. The following directives may be substituted for **ELSEIF**: **ELSEIFB**,

ELSEIFDEF, **ELSEIFDIF**, **ELSEIFDIFI**, **ELSEIFE**, **ELSEIFIDN**, **ELSEIFIDNI**, **ELSEIFNB**, and **ELSEIFNDEF**. Optionally, assembles *elsestatements* if the previous expression is false. Note that the expressions are evaluated at assembly time.

IF2 *expression*
IF block is evaluated on every assembly pass if **OPTION:SETIF2** is **TRUE**. See **IF** for complete syntax.

IFB *textitem*
Grants assembly if *textitem* is blank. See **IF** for complete syntax.

IFDEF *name*
Grants assembly if *name* is a previously defined label, variable, or symbol. See **IF** for complete syntax.

IFDIF[**I**] *textitem1*, *textitem2*
Grants assembly if the text items are different. If **I** is given, the comparison is case insensitive. See **IF** for complete syntax.

IFE *expression*
Grants assembly if *expression* is false (0). See **IF** for complete syntax.

IFIDN[**I**] *textitem1*, *textitem2*
Grants assembly if the text items are identical. If **I** is given, the comparison is case insensitive. See **IF** for complete syntax.

IFNB *textitem*
Grants assembly if *textitem* is not blank. See **IF** for complete syntax.

IFNDEF *name*
Grants assembly if *name* has not been defined. See **IF** for complete syntax.

INCLUDE *filename*
Inserts source code from the source file given by *filename* into the current source file during assembly. The *filename* must be enclosed in angle brackets if it includes a backslash, semicolon, greater-than symbol, less-than symbol, single quotation mark, or double quotation mark.

INCLUDELIB *libraryname*
Informs the linker that the current module should be linked with *libraryname*. The *libraryname* must be enclosed in angle brackets if it includes a backslash, semicolon, greater-than symbol, less-than symbol, single quotation mark, or double quotation mark.

name **INSTR** [*position*,] *textitem1*, *textitem2*
Finds the first occurrence of *textitem2* in *textitem1*. The starting *position* is optional. Each text item can be a literal string, a constant preceded by a **%**, or the string returned by a macro function.

INVOKE *expression* [[**,** *arguments*]]
Calls the procedure at the address given by *expression*, passing the arguments on the stack or in registers according to the standard calling conventions of the language type. Each argument passed to the procedure may be an expression, a register pair, or an address expression (an expression preceded by **ADDR**).

IRP
See **FOR**.

IRPC
See **FORC**.

name **LABEL** *type*
Creates a new label by assigning the current location-counter value and the given *type* to *name*.

name **LABEL** [[**NEAR** | **FAR** | **PROC**]] **PTR** [[*type*]]
Creates a new label by assigning the current location-counter value and the given *type* to *name*.

.LALL
See **.LISTMACROALL**.

.LFCOND
See **.LISTIF**.

.LIST
Starts listing of statements. This is the default.

.LISTALL
Starts listing of all statements. Equivalent to the combination of **.LIST**, **.LISTIF**, and **.LISTMACROALL**.

.LISTIF
Starts listing of statements in false conditional blocks. Same as **.LFCOND**.

.LISTMACRO
Starts listing of macro expansion statements that generate code or data. This is the default. Same as **.XALL**.

.LISTMACROALL
Starts listing of all statements in macros. Same as **.LALL**.

LOCAL *localname* [[**,** *localname*]]...
Within a macro, **LOCAL** defines labels that are unique to each instance of the macro.

LOCAL *label* [[**[***count* **]**]][[**:***type*]] [[**,** *label* [[**[***count***]**]] [[*type*]]]]...
Within a procedure definition (**PROC**), **LOCAL** creates stack-based variables that exist for the duration of the procedure. The *label* may be a simple variable or an array containing *count* elements.

name **MACRO** [[*parameter* [[**:REQ** | **:=***default* | **:VARARG**]]]...
statements
ENDM [[*value*]]
Marks a macro block called *name* and establishes *parameter* placeholders for arguments passed when the macro is called. A macro function returns *value* to the calling statement.

.MODEL *memorymodel* [[**,** *langtype*]] [[**,** *stackoption*]]
Initializes the program memory model. The *memorymodel* can be **TINY**, **SMALL**, **COMPACT**, **MEDIUM**, **LARGE**, **HUGE**, or **FLAT**. The *langtype* can be **C**, **BASIC**, **FORTRAN**, **PASCAL**, **SYSCALL**, or **STDCALL**. The *stackoption* can be **NEARSTACK** or **FARSTACK**.

NAME *modulename*
Ignored.

.NO87
Disallows assembly of all floating-point instructions.

.NOCREF [[*name*[[**,** *name*]]...]]
Suppresses listing of symbols in the symbol table and browser file. If names are specified, only the given names are suppressed. Same as **.XCREF**.

.NOLIST
Suppresses program listing. Same as **.XLIST**.

.NOLISTIF
Suppresses listing of conditional blocks whose condition evaluates to false (0). This is the default. Same as **.SFCOND**.

.NOLISTMACRO
Suppresses listing of macro expansions. Same as **.SALL**.

OPTION *optionlist*
Enables and disables features of the assembler. Available options include **CASEMAP**, **DOTNAME**, **NODOTNAME**, **EMULATOR**, **NOEMULATOR**, **EPILOGUE**, **EXPR16**, **EXPR32**, **LANGUAGE**, **LJMP**, **NOLJMP**, **M510**, **NOM510**, **NOKEYWORD**, **NOSIGNEXTEND**, **OFFSET**, **OLDMACROS**, **NOOLDMACROS**, **OLDSTRUCTS**, **NOOLDSTRUCTS**, **PROC**, **PROLOGUE**, **READONLY**, **NOREADONLY**, **SCOPED**, **NOSCOPED**, **SEGMENT**, and **SETIF2**.

ORG *expression*
Sets the location counter to *expression*.

%OUT
See **ECHO**.

PAGE [[[[*length*]], *width*]]
Sets line *length* and character *width* of the program listing. If no arguments are given, generates a page break.

PAGE +
Increments the section number and resets the page number to 1.

POPCONTEXT *context*
Restores part or all of the current *context* (saved by the **PUSHCONTEXT** directive). The *context* can be **ASSUMES**, **RADIX**, **LISTING**, **CPU**, or **ALL**.

label **PROC** [[*distance*]] [[*langtype*]] [[*visibility*]] [[<*prologuearg*>]]
[[**USES** *reglist*]] [[**,** *parameter* [[**:***tag*]]]]...
statements
label **ENDP**
Marks start and end of a procedure block called *label*. The statements in the block can be called with the **CALL** instruction or **INVOKE** directive.

label **PROTO** [[*distance*]] [[*langtype*]] [[**,** [[*parameter*]]**:***tag*]]...
Prototypes a function.

PUBLIC [[*langtype*]] *name* [[**,** [[*langtype*]] *name*]]...
Makes each variable, label, or absolute symbol specified as *name* available to all other modules in the program.

PURGE *macroname* [[**,** *macroname*]]...
Deletes the specified macros from memory.

PUSHCONTEXT *context*
Saves part or all of the current *context*: segment register assumes, radix value, listing and cref flags, or processor/coprocessor values. The *context* can be **ASSUMES**, **RADIX**, **LISTING**, **CPU**, or **ALL**.

[[*name*]] **QWORD** *initializer* [[**,** *initializer*]]...
Allocates and optionally initializes 8 bytes of storage for each *initializer*. Also can be used as a type specifier anywhere a type is legal.

.RADIX *expression*
Sets the default radix, in the range 2 to 16, to the value of *expression*.

name **REAL4** *initializer* [[**,** *initializer*]]...
Allocates and optionally initializes a single-precision (4-byte) floating-point number for each *initializer*.

name **REAL8** *initializer* [[**,** *initializer*]]...
Allocates and optionally initializes a double-precision (8-byte) floating-point number for each *initializer*.

name **REAL10** *initializer* [[**,** *initializer*]]...
Allocates and optionally initializes a 10-byte floating-point number for each *initializer*.

recordname **RECORD** *fieldname*:*width* [[= *expression*]]
[[, *fieldname*:*width* [[= *expression*]]]]...
Declares a record type consisting of the specified fields. The *fieldname* names the field, *width* specifies the number of bits, and *expression* gives its initial value.

