COMMAND LINE

AT90S COMMAND LINE

Interface Guide

COPYRIGHT NOTICE

DISCLAIMER

The information in this document is subject to change without notice and does not represent a commitment on any part of IAR Systems. While the information contained herein is assumed to be accurate, IAR Systems assumes no responsibility for any errors or omissions.

In no event shall IAR Systems, its employees, its contractors, or the authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claim for lost profits, fees, or expenses of any nature or kind.

TRADEMARKS

First edition: September 1996
Part no: CLA90–1

This documentation was produced by Human-Computer Interface.

WELCOME

Welcome to the IAR Systems development tools for the ATMEL AT90S Series microprocessors.

Before reading this guide refer to the *QuickStart Card*, or the chapter *Installation and documentation route map*, for information about installing the tools and an overview of the documentation.

This guide explains how to configure and run the IAR Systems development tools from the command line. It also includes reference information about the command line environment variables.

Refer to the *AT90S C Compiler Programming Guide* and *AT90S Assembler, Linker, and Librarian Programming Guide* for detailed information about working with the individual development tools.

If your product includes the optional AT90S C-SPY debugger refer to the *AT90S C-SPY User Guide* for information about debugging with C-SPY.

ABOUT THIS GUIDE

This guide consists of the following chapters:

Installation and documentation route map explains how to install and run the IAR Systems tools, and gives an overview of the documentation supplied with them.

The *Introduction* provides a brief summary of the IAR Systems tools, and describes how you would use them to develop a typical project.

Getting started lists the file extensions used by the AT90S tools. It also gives details of the files installed by the installation procedure, and explains how to run the tools from the command line.

The *Tutorial* demonstrates how to use the most important features of the tools to develop, compile, link, and debug a simple C project.

XLINK environment variables gives reference information about the XLINK Linker environment variables.

XLIB environment variables gives reference information about the XLIB Librarian environment variables.

THE OTHER GUIDES

The other guides provided with the IAR Systems tools are as follows:

AT90S C Compiler Programming Guide

This guide provides programming information about the AT90S C Compiler. It includes reference information about the C library functions and language extensions, and provides information about support for the target-specific options such as memory models.

You should refer to this guide for details of the C compiler command line options, and for information about the C language when writing and debugging C source programs.

This guide also includes a list of the C compiler diagnostic messages.

AT90S Assembler, Linker, and Librarian Programming Guide

This guide provides reference information about the AT90S Assembler, XLINK Linker, and XLIB Librarian.

The assembler programming sections include details of the assembler source format, and reference information about the assembler operators, directives, and mnemonics.

The XLINK Linker programming reference sections provide information about the XLINK Linker commands and output formats.

The XLIB Librarian programming sections provide information about the XLIB Librarian commands.

Finally, the guide includes a list of diagnostic messages for each of these tools.

AT90S C-SPY User Guide

This optional guide describes how to use C-SPY for the AT90S Series microprocessors, and provides reference information about the features of C-SPY.

ASSUMPTIONS AND CONVENTIONS

ASSUMPTIONS

This guide assumes that you already have a working knowledge of the following:

- The C programming language.
- The AT90S Series processor you are using.
- The procedures for running commands from the command line.

CONVENTIONS

This user guide uses the following typographical conventions:

Style	*Used for*
`computer`	Text that you type in, or that appears on the screen.
`parameter`	A label representing the actual value you should type as part of a command.
`[`*`option`*`]`	An optional part of a command.
bold	Names of menus, menu commands, buttons, and dialog boxes that appear on the screen.
reference	A cross-reference to another part of this guide, or to another guide.

CONTENTS

INSTALLATION AND DOCUMENTATION ROUTE MAP

This chapter explains how to install and run the command line and Windows Workbench versions of the IAR products, and gives an overview of the user guides supplied with them.

Please note that some products only exist in a command line version, and that the information may differ slightly depending on the product or platform you are using.

COMMAND LINE VERSIONS

This section describes how to install and run the command line versions of the IAR Systems tools.

WHAