MPASM™ to MPLAB® XC8 PIC® Assembler Migration Guide

Notice to Customers

All documentation becomes dated and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions can differ from those in this document. Please refer to our web site (https://www.microchip.com) to obtain the latest documentation available.

Documents are identified with a "DS" number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is "DSXXXXXA," where "XXXXX" is the document number and "A" is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® IDE online help. Select the Help menu, and then Topics to open a list of available online help files.

Table of Contents

1.   Preface

1.1   Conventions Used in This Guide

The following conventions may appear in this documentation:

Table 1-1. Documentation Conventions

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	*MPLAB® IDE User's Guide*
	Emphasized text	...is the *only* compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	*File>Save*
Bold characters	A dialog button	Click **OK**
	A tab	Click the **Power** tab
N'Rnnnn	A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.	4'b0010, 2'hF1
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier New font:		
Plain Courier New	Sample source code	`#define START`
	Filenames	`autoexec.bat`
	File paths	`c:\mcc18\h`
	Keywords	`_asm, _endasm, static`
	Command-line options	`-Opa+, -Opa-`
	Bit values	`0, 1`
	Constants	`0xFF, 'A'`
Italic Courier New	A variable argument	`file.o`, where `file` can be any valid filename
Square brackets [ ]	Optional arguments	`mcc18 [options] file [options]`
Curly brackets and pipe character: {	}	Choice of mutually exclusive arguments; an OR selection	`errorlevel {0|1}`
Ellipses...	Replaces repeated text	`var_name [, var_name...]`

	Represents code supplied by user	`void main (void)`
		`{ ...`
		`}`

1.2   Recommended Reading

This guide is for customers who have MPASM projects and who wish to migrate them to the MPLAB XC8 PIC assembler. The following Microchip documents are available and recommended as supplemental reference resources.

MPLAB® XC8 PIC Assembler User's Guide
This user's guide describes the use and features of the MPLAB XC8 PIC Assembler.

MPLAB® XC8 PIC Assembler Guide For Embedded Engineers
This guide is a getting started guide, describing example projects and commonly used coding sequences used by the MPLAB XC8 PIC assembler. Use this guide if you need to develop new projects using the assembler.

MPLAB® XC8 C Compiler Release Notes for PIC MCU
For the latest information on changes and bug fixes to this assembler, read the Readme file in the docs subdirectory of the MPLAB XC8 installation directory.

Development Tools Release Notes
For the latest information on using other development tools, read the tool-specific Readme files in the docs subdirectory of the MPLAB X IDE installation directory.

1.3   Document Revision History

Revision A (March 2020)
*   Initial release of this document.

2. Introduction

This guide highlights the source code and build changes that might be required should you decided to migrate a project from the Microchip MPASM™ assembler (MPASM) to the MPLAB® XC8 PIC Assembler (PIC Assembler).

The majority of the changes required when migrating a project will be to assembler directives. Instruction sequences typically do not require alteration, although some instruction operand expressions might require the use of different operators or syntax.

A migrated project will almost certainly produce a hex file that differs to one produced from the original MPASM project built with the MPASM assembler. When building with the PIC Assembler, objects and code might be linked at different addresses, meaning that different address operands will be used with instructions, and different bank and page selection sequences might be present. Instruction sequences, however, should remain unchanged, as no optimizations or code transformations are performed by the PIC Assembler.

You can use the PIC Assembler from within the MPLAB X IDE. Projects dedicated to this tool can be created and will use their own set of options displayed in the project's properties.

See the *MPLAB® XC8 PIC Assembler User's Guide* for full information on how to use the assembler and for more detailed information on the assembler's directives and language. A separate *MPLAB® XC8 PIC Assembler Guide for Embedded Engineers* document contains code and build option examples and getting started information.

2.1 File Types

The source file extensions used by the PIC Assembler differ to those used by MPASM.

Use a `.s` extension (lower case) for assembly source files. Use `.S` (upper case) for assembly source files that must be preprocessed before being passed to the assembler, or alternatively, use the `-xassembler-with-cpp` option to request that source files be preprocessed regardless of their extension.

A list of common file extension equivalents is shown in the table below.

Table 2-1. Equivalent File Extensions

MPASM File Extension	File type	PIC Assembler Equivalent
`.asm`	Assembly source file	`.s` or `.S`
`.inc`	Include (header) file	`.inc`
`.hex`	HEX file output	`.hex`
`.o`	Object file	`.o`
`.lib`	Library archive	`.a`

As output, the PIC Assembler will produce a `.hex` and `.elf` file, each with the same basename as the first source file listed on the command line. If you are building from within the MPLAB X IDE, the files' basename will be the project name. Note that Hexmate, which is often run by the driver, can create a log file with a `.hxl` extension. Do not confuse this file with MPASM split hex files, which also use this extension.

Object files are the intermediate file format used by the assembler, but note that the format of MPASM and PIC Assembler object files are very different. The PIC Assembler cannot read object files created by MPASM, so these cannot be included in PIC Assembler projects. Instead, include into the project the migrated original source code from which the MPASM object files were built.

Library archives created with the archiver, `xc8-ar`, should use a `.a` extension. Library archives may be passed to the assembler on the command line. Although these archives may contain any mix of p-code or object modules, only the object modules will be searched by the assembler. Note that the format of MPASM and PIC Assembler library archives are very different. The PIC Assembler cannot read library files created by MPASM, so these cannot be included in PIC Assembler projects. Instead, include into the project the migrated original source code from which the MPASM libraries were built.

The PIC Assembler does not support the generation of cross reference files. The MPASM assembler produces these files (`.xrf` extension), which can be used to obtain listings of program symbols. When using the PIC Assembler, all the symbols used by a module are shown in the assembly list file associated with that module. All program-wide global symbols are also shown in the map file.

2.2   Command-line Options

Some of the command-line options used with MPASM have equivalents in the PIC Assembler.

All PIC Assembler options begin with either a dash, `-`, or double dash, `--`. The options are case sensitive.

The MPASM command-line options and their PIC Assembler equivalents are shown in the table below.

Table 2-2. Equivalent Command-line Options

MPASM Option	Purpose	PIC Assembler Equivalent
`?` or `h`	Display help	`--help`
`ahex-format`	Specify hex file output and format	Use `-g` to specify the type of hex file; code is always relocatable
`c`	Control case sensitivity	No equivalent
`dlabel[=value]`	Define textual substitution	The preprocessor feature controlled by `-Dmacro=text` is similar
`e`	Specify error file	No equivalent
`l` or `l+`	Enable listing file	`-Wa,-a`
`l path`	Specify listing file path	No equivalent
`m`	Enable macro expansion	No equivalent
`o`	Enable object file and specify path	No equivalent
`pdevice`	Specify target device	`-mcpu=device`
`q`	Quiet mode	No equivalent
`rradix`	Specify default radix	No equivalent
`s`	Show progress window	No equivalent
`t`	Set tab size	No equivalent
`wvalue`	Specify messages output	`-w`, see also `-mwarn`.
`x`	Enable cross reference file and specify path	No equivalent
`y`	Specify PIC18 instruction set	`-misa=[std|xinst]`

2.3   Relocatable Code

The MPLAB XC8 PIC Assembler package includes a linker, `hlink`, and always builds code in a manner that is referred to by MPASM documentation as relocatable code. That is to say that the PIC Assembler produces relocatable object modules that can be linked with other modules or archive libraries to form the final program image.

Typically, you always run the PIC Assembler through the entire build sequence, so that a final image of your project is produced, but you can, if required, stop the build process before the link step by using the `-c` option. This results in an object file output, which must later be linked (together with other object modules or library archives, if required) to form the final program image.

3. Expressions and Operators

The operands of instructions and assembler directives are represented and evaluated differently in the PIC Assembler compared with MPASM.

3.1 Constants and Radices

The default radix for constants in the PIC Assembler is different to that used by MPASM, and the radix specifiers used by each assembler are different.

Without any radix specifier or directive, numeric constants in the PIC Assembler are interpreted as decimal values. Such values would be interpreted as hexadecimal in MPASM. Use the `RADIX hex` directive in migrated assembly code to ensure that the default radix assumed by the PIC Assembler matches that used by MPASM.

The MPASM radix specifiers and the equivalent specifiers used by the PIC Assembler are tabulated below.

Table 3-1. Equivalent Constants Radix Specifiers

MPASM Constant Forms	Radix	PIC Assembler Equivalent
`B'`*`binary_digits`*`'`	Binary	*`binary_digits`*`B`
`O'`*`octal_digits`*`'`	Octal	*`octal_digits`*`[O|o|Q|q]`
`D'`*`decimal_digits`*`'` or `.`*`decimal_digits`*	Decimal	*`decimal_digits`*`[D|d|`nothing`]`
`H'`*`hexadecimal_digits`*`'` or `0x`*`hexadecimal_digits`*	Hexadecimal	`0`*`hexadecimal_digits`*`[H|h]` or `0x`*`hexadecimal_digits`*
`A'`*`character`*`'` or `'`*`character`*`'`	ASCII	`'`*`character`*`'`

Note that the binary digits suffix (`B`) used by the PIC Assembler must be in upper case to avoid confusion with temporary labels. Hexadecimal values must always begin with a zero, `0`.

The following example shows the PIC Assembler's radix specifiers in use.

```
movlw 10110011B  ;binary value
movlw 72q        ;octal value
movlw 34         ;decimal value
movlw 04Fh       ;hexadecimal value
movlw 'b'        ;ASCII value
```

3.2 Labels

The PIC Assembler is more strict than MPASM regarding the definition of labels.

A label is a symbolic alias that is assigned a value equal to the current location within the current psect. This assignment is typically performed by the linker.

In the PIC Assembler, a label definition consists of any valid assembly identifier followed by a colon, `:`. In MPASM, the colon is option. When migrating, you must add colons to any label in MPASM code that does not already use one.

Label identifiers used by the PIC Assembler are always case sensitive.

A label identifier can contain any number of characters drawn from the alphabetics, numerics, and the special characters: dollar, `$`; question mark, `?`; and underscore, `_`. The first character of an identifier cannot be numeric. A identifier cannot have the same name as any of the assembler directives, keywords, or psect flags. A label definition can appear on a line by itself or it can be positioned to the left of an instruction or assembler directive.

The following shows the definition of valid and unique labels in the PIC Assembler.

```
An_identifier:
  movlw 55
```

```
an_identifier:  movlw  0AAh
an_identifier1:  DW 0x1234
?$_12345:
  return
```

3.3   File Register Address Masking

It is recommended that you mask all addresses used by PIC file register instructions whose operand represents a bank offset.

The value of a label (for example the label of an object in RAM) is always the full address of where the label was positioned in memory. Most PIC instructions that operate on a file register require only the offset within a bank to be specified as the file register operand. The upper bits of the address, which indicate the bank number, must be removed. The PIC Assembler will issue a fixup overflow error (for symbolic operands) or warning (for absolute operands) should it detect that there is superfluous bank information in a file register address operand.

There are two common ways that you can mask the address operand.

- Mask out the bank information from the address using a bitwise AND (`&` operator), typically using the `BANKMASK()` macro.
- Mask out the bank information from the address using a bitwise XOR (`^` operator).

The mask value used by both these methods will vary between device, as the device families have different sized banks.

Shown below are the compositions of an address for each device family. They all consist of a bank value, `b`, and an offset into that bank, `o`. As indicated, a Baseline device, which has data banks 0x20 bytes in size, has an address comprised of 5 bits of bank offset and the bank value consuming the remaining upper bits. (All the bits of the bank information shown might not be implemented by every device). So, for example, the Baseline address 0x65 (binary bit pattern 011**0 0101**) is the address of the byte residing at bank offset **5** in bank 3. By comparison, a PIC18 device has 8 bits of bank offset and the remainder being the bank value.

```
Address decomposition (b: bank value, o: bank offset)
bit position:    15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
Baseline:         b  b  b  b  b  b  b  b  b  b  b  o  o  o  o  o
Mid-range:        b  b  b  b  b  b  b  b  b  o  o  o  o  o  o  o
PIC18:            b  b  b  b  b  b  b  b  o  o  o  o  o  o  o  o
```

To remove the bank value from an address using an AND, use a mask with a 1 in each bit position that corresponds to the bank offset, zeros elsewhere. For example on PIC18 devices, the mask would be 0xFF; on Mid-range devices, this would be 0x7F; and Baseline, 0x1F. The `BANKMASK()` macro, available once you include `<xc.inc>`, can perform the mask using the correct values, based on the selected device. Its use is shown in the following example.

```
#include <xc.inc>

copy:
  BANKSEL (src)              ;select the bank of src
  movf      BANKMASK(src),w  ;move from src, masking the address
  BANKSEL (dst)              ;select the bank of dst
  movwf     BANKMASK(dst)    ;move to dst, masking the address
```

Alternatively, using an XOR operation to mask the address gives you the opportunity to perform additional checks that ensure your assumptions about the location of objects are correct. You should XOR the address with a value that will clear the *expected* bank value but leave the bank offset unchanged. Such a value will have zeros in the bank offset locations and specify the bit pattern of the bank in which the object should be located as the bank value. So, for example, XOR the address operand with the mask 0x100 on PIC18 devices if the operand is assumed to be an object in bank 1; XOR with 0x300 if it meant to be an operand in bank 3. On Mid-range devices, XOR with 0x80 for bank 1 objects; 0x180 for bank 3 objects. On Baseline devices, XOR with 0x20 for bank 1 objects; 0x60 for bank 3 objects etc. In the following Mid-range example, an error will produced if `src` is not in bank 1 or if `dst` is not in bank 2.

```
#include <xc.inc>

copy:
  BANKSEL (src)              ;select the bank of src
```

```
    movf    src^080h,w          ;move from src (in bank 1)
    BANKSEL (dst)               ;select the bank of dst
    movwf   dst^0100h           ;move to dst (in bank 2)
```

You can also XOR the address operand with a symbolic value. In the following Mid-range example, the programmer has now assumed that `scr` and `dst` used in the previous example are in the same bank, and so only one bank selection sequence is present to reduce the code size and speed up execution. An XOR can be performed using these two symbols to ensure that this assumption is valid. In the last instruction, the expression `src&0FF80h` obtains the bank value for `src`. This value is then XORed with the full address of `dst`.

```
 #include <xc.inc>

 copy:
   BANKSEL (src)               ;select the bank of src
   movf    BANKMASK(src),w     ;move from src, masking the address
   movwf   dst^(src&0FF80h)    ;move to dst, which should be in the same bank as src
```

If `scr` and `dst` in the above code are subsequently linked in the same bank, then the XOR of the bits representing their bank values will be 0, effectively masking the bank value from the operand address to the `movwf` instruction. If `src` and `dst` are instead linked into different banks, the XOR will yeild non-zero bits in bank value that will trigger an error, thus confuting the programmer's assumption and preventing the code from failing at runtime. This is a case where it does not matter what bank these objects are in, but they must both be in the same bank for the code to execute correctly.

You must not mask any address used as the operand to an instruction that accepts a full address, such as the `movff` or `movffl` instruction. Address making is not required if the object is in bank 0 (since its bank value will already be 0); however, it is good practice to mask all address operands.

3.4   Operators

The operators defined by the PIC Assembler can be used in most expressions. Unlike those in MPASM, the PIC Assembler operators have no operator precedence and they all have left-to-right associativity. You may need to use parentheses when migrating complex expressions to ensure they are evaluated as they would have been in MPASM.

The MPASM operators and their PIC Assembler equivalents are tabulated below.

Table 3-2. Equivalent Operators

MPASM Operator	Purpose	PIC Assembler Equivalent
$	Current/Return program counter	$
(	Left parenthesis	(
)	Right parenthesis	)
!	Logical compliment (NOT)	No equivalent
–	Negation (2's compliment)	–
~	Complement	not
low	Low address byte	low
high	High address byte	high
upper	Upper address byte	low highword
*	Multiplication	*
/	Division	/
%	Modulus	%
+	Addition	+
–	Subtraction	–

MPASM Operator	Purpose	PIC Assembler Equivalent
..........continued		
`<<`	Left shift	`<<` or `shl`
`>>`	Right shift	`>>` or `shr`
`>=`	Greater than or equal	`>=` or `ge`
`>`	Greater than	`>` or `gt`
`<`	Less than	`<` or `lt`
`<=`	Less than or equal	`<=` or `le`
`==`	Equality	`=` or `eq`
`!=`	Inequality	`<>` or `ne`
`&`	Bitwise AND	`&` or `and`
`^`	Bitwise XOR	`^`
`|`	Bitwise OR	`|`
`&&`	Logical AND	No equivalent
`||`	Logical OR	No equivalent
`=`	Assignment	No equivalent
`+=`	Assignment addition	No equivalent
`-=`	Assignment subtraction	No equivalent
`*=`	Assignment multiplication	No equivalent
`/=`	Assignment division	No equivalent
`%=`	Assignment modulus	No equivalent
`<<=`	Assignment left shift	No equivalent
`>>=`	Assignment right shift	No equivalent
`&=`	Assignment logical AND	No equivalent
`|=`	Assignment logical OR	No equivalent
`^=`	Assignment logical XOR	No equivalent
`++`	Increment	No equivalent
`--`	Decrement	No equivalent

4.   Assembler Directives

Some MPASM assembler directives have equivalents in the MPLAB XC8 PIC Assembler; however, other MPASM directives must be adapted to a new syntax or replaced with an alternate sequence of directives.

The following table shows each MPASM directive and the best PIC Assembler equivalent. These directives and their equivalents are discussed in more detail in the sections that follow. Note that there can be subtle differences in the behavior of what appear to be identical directives, and this could lead to unexpected program behavior. It is recommended that you compare the directive description in the *MPLAB® XC8 PIC Assembler User's Guide* with that in your MPASM documentation to ensure that your migrated code will work as expected in all situations.

Table 4-1. Equivalent Directives

MPASM Directive	PIC Assembler Replacement
ACCESS_OVR	A psect with the ovrld flag set
__BADRAM and __BADROM	No replacement
BANSISEL	The instruction sequence to write indirect access registers
BANKSEL	The BANKSEL directive (no change required)
CBLOCK	Consider the SET/EQU directives, or DS.
CODE	The code psect or similar
CODE_PACK	The code psect or similar
__CONFIG	The CONFIG directive with appropriate settings and values
CONFIG	The CONFIG directive with appropriate settings and values
CONSTANT	The EQU directive
DA	Consider the DB or IRPC directives
DATA	Consider the DW directive
DB	The DB directive
DE	Consider the DB directive inside a suitable psect
#DEFINE	The #define preprocessor directive
DT	The IRP directive
DTM	The IRP directive
DW	The DW or DB directive
ELSE	The ELSE directive (no change required)
END	The END directive (no change required)
ENDC	No replacement
ENDM	The ENDM directive (no change required)
ENDW	No replacement
EQU	The EQU directive (no change required)
ERROR	The ERROR directive

..........continued	
MPASM Directive	**PIC Assembler Replacement**
ERRORLEVEL	Consider the -w driver option
EXITM	No replacement
EXPAND	The EXPAND directive (no change required)
EXTERN	The EXTRN directive (note different spelling)
FILL	Consider the --fill driver option
GLOBAL	The GLOBAL directive (no change required)
IDATA	A psect with the initial values in program memory and another reserving space for the data objects
IDATA_ACS	A psect with the initial values in program memory and another reserving space for the data objects
IF	The IF directive (no change required)
IFDEF	Consider the #ifdef preprocessor directive
IFNDEF	Consider the #ifndef preprocessor directive
#INCLUDE	The #include preprocessor directive
LIST	The LIST directive or consider alternate assembler options
LOCAL	The LOCAL directive (no change required)
MACRO	The MACRO directive (no change required)
__MAXRAM and __MAXROM	No replacement
MESSG	The MESSG directive (no change required)
NOEXPAND	The NOEXPAND directive (no change required)
NOLIST	The NOLIST directive (no change required)
ORG	Consider the ORG directive
PAGE	No replacement
PAGESEL	The PAGESEL directive (no change required)
PAGESELW	Consider the PAGESEL directive
PROCESSOR	The PROCESSOR directive (no change required)
RADIX	The RADIX directive (no change required)
RES	Consider the DS directive
SET	The SET directive (no change required)
SPACE	The SPACE directive (no change required)
SUBTITLE	The SUBTITLE directive (no change required)
TITLE	The TITLE directive (no change required)
UDATA	The udata_bank*n* psect or similar
UDATA_ACS	The udata_acs psect or similar

..........continued	
MPASM Directive	**PIC Assembler Replacement**
UDATA_OVR	A psect with the `ovrld` flag set
UDATA_SHR	A psect with the `ovrld` flag set
#UNDEFINE	The `#undefine` preprocessor directive
VARIABLE	Consider the `SET` directive
WHILE	Consider the `REPT` directive

4.1 Access_ovr Directive

On PIC18 devices, the MPASM `ACCESS_OVR` directive declares the beginning of a section of overlaid data in Access Bank RAM.

Suggested Replacement

Define a psect with the `ovrld` flag set. Associate the psect with the access bank linker class, `COMRAM` to have it linked somewhere in the PIC18's Access Bank.

The example following shows objects placed into the `myData` psect, which is used in two different modules (file 1 and file 2). The `PSECT` directive uses the same psect name, `ovrld` flag (to indicate that the psects will be overlaid), and `space=1` flag (to indicate the psects will reside in the data space memory). The psects are associated with the PIC18 `COMRAM` linker class, which defines the Access Bank memory, so once overlaid, `myData` will appear anywhere in the memory defined by this class.

```
;file 1
PSECT myData,space=1,ovrld,class=COMRAM
zero:
  DS 1
  ;leave a 1-byte gap for another object here
  ORG 2
two:
  DS 1

;file 2
PSECT myData,space=1,ovrld,class=COMRAM
  ORG 1
one:
  DS 1
```

Note that the contributions to an overlaid psect are concatenated in each module, but the psects from each module are then overlaid at link time. When the above example is built, the labels will appear in memory in the order `zero`, `one`, `two`.

The `ORG` directive in the above example has allowed the psects' content to interleave. If this directive had not been used in file 1, the space associated with the label `zero` and the label `one` would overlap, and these objects would appear at the same address. Such code is legal and may be desired in some applications.

Overlaid psects can be linked into any RAM area, not just the Access bank, and this construct will work on any device with the selection of a suitable linker class.

4.2 Badram and Badrom Directives

The MPASM `__BADRAM` and `__BADROM` directives along with the `__MAXRAM` and `__MAXROM` directives specifies file register address ranges that should be flagged as being invalid should they be used by code.

Suggested Replacement

There is no replacement for these directives.

Using the `-mreserve` driver option will restrict linker classes to the desired memory ranges. If you only use symbols defined in psects placed into those classes, invalid memory areas will be avoided.

The `_RAMSIZE` and `_ROMSIZE` preprocessor macros are also available and indicate the largest data and program memory space address available with the selected target device.

```
#define DEST 0x7D0
PSECT text,CLASS=code,reloc=2  ;for PIC18 devices
;for other devices: PSECT text,class=CODE,delta=2
storeIt:
  movlw 66
#if DEST > _ROMSIZE
#error "destination out of bounds"
#endif
  BANKSEL (DEST)
  movwf BANKMASK(DEST)
  return
```

4.3 Bankisel Directive

On Mid-range devices only, the MPASM `BANKISEL` directive generates bank selection code appropriate for indirect access of the register address specified by its argument.

Suggested Replacement
There is no equivalent directive in the PIC Assembler.

Replace this directive with instructions that setup the registers associated with indirect access as required for the code following. Consult your device data sheet for the registers involved with indirect access.

4.4 Banksel Directive

The MPASM `BANKSEL` directive generates bank selection code appropriate for the label specified as the argument.

Suggested Replacement
No change is necessary; continue to use the `BANKSEL` assembler directive.

This directive is case-insensitive, can be used with any device, and works with numeric or symbolic operands. Note that it may generate more than one instruction, so should not be used immediately following any test-and-skip instruction.

```
  movlw 66
  BANKSEL input
  movwf BANKMASK(input)
```

4.5 Cblock Directive

The MPASM `CBLOCK` directive generates group of labels with sequential addresses.

Suggested Replacement
There is no equivalent directive in the PIC Assembler.

Explicitly define the required symbols using the `SET` or `EQU` directives. If there is to be storage associated with each label, instead define labels followed by a `DS` directive in a suitable psect .

4.6 Code Directive

The MPASM `CODE` directive starts a section containing program code.

Suggested Replacement

Use the predefined `code` psect, or a create a similar psect, ensuring the flags are suitable for a section containing executable code on the target device. If the psects must be padded to a multiple of a certain size, use the `ALIGN` directive after the code you place inside the psect.

All program code must be placed in a psect (or section) using the `PSECT` directive. The PIC Assembler provides the `code` psect once you include `<xc.inc>`. You can use this psect without having to specify any psect flags, for example:

```
PSECT code
;place code for any device here
```

Alternatively, you can define your own psect with any name and suitable psect flags. The psect's `space` flag must be 0, to indicate that the psect should be positioned in program memory; however, this is the default value. Typically, you would use the `class` flag to assign the psect to the `CODE` linker class, which is also predefined by the driver, so that the psect will be positioned somewhere in the memory associated with this class without you having to specify any linker options. You could also position this psect at a particular address using the linker's `-p` option, passed to the linker from the driver's `-Wl` option.

For PIC18 devices, set the psect's `reloc` flag to be 2, to indicate that the psect contents must be aligned on even addresses. Use the default `delta` value of 1. For example:

```
PSECT myText,reloc=2,class=CODE
;PIC18 code goes here
```

For all other devices, use the default `reloc` flag value of 1, but set the `delta` value to be 2, to indicate that the device uses (16-bit) word addressable program memory. For example:

```
PSECT myText,delta=2,class=CODE
;Baseline/Mid-range code goes here
```

You can add more content to the same psect later in the source file by using the `PSECT` directive with the psect's name, but you do not need to repeat the psect flags. You could, for example, concatenate more content to the psects defined above by using:

```
PSECT myText
;more content goes here
```

4.7   Code_pack Directive

The MPASM `CODE_PACK` directive starts a section containing program code that will not be padded to an even length.

Suggested Replacement

Use the predefined `code` psect, or a create a similar psect, ensuring the flags are suitable for a section containing executable code on the target device.

See 4.6 Code Directive for more information and examples.

4.8   __config Directive

The MPASM `__CONFIG` directive sets the device's configuration bits.

Suggested Replacement

Use the `CONFIG` directive and setting-value pairs relevant to your target device and application. See 4.9 Config Directive.

4.9   Config Directive

The MPASM `CONFIG` directive sets the device's configuration bits.

Suggested Replacement

The PIC Assembler's `CONFIG` directive is a direct replacement, but ensure that the settings and values used with the directive are correct for your device.

This directive can be used for all devices. The following example shows the different ways it can be used to program individual configuration bits, a configuration word as a whole, or ID location values.

```
; PIC18F67K22
; VREG Sleep Enable bit : Enabled
; LF-INTOSC Low-power Enable bit : LF-INTOSC in High-power mode during Sleep
; SOSC Power Selection and mode Configuration bits : High Power SOSC circuit selected
; Extended Instruction Set : Enabled
config RETEN = ON, INTOSCSEL = HIGH, SOSCSEL = HIGH, XINST = ON

; Alternatively, set the entire word
config CONFIG1L = 0x5D

; IDLOC @ 0x200000
config IDLOC0 = 0x15
```

See the *MPLAB® XC8 PIC Assembler User's Guide* for full information on this directive.

4.10   Constant Directive

The MPASM `CONSTANT` directive declares a symbol whose initial value cannot be changed.

Suggested Replacement

Use the PIC Assembler's `EQU` directive, for example:

```
threshold EQU 0xFF
```

4.11   Da Directive

The MPASM `DA` directive packs each sequence of two ASCII characters within a string into a 14-bit word in program memory.

Suggested Replacement

The PIC Assembler's `DB` directive inside a suitable psect performs a similar function, but note that packed strings are not supported by the PIC assembler and that each character will appear as a full byte in memory.

Consider placing the `IRPC` and/or `DB` directives in the `data` psect that is provided once you include `<xc.inc>`, for example:

```
#include <xc.inc>

PSECT data
myString:
  IRPC char,ABC
    DB   'char'
  ENDM
```

but note that each character will consume one entire byte of memory.

Alternatively, you can define your own psect and allocate it to program memory. Ensure that the psect's `space` flag is set to 0 (the default value). It can be assigned an address by associating it with a suitable linker class (e.g. `CONST` for

PIC18 devices, or `STRCODE` for other devices), or by explicitly positioning the psect using the linker's `-P` option (accessible from the driver using the `-Wl` option), as in the following PIC18 example.

```
PSECT romData,space=0,class=CONST
myString:
    DB   'A', 'B', 'C'
```

4.12   Data Directive

The MPASM `DATA` directive initializes one or more words of program memory with data.

Suggested Replacement

There is no direct replacement for this directive; however, the PIC Assembler's `DW` directive performs a similar function. See 4.18 Dw Directive for more information and examples.

4.13   Db Directive

The MPASM `DB` directive places bytes into program memory.

Suggested Replacement

The PIC Assembler's `DB` directive inside a suitable psect performs a similar function, but there are some differences in its operation.

This directive places the value of its operands as bytes into the current psect. String literal operands are not supported, but as shown in the example below, you may use characters and multiple comma-separated operands.

With PIC18 devices, each byte specified will consume one byte pf program memory. With Mid-range devices, there will be one byte stored in each program word, with the upper bits of that word left as zeros. To have data encapsulated into `retlw` instructions, use `retlw` instructions instead of this directive (see the example in the 4.16 Dt Directive).

You can use the `data` psect to hold the values defined. This psect is predefined once you include `<xc.inc>`. For example:

```
PSECT data
symbols:
  DB 76h
  DB 'A', -23
```

Alternatively, you can define your own psect and allocate it to program memory. Ensure that the psect's `space` flag is set to 0 (the default value). It can be assigned an address by associating it with a suitable linker class (e.g. `CONST` for PIC18 devices, or `STRCODE` for other devices), or by explicitly positioning the psect using the linker's `-P` option (accessible from the driver using the `-Wl` option), as in the following PIC18 example.

```
PSECT romData,space=0,class=CONST
symbols:
  DB 76h
  DB 'A', -23
```

4.14   De Directive

The MPASM `DE` directive places words into EEPROM.

Suggested Replacement

The PIC Assembler's `DW` directive inside a suitable psect performs a similar function.

You can use the `edata` psect to hold the values defined. This psect is predefined once you include `<xc.inc>`. For example:

```
PSECT edata
  DW 9700h
  DW 'r', -48
```

Alternatively, you can define your own psect and allocate it to EEPROM. Ensure that the psect's `space` flag is set to 0 (the default value) for PIC18 devices, or set to 3 for any other devices that support EEPROM. The psect can be assigned an address by associating it with the EEDATA linker class, or by explicitly positioning the psect using the linker's `-P` option (accessible from the driver using the `-Wl` option), as in the following Mid-range example.

```
PSECT eepromData,space=3,class=CONST
symbols:
  DW 76h
  DW 'A', -23
```

4.15 #define Directive

The MPASM `#DEFINE` directive defines textual replacement for a macro.

Suggested Replacement

As assembly source files can be preprocessed, the `#define` preprocessor directive (case sensitive) can be used as a direct replacement for this assembler directive, as shown in the example below.

All the usual preprocessor features associated with macro replacement are available. Ensure the assembly source file uses a `.S` extension so that it will be preprocessed by the assembler.

```
#define SUMSP(a, b) (a+2*b)
PSECT code
process:
  movlw SUMSP(5, 3)
  movwf volume
  ...
```

4.16 Dt Directive

The MPASM `DT` directive creates a table of `retlw` instructions.

Suggested Replacement

The PIC Assembler's `IRP` directive inside a suitable psect performs a similar task.

The `IRP` directive works like a macro. The block of code terminated by an `ENDM` token is output once for each value specified after the argument name. Any occurrence of the name argument inside the definition is replaced with the value being processed.

You can use the `data` psect to hold the values defined. This psect is predefined once you include `<xc.inc>`. For example:

```
PSECT data
symbols:
IRP number,48,65h,6Fh
    retlw number
ENDM
```

This would expand to:

```
PSECT data
symbols:
  retlw 48
```

```
   retlw 65h
   retlw 6Fh
```

Alternatively, you can define your own psect and allocate it to program memory. Ensure that the psect's `space` flag is set to 0 (the default value). It can be assigned an address by associating it with a suitable linker class (e.g. `CONST` for PIC18 devices, or `STRCODE` for other devices), or by explicitly positioning the psect using the linker's `-P` option (accessible from the driver using the `-Wl` option), as in the following PIC18 example.

```
PSECT romData,space=0,class=CONST
symbols:
IRP number,48,65h,6Fh
    retlw number
ENDM
```

4.17   Dtm Directive

The MPASM `DTM` directive creates a table of `movlw` instructions.

Suggested Replacement

The PIC Assembler's `IRP` directive inside a suitable psect performs a similar task.

See 4.16 Dt Directive for more information and examples.

4.18   Dw Directive

The MPASM `DW` directive places words into program memory.

Suggested Replacement

The PIC Assembler's `DW` directive inside a suitable psect performs a similar function, but there are some differences in its operation.

This directive places the value of its operands as 16-bit words into the current psect. For PIC18 devices, each operand will consume two addresses (bytes). For other devices, the assembler will attempt to place the entire operand value into one program memory word, but the program memory of these devices are less than 2 bytes wide, so the value may be truncated. Typically, data is stored in the program memory of these devices using `retlw` instructions that store each byte in separate memory locations.

String literal operands are not supported with this directive, but as shown in the example below, you may use characters and multiple comma-separated operands.

You can use the `data` psect to hold the values defined. This psect is predefined once you include `<xc.inc>`. For example:

```
PSECT data
modifiers:
  DW 1354h
  DW 's', -23
```

Alternatively, you can define your own psect and allocate it to program memory. Ensure that the psect's `space` flag is set to 0 (the default value). It can be assigned an address by associating it with a suitable linker class (e.g. `CONST` for PIC18 devices, or `STRCODE` for other devices), or by explicitly positioning the psect using the linker's `-P` option (accessible from the driver using the `-Wl` option), as in the following PIC18 example.

```
PSECT myData,space=0,class=CONST
modifiers:
  DW 1354h
  DW 's', -23
```

4.19   Else Directive

The MPASM `ELSE` directive provides an alternative block of assembly code to build should the `IF` directive condition evaluate to false.

Suggested Replacement

The PIC Assembler's `ELSE` directive is a direct replacement for this directive.

See 4.34 If Directive for examples.

4.20   End Directive

The MPASM `END` directive indicates the end of the program's source code.

Suggested Replacement

The PIC Assembler's `END` directive performs the same task.

Use of the `END` directive is optional. Once encountered, the assembler will assume there are no more lines of input, and even blank lines after an `END` directive will trigger an error.

The program's start label should be specified as an argument to one of these directives to prevent an assembler warning.

```
 ...
 return
 END  startMain ;the end of the program, folks
```

4.21   Endc Directive

The MPASM `ENDC` directive terminates a `CBLOCK` directive.

Suggested Replacement

There is no direct replacement for this directive.

4.22   Endm Directive

The MPASM `ENDM` directive terminates a macro definition.

Suggested Replacement

The PIC Assembler's `ENDM` directive is direct replacement.

See 4.40 Macro Directive for examples.

4.23   Endw Directive

The MPASM `ENDW` directive terminates a `WHILE` directive.

Suggested Replacement

There is no direct replacement for this directive. See 4.62 While Directive for alternatives.

4.24   Equ Directive

The MPASM `EQU` directive equates a value with a symbol.

Suggested Replacement

The PIC Assembler's `EQU` directive is a direct replacement for this directive.

There must be no prior definition of the symbol used with `EQU`.

4.25   Error Directive

The MPASM `ERROR` directive generates a user-defined error message.

Suggested Replacement

The PIC Assembler's `ERROR` directive is a direct replacement for this directive.

4.26   Errorlevel Directive

The MPASM `ERRORLEVEL` directive controls which assembler messages are printed.

Suggested Replacement

The `-w` driver option can be used to suppress all warning messages produced by the assembler. Consider also the `-mwarn` option, which can restrict warnings to the specified level of severity.

4.27   Exitm Directive

The MPASM `EXITM` directive forces a premature exit from the macro.

Suggested Replacement

There is no replacement for this directive.

4.28   Expand Directive

The MPASM `EXPAND` directive requests that assembler macros be expanded in the listing file.

Suggested Replacement

The PIC Assembler's `EXPAND` directive is a direct replacement for this directive.

4.29   Extern Directive

The MPASM `EXTERN` directive links a symbol with a symbol globally defined in another module.

Suggested Replacement

The PIC Assembler's `EXTRN` directive (note the subtle difference in spelling) is a direct replacement for this directive.

An error will be issued if you use this directive with a symbol that was defined in the current module.

4.30   Fill Directive

The MPASM `FILL` directive fills memory with the specified value.

Suggested Replacement

The PIC Assembler's `--fill` driver option can be used to perform a similar task. See the *MPLAB® XC8 PIC Assembler User's Guide* for details of how this feature is used.

4.31   Global Directive

The MPASM `GLOBAL` directive declares symbols that may be shared by code in other modules.

Suggested Replacement

The PIC Assembler's `GLOBAL` directive is a direct replacement for this directive.

In other modules, you may use either `GLOBAL` or `EXTRN` to link in with the symbols declared by this directive.

4.32   Idata Directive

The MPASM `IDATA` directive creates a section for objects that must be initialised.

Suggested Replacement

Use any of the PIC Assembler's directives that can reserve space in a suitable data memory psect, and collate the initial values in a separate psect placed in program memory.

The following PIC18 example reserves the runtime memory for the variables in bank 1 and places the initial values for these variables in program memory. Assembler-provided psects were used in this example, but you can define you own suitable psects, if required.

```
#include <xc.inc>
;define space for the variables in RAM
PSECT udata_bank1
vars:
input:
  DS 2
output:
  DS 1

;place the initial values for the above in a matching order in program memory
PSECT data
iValues:
  DW 55AAh
  DB 67h
```

Your application will need to provide code that copies the initial values to the reserved data memory space, as shown in the following example which copies the values at `iValues` to `vars`.

```
PSECT text,class=CODE
copy0:
  movlw   low (iValues)
  movwf   tblptrl
  movlw   high(iValues)
  movwf   tblptrh
  movlw   low highword(iValues)
  movwf   tblptru
  tblrd*+
  movff   tablat, vars+0
  tblrd*+
  movff   tablat, vars+1
  tblrd*+
  movff   tablat, vars+2
  return
```

When copying larger amounts of data, consider using a loop and FSR register to indirectly write to the variables in data memory.

4.33   Idata_acs Directive

The MPASM `IDATA_ACS` directive creates a section for PIC18 access bank objects that must be initialised.

Suggested Replacement

Use any of the PIC Assembler's directives that can reserve space in a suitable data memory psect, and collate the initial values in a separate psect placed in program memory.

The following PIC18 example reserves the runtime memory for the variables in the Access Bank and places the initial values for these variables in program memory. Assembler-provided psects were used in this example, but you can define you own suitable psects, if required.

```
#include <xc.inc>
;define space for the variables in RAM
PSECT udata_acs
vars:
input:
  DS 2
output:
  DS 1

;place the initial values for the above in a matching order in program memory
PSECT data
iValues:
  DW 55AAh
  DB 67h
```

Your application will need to provide code that copies the initial values to the reserved data memory space, as shown in the following example which copies the values at `iValues` to `vars`.

```
PSECT text,class=CODE
copy0:
  movlw   low (iValues)
  movwf   tblptrl
  movlw   high(iValues)
  movwf   tblptrh
  movlw   low highword(iValues)
  movwf   tblptru
  tblrd*+
  movff   tablat, vars+0
  tblrd*+
  movff   tablat, vars+1
  tblrd*+
  movff   tablat, vars+2
  return
```

When copying larger amounts of data, consider using a loop and FSR register to indirectly write to the variables in data memory.