.REPEAT
statements
.UNTIL *condition*
Generates code that repeats execution of the block of *statements* until *condition* becomes true. **.UNTILCXZ**, which becomes true when CX is zero, may be substituted for **.UNTIL**. The *condition* is optional with **.UNTILCXZ**.

REPEAT *expression*
statements
ENDM
Marks a block that is to be repeated *expression* times. Same as **REPT**.

REPT
See **REPEAT**.

.SALL
See **.NOLISTMACRO**.

name **SBYTE** *initializer* [[, *initializer*]]...
Allocates and optionally initializes a signed byte of storage for each *initializer*. Can also be used as a type specifier anywhere a type is legal.

name **SDWORD** *initializer* [[, *initializer*]]...
Allocates and optionally initializes a signed doubleword (4 bytes) of storage for each *initializer*. Also can be used as a type specifier anywhere a type is legal.

name **SEGMENT** [[**READONLY**]] [[*align*]] [[*combine*]] [[*use*]] [['*class*']]
statements
name **ENDS**
Defines a program segment called *name* having segment attributes *align* (**BYTE**, **WORD**, **DWORD**, **PARA**, **PAGE**), *combine* (**PUBLIC**, **STACK**, **COMMON**, **MEMORY**, **AT** *address*, **PRIVATE**), *use* (**USE16**, **USE32**, **FLAT**), and *class*.

.SEQ
Orders segments sequentially (the default order).

.SFCOND
See **.NOLISTIF**.

name **SIZESTR** *textitem*
Finds the size of a text item.

.STACK [[*size*]]
When used with **.MODEL**, defines a stack segment (with segment name STACK). The optional *size* specifies the number of bytes for the stack (default 1,024). The **.STACK** directive automatically closes the stack statement.

.STARTUP
Generates program start-up code.

STRUC
See **STRUCT**.

name **STRUCT** [[*alignment*]] [[**, NONUNIQUE**]]
fielddeclarations
name **ENDS**
Declares a structure type having the specified *fielddeclarations*. Each field must be a valid data definition. Same as **STRUC**.

name **SUBSTR** *textitem*, *position* [[, *length*]]
Returns a substring of *textitem*, starting at *position*. The *textitem* can be a literal string, a constant preceded by a **%**, or the string returned by a macro function.

SUBTITLE *text*
Defines the listing subtitle. Same as **SUBTTL**.

SUBTTL
See **SUBTITLE**.

name **SWORD** *initializer* [[, *initializer*]]...
Allocates and optionally initializes a signed word (2 bytes) of storage for each *initializer*. Can also be used as a type specifier anywhere a type is legal.

[[*name*]] **TBYTE** *initializer* [[, *initializer*]]...
Allocates and optionally initializes 10 bytes of storage for each *initializer*. Can also be used as a type specifier anywhere a type is legal.

name **TEXTEQU** [[*textitem*]]
Assigns *textitem* to *name*. The *textitem* can be a literal string, a constant preceded by a **%**, or the string returned by a macro function.

.TFCOND
Toggles listing of false conditional blocks.

TITLE *text*
Defines the program listing title.

name **TYPEDEF** *type*
Defines a new type called *name*, which is equivalent to *type*.

name **UNION** [[*alignment*]] [[**, NONUNIQUE**]]
fielddeclarations
[[*name*]] **ENDS**
Declares a union of one or more data types. The *fielddeclarations* must be valid data definitions. Omit the **ENDS** *name* label on nested **UNION** definitions.

.UNTIL
See **.REPEAT**.

.UNTILCXZ
See **.REPEAT**.

.WHILE *condition*
statements
.ENDW
Generates code that executes the block of *statements* while *condition* remains true.

WHILE *expression*
statements
ENDM
Repeats assembly of block *statements* as long as *expression* remains true.

[[*name*]] **WORD** initializer [[**,** *initializer*]]...
Allocates and optionally initializes a word (2 bytes) of storage for each *initializer*. Can also be used as a type specifier anywhere a type is legal.

.XALL
See **.LISTMACRO**.

.XCREF
See **.NOCREF**.

.XLIST
See **.NOLIST**.