4.34   If Directive

The MPASM `IF` directive begins a conditional block of assembly code.

Suggested Replacement

The PIC Assembler's `IF` directive is a direct replacement for this directive.

The operand must be an absolute expression and if non-zero, then the code following the `IF` up to the next matching `ELSE`, `ELSIF` or `ENDIF` will be assembled. If the operand is zero, then the code up to the next matching `ELSE` or `ENDIF` will not be output. At an `ELSE`, the sense of the conditional compilation will be inverted, while an `ENDIF` will terminate the conditional assembly block.

The following shows the directive being used to include one of two calls into the output.

```
IF DEMO
  call demo_mode
ELSE
  call play_mode
ENDIF
```

Assembly code in both true and false cases is always scanned and interpreted, but the machine code corresponding to the instructions is output only if the condition matches. This implies that assembler directives (e.g., `EQU`) will be

processed regardless of the state of the condition expression, and so, should not be used inside an `IF` - `ENDIF` construct.

Although the MPASM assembler allows you to use a `#IF` form of this directive, note that it is still an assembler directive. The PIC Assembler's `#if` directive is a *preprocessor* directive, hence will be looking at the result of a *preprocessor* expression (potentially using preprocessor symbols), not an *assembler* expression (potentially using assembler symbols). If you migrate your code to use the preprocessor directive, ensure that you also examine the expressions involved, defining as required the equivalent preprocessor macros using either the `#define` directive or `-D` option.

4.35 Ifdef Directive

The MPASM `IFDEF` directive begins a conditional block of assembly code.

Suggested Replacement
As assembly source files can be preprocessed, the `#ifdef` preprocessor directive can be used as a replacement for this directive. Ensure the assembly source file uses a `.S` extension so that it will be preprocessed by the assembler. For example:

```
#ifdef DBG
  movf state,w
  call diag
#endif
```

Although the MPASM assembler allows you to use a `#IFDEF` form of this directive, note that it is still an assembler directive. The PIC Assembler's `#ifdef` directive is a *preprocessor* directive, hence will be looking for the definition of a *preprocessor* symbol, not an *assembler* symbol. If you migrate your code to use the preprocessor directive, ensure that you also define as required the equivalent preprocessor macros using either the `#define` directive or `-D` option.

4.36 Ifndef Directive

The MPASM `IFNDEF` directive begins a conditional block of assembly code.

Suggested Replacement
As assembly source files can be preprocessed, the PIC Assembler `#ifndef` preprocessor directive can be used as a replacement for this directive, as shown in the example below. Ensure the assembly source file uses a `.S` extension so that it will be preprocessed by the assembler. For example:

```
#ifndef RUNMODE
  movf state,w
  call diag
#endif
```

Although the MPASM assembler allows you to use a `#IFNDEF` form of this directive, note that it is still an assembler directive. The PIC Assembler's `#ifndef` directive is a *preprocessor* directive, hence will be looking for the definition of a *preprocessor* symbol, not an *assembler* symbol. If you migrate your code to use the preprocessor directive, ensure that you also define as required the equivalent preprocessor macros using either the `#define` directive or `-D` option.

4.37 #include Directive

The MPASM `#include` directive is textually replaced with the specified file.

Suggested Replacement
As assembly source files can be preprocessed, the PIC Assembler's `#include` preprocessor directive can be used as a direct replacement for this directive, as shown in the example below.

The search for filenames enclosed in angle brackets will be in the standard header locations. The search for quoted filenames will be in the current working directory first, then the standard header locations. Ensure the assembly source file uses a `.S` extension so that it will be preprocessed by the assembler.

```
#include <xc.inc>
#include "buttons.inc"
```

Alternatively, the PIC Assembler's `INCLUDE "file"` directive can be used as a replacement. No search paths are used. If the file to be included is not in the current working directory, the full path to the file must be specified with the directive. This directive cannot be used to include header files that contain preprocessor directives, so you cannot, for example use it to include `<xc.inc>`.

```
INCLUDE "buttons.inc"
```

4.38   List Directive

The MPASM `LIST` directive enables and controls content in the listing file.

Suggested Replacement
The PIC Assembler's `LIST options` directive performs many of the functions of this directive, as shown in Table 4-2.

Table 4-2. Equivalent List Options

MPASM List Option	Purpose	Suggested PIC Assembler Replacement
b=*nnn*	Set tabs spaces	No replacement available
c=*nnn*	Set column width	c=*nnn*
f=*format*	Set the hex file format	Use the `-g` driver option
free	Use free-format parser	No replacement available
fixed	Use fixed-format parser	No replacement available
mm=[on|off]	Print memory map in listing	Use the `-msummary` option to print memory usage to the console
n=*nnn*	Set lines per page	n=*nnn*
p=*device*	Select device	p=*device*
pe=*type*	Select device and enable PIC18 extended instruction mode	Use the `-mcpu` option in conjunction with the `-misa` option
r=*radix*	Set source code radix	Use the `RADIX` assembler directive
st=[on|off]	Print symbol table in listing	No replacement available, but a symbol table is always produced in the listing
t=[on|off]	Truncate listing lines	t=[on|off]
w=[0|1|2]	Set message level	Use the `-w` option to suppress warnings
x=[on|off]	Enable macro expansion	x=[on|off]

4.39 Local Directive

The MPASM `LOCAL` directive defines a local label inside a macro.

Suggested Replacement

The PIC Assembler's `LOCAL` directive is direct replacement. Separate multiple label names with a comma

4.40 Macro Directive

The MPASM `MACRO` directive defines a macro.

Suggested Replacement

The PIC Assembler's `MACRO` directive is direct replacement.

Within a macro definition, the `&` character can be used to permit the concatenation of macro arguments with other text, but is removed in the actual expansion. For example:

```
loadPort MACRO port, value
  movlw value
  movwf PORT&port
ENDM
```

will load `PORTA` if `port` is `A` when called, etc. The special meaning of the `&` token in macros implies that you must only use the `and` form of the bitwise AND operator.

A comment can be suppressed within the expansion of a macro by opening the comment with two semicolons, `;;`.

When invoking a macro, the argument list must be comma-separated. If it is desired to include a comma (or other delimiter such as a space) in an argument then angle brackets `<` and `>` can be used to quote.

If an argument is preceded by a percent sign, `%`, that argument will be evaluated as an expression and passed as a decimal number, rather than as a string. This is useful if evaluation of the argument inside the macro body would yield a different result.

The `nul` operator can be used within a macro to test a macro argument, for example:

```
IF nul     arg3  ;argument was not supplied.
...
ELSE             ;argument was supplied
...
ENDIF
```

4.41 Maxram and Maxrom Directives

The MPASM `__maxram` and `__maxrom` directives, along with the `__badram` and `__badrom` directives specifies file register address ranges that should be flagged as being invalid should they be used by code.

Suggested Replacement

There is no replacement for these directives.

4.42 Messg Directive

The MPASM `MESSG` directive generates a user-defined advisory message.

Suggested Replacement

The PIC Assembler's `MESSG` directive is a direct replacement for this directive.

4.43 Noexpand Directive

The MPASM NOEXPAND directive requests that assembler macros not be expanded in the listing file.

Suggested Replacement

The PIC Assembler's NOEXPAND directive is a direct replacement for this directive.

4.44 Nolist Directive

The MPASM NOLIST directive enables and controls content in the listing file.

Suggested Replacement

The PIC Assembler's NOLIST directive is a direct replacement this directive.

4.45 Org Directive

The MPASM ORG directive sets the initial location counter for subsequent code to be the specified address.

Suggested Replacement

The PIC Assembler's ORG directive performs a similar task to this directive, but there are some differences in its operation.

The ORG directive changes the value of the location counter *within the current psect* to be that specified. This means that the address set by the ORG directive is relative to the base address of the psect, which is typically not determined until link time. For example, using ORG 0100h inside a psect that is ultimately linked to address 0x2000 will move the location counter to address 0x2100. Only if the psect in which this directive is placed is absolute (uses the abs flag) and overlaid (ovrld flag) will the location counter be moved to the absolute address specified.

The ORG directive is seldom needed in programs. To have code or data located at a particular address, place it in a unique psect and have the linker position that psect at the required location.

4.46 Page Directive

The MPASM PAGE directive inserts a page eject into the listing file.

Suggested Replacement

There is no direct replacement for this directive.

4.47 Pagesel Directive

The MPASM PAGESEL directive generates page selection code appropriate for the label specified as the argument.

Suggested Replacement

The PIC Assembler's PAGESEL directive is a direct replacement for this directive.

4.48 Pageselw Directive

The MPASM PAGESELW directive generates page selection code (using WREG as an intermediate register) appropriate for the label specified as the argument.

Suggested Replacement

The PIC Assembler's `PAGESEL` directive performs a similar function but will never use WREG to adjust page selection bits. The output page selection code will perform bit operations on the bits that must be toggled.

4.49   Processor Directive

The MPASM `PROCESSOR` directive sets the device type.

Suggested Replacement

The PIC Assembler's `PROCESSOR` directive is a direct replacement for this directive.

The device must be selected using the `-mcpu` driver option, but this directive may be used to ensure that the source code is built for the intended device. An error will be generated if the device specified by this directive clashes with that set by the option.

4.50   Radix Directive

The MPASM `RADIX` directive sets the radix for constants specified in the assembly source.

Suggested Replacement

The PIC Assembler's `RADIX` directive is a direct replacement for this directive.

4.51   Res Directive

The MPASM `RES` directive reserves memory by advancing the location counter.

Suggested Replacement

The PIC Assembler's `DS` directive has a similar function, but there are some differences in its operation.

The `DS` directive advances the location counter, allowing memory to be allocated to a label defined before the directive. Typically it is used inside psects linked to the data space, providing a mechanism to reserve memory for variables, as in the following example.

```
PSECT udata_bank0
input:
  DS 2  ;allocate 2 bytes for input
output:
  DS 4  ;allocate 4 bytes for output
```

If the directive is used in a psect assigned to program memory (`space=0` flag), it will move the location counter, but not place anything in the HEX file output.

The address units of this directive are determined by context, in particular, the `delta` and `bit` flag values associated with the psect in which it resides. If the `delta` flag is set to 1, then the directive moves the location counter forward by the specified number of bytes; if it is 2, it moves it forward by 16-bit words, etc. When the `bit` flag is used, the allocation units are bits.

4.52   Set Directive

The MPASM `SET` directive equates a value with a symbol.

Suggested Replacement

The PIC Assembler's `SET` directive is a direct replacement for this directive.

The symbol can be redefined without error.

4.53   Space Directive

The MPASM `SPACE` directive inserts blank lines into the assembly listing file.

Suggested Replacement
The PIC Assembler's `SPACE` assembler directive is a direct replacement for this directive.

4.54   Subtitle Directive

The MPASM `SUBTITLE` directive specifies the listing file subtitle string.

Suggested Replacement
The PIC Assembler's `SUBTITLE` assembler directive is a direct replacement for this directive.

4.55   Title Directive

The MPASM `TITLE` directive specifies the listing file title string.

Suggested Replacement
The PIC Assembler's `TITLE` assembler directive is a direct replacement for this directive.

4.56   Udata Directive

The MPASM `UDATA` directive creates a section for objects that are uninitialised..

Suggested Replacement
Use the predefined `udata_bank`*n* psect, where *n* represents the bank number in which the psect should reside, or a create a similar psect, ensuring the flags are suitable for a section containing variables on the target device.

The PIC Assembler provides the `udata_bank`*n* psect once you include `<xc.inc>`. You can use this psect without having to specify any psect flags, for example:

```
#include <xc.inc>

PSECT udata_bank1
;data goes here
```

Alternatively, you can define your own psect with any name and suitable psect flags. The psect's `space` flag must be 1, to indicate that the psect should be positioned in data memory. Typically, you would use the `class` flag to assign the psect to one of the BANK*N* linker classes, where *N* represents the bank number for which the class defines. These classes are also predefined by the driver, so that the psect will be positioned somewhere in the memory associated with the specified class without you having to specify any linker options. You could also position this psect at a particular address using the linker's `-p` option, passed to the linker from the driver's `-Wl` option. For example:

```
PSECT myData,space=1,class=BANK1
;data goes here
```

4.57   Udata_acs Directive

For PIC18 devices, the MPASM `UDATA_ACS` directive creates a section for Access bank objects that are uninitialised..

Suggested Replacement

Use the predefined `udata_acs` psect or a create a similar psect, ensuring the flags are suitable for a section containing Access bank variables on the PIC18 device.

The PIC Assembler provides the `udata_acs` psect once you include `<xc.inc>`. You can use this psect without having to specify any psect flags, for example:

```
#include <xc.inc>

PSECT udata_acs
;data goes here
```

Alternatively, you can define your own psect with any name and suitable psect flags. The psect's `space` flag must be 1, to indicate that the psect should be positioned in data memory. Typically, you would use the `class` flag to assign the psect to one of the `COMRAM` linker classes, which is also predefined by the driver, so that the psect will be positioned somewhere in the memory associated with the specified class without you having to specify any linker options. You could also position this psect at a particular address using the linker's `-p` option, passed to the linker from the driver's `-Wl` option. For example:

```
PSECT myData,space=1,class=COMRAM
;data goes here
```

4.58   Udata_ovr Directive

For Baseline and Mid-range devices, the MPASM `UDATA_OVR` directive creates an overlaid section for objects that are uninitialised.

Suggested Replacement

Define a psect with the `ovrld` flag set. Associate the psect with the access bank linker class, `COMMON` to have it linked somewhere in the common memory on Baseline or Mid-range devices.

The example following shows objects placed into the `myData` psect, which is used in two different modules (file 1 and file 2). The `PSECT` directive uses the same psect name, `ovrld` flag (to indicate that the psects will be overlaid), and `space=1` flag (to indicate the psects will reside in the data space memory). The psects are associated with the `COMMON` linker class, which defines the common memory, so once overlaid, `myData` will appear anywhere in the memory defined by this class.

```
;file 1
PSECT myData,space=1,ovrld,class=COMMON
zero:
  DS 1
  ;leave a 1-byte gap for another object here
  ORG 2
two:
  DS 1

;file 2
PSECT myData,space=1,ovrld,class=COMMON
  ORG 1
one:
  DS 1
```

Note that the contributions to an overlaid psect are concatenated in each module, but the psects from each module are then overlaid at link time. When the above example is built, the labels will appear in memory in the order `zero`, `one`, `two`.

The `ORG` directive in the above example has allowed the psects' content to interleave. If this directive had not been used in file 1, the space associated with the label `zero` and the label `one` would overlap, and these objects would appear at the same address. Such code is legal and may be desired in some applications.

Overlaid psects can be linked into any RAM area, not just common memory. This construct will work on any device with the selection of a suitable linker class.

4.59 Udata_shr Directive

On Baseline and Mid-range devices, the MPASM `UDATA_SHR` directive creates a section for unbanked objects that are uninitialised.

Suggested Replacement

Use the predefined `udata_shr` psect or a create a similar psect, ensuring the flags are suitable for a section containing variables on the Baseline or Mid-range device.

The PIC Assembler provides the `udata_shr` psect once you include `<xc.inc>`. You can use this psect without having to specify any psect flags, for example:

```
#include <xc.inc>

PSECT udata_shr
;data goes here
```

Alternatively, you can define your own psect with any name and suitable psect flags. The psect's `space` flag must be 1, to indicate that the psect should be positioned in data memory. Typically, you would use the `class` flag to assign the psect to one of the `COMMON` linker classes, which is also predefined by the driver, so that the psect will be positioned somewhere in the memory associated with the specified class without you having to specify any linker options. You could also position this psect at a particular address using the linker's `-p` option, passed to the linker from the driver's `-Wl` option. For example:

```
PSECT myData,space=1,class=COMMON
;data goes here
```

4.60 #define Directive

The MPASM `#define` directive defines textual replacement for a macro.

Suggested Replacement

As assembly source files can be preprocessed, the PIC Assembler's `#define` preprocessor directive can be used as a direct replacement for this directive, as shown in the example below. All the usual preprocessor features associated with macro replacement are available with this directive.

Ensure the assembly source file uses a `.S` extension so that it will be preprocessed by the assembler.

```
#define SUMSP(a, b) (a+2*b)
PSECT text,class=CODE
process:
  movlw SUMSP(5, 3)
  movwf volume
```

4.61 Variable Directive

The MPASM `VARIABLE` directive creates a variable set to be a value.

Suggested Replacement

There is no replacement for this directive, but the PIC Assembler's `SET` directive performs a similar task.

The `SET` directive associates a value with a symbol. Although the symbol's value can be redefined using another `SET` directive without error, it is not possible to perform operations on the symbol's value in other way.

4.62 While Directive

The MPASM `WHILE` directive loops over a block of assembly code while some condition is true.

Suggested Replacement

There is no replacement for this directive, but the PIC Assembler's REPT directive performs a similar task.

The block of code following REPT and terminated by ENDM is output a number of times, based on the directive's argument.

The following example shows a shift instruction that will be performed 4 times.

```
REPT 4
  rlncf mask,f
ENDM
```

5. Linking

The linker is always invoked when building projects using the PIC Assembler, unless you stop the build prematurely by using the `-c` option. The linker must be executed to obtain a final program image to download to your hardware.

The linker is controlled by its own set of options. These options include those that specify the memory arrangement of the device. Linker scripts are not and cannot be used.

You do not need to run the linker explicitly to specify or change the linker options. Several `pic-as` driver options indirectly control the linker, and the `-Wl,` driver option can be used to pass through linker options directly to the linker. This option also allows you to override some of the default options issued to the linker by the driver.

5.1 Reserving memory

You can make memory unavailable for your code and data by using the `-mreserve` driver option. Alternatively, you can also use the `-mram` and `-mrom` options to do the same thing. For example `-mreserve=ram@100:103` will remove the range 100-103h in data memory. You could also use `-mram=default,-100-103`. To reserve program memory, use for example, `-mreserve=rom@1800-1fff` or `-mrom=default,-1800-1fff.`.

When reserving memory, it is removed from all of the linker classes that include that memory range, and thus this memory will not be used by any psect placed into those classes. Memory reservations will not affect any psect that has been linked to an absolute address, nor those placed relative to other psects. To move those, you must change the linker option that places them in that location.

5.2 Placing Psects into Memory

All code and objects must be placed in a psect (program section), which is just a way of grouping together parts of a program under the psect's name. All psects are allocated memory by the linker, after which, the values for any labels defined in those psects can be determined.

When placing a psect into memory, the linker performs the first of the following operations which matches the situation.

- If the psect specifies the `abs` flag, it is placed at address 0 in the memory space indicated by the psect's `space` flag, or the program memory (default) space if no space has been specified.
- If a `-p` linker option references the psect name, the psect is placed at the location specified by that option in the memory space indicated by the psect's `space` flag, or the program memory (default) space if no space has been specified.
- If the psect is associated with a linker class, the psect is placed at any free location in the address ranges defined by that class.
- If the psect specifies a space number, it is placed at a free location in that memory space (Not recommended).
- The psect is placed in a free location in the program memory (default) space (Not recommended).

Some situations are illegal, for example if you use a `-p` option to place a psect that also uses the `abs` flag, then an error will be issued. It is recommended that psects are always linked using the `abs` flag, using a `-p` option, or via a linker class (the first three of the above methods). If the linker has to position a psect with no guidance from the user (the last two of the above methods), a warning similar to, `(526) psect "wanderer" not specified in -P option (first appears in "not_right.o")`, will be emitted.

In most cases, psects can be linked anywhere in a suitable address range that is dictated by the device. For example, most executable code can be placed anywhere in program memory, or at least anywhere in a program memory page. Data objects can usually be placed anywhere in a data bank. In this case, the easiest way to have these psects linked is to associate them with a linker class. If you are using a psect provided by the PIC Assembler, then these are already associated with a suitable linker class and you do not need to specify any linker options to have them correctly linked. In the following example,

```
PSECT udata_bank1
myVar:
  DS 2
```

the `udata` psect has already been associated with the `RAM` linker class and will be linked anywhere in free memory associated with that class.

If you have created your own psect, you can associate it with any of the existing linker classes provided by the PIC Assembler by using the `class` flag with the psect definition. In the following example, a psect has been created by the programmer to use instead of `udata`.

```
PSECT machData,space=1,class=MDATA
myVar:
  DS 2
```

This psect uses a new class, `MDATA`, which will need to be defined by a linker option. To do that, use, for example, the driver option, `-Wl,-AMDATA=050h-05fh`, which passes the `-A` linker option directly to the linker and which will associate the specified address range with the `MDATA` class.

There are, however, times when a psect must be placed at a specific address. The reset vector code is one good example, as are interrupt routines. In this case, you will need to use a `-p` linker option to place the psect at the desired location. This might be as simple as providing an absolute address, for example using the driver option `-Wl,-pInterrupt=08h` to place the psect called `Interrupt` at address 8, but there are more advanced usages of this option.

Check the *MPLAB® XC8 PIC Assembler User's Guide* for full details concerning the psects and linker classes provided by the PIC Assembler, as well as the linker and driver options mentioned in this section.

The Microchip Website

Microchip provides online support via our website at http://www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to http://www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: http://www.microchip.com/support

Product Identification System

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

PART NO.   [X]⁽¹⁾   -   X   /XX   XXX

Device   Tape and Reel   Temperature   Package   Pattern
          Option   Range

Device:	PIC16F18313, PIC16LF18313, PIC16F18323, PIC16LF18323	
Tape and Reel Option:	Blank	= Standard packaging (tube or tray)
	T	= Tape and Reel[1]
Temperature Range:	I	= -40°C to +85°C (Industrial)
	E	= -40°C to +125°C (Extended)
Package:[2]	JQ	= UQFN
	P	= PDIP
	ST	= TSSOP
	SL	= SOIC-14
	SN	= SOIC-8
	RF	= UDFN
Pattern:	QTP, SQTP, Code or Special Requirements (blank otherwise)	

Examples:

- PIC16LF18313- I/P Industrial temperature, PDIP package
- PIC16F18313- E/SS Extended temperature, SSOP package

Note:

1. Tape and Reel identifier only appears in the catalog part number description. This identifier is used for ordering purposes and is not printed on the device package. Check with your Microchip Sales Office for package availability with the Tape and Reel option.
2. Small form-factor packaging options may be available. Please check http://www.microchip.com/packaging for small-form factor package availability, or contact your local Sales Office.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2020, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-5884-5

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit http://www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: http://www.microchip.com	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79
Atlanta Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455	**China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115	**Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200	**Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400
Austin, TX Tel: 512-257-3370	**China - Hong Kong SAR** Tel: 852-2943-5100	**Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906	**Germany - Heilbronn** Tel: 49-7131-72400
Boston Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088	**China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355	**Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065	**Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44
Chicago Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075	**China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829	**Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366	**Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705
Dallas Addison, TX Tel: 972-818-7423 Fax: 972-818-2924	**China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526	**Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600	**Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286
Detroit Novi, MI Tel: 248-848-4000	**China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252	**Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340
Houston, TX Tel: 281-894-5983	**China - Xiamen** Tel: 86-592-2388138		**Norway - Trondheim** Tel: 47-72884388
Indianapolis Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380	**China - Zhuhai** Tel: 86-756-3210040		**Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50
Los Angeles Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800			**Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91
Raleigh, NC Tel: 919-844-7510			**Sweden - Gothenberg** Tel: 46-31-704-60-40
New York, NY Tel: 631-435-6000			**Sweden - Stockholm** Tel: 46-8-5090-4654
San Jose, CA Tel: 408-735-9110 Tel: 408-436-4270			**UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820
Canada - Toronto Tel: 905-695-1980 Fax: 905-695-2078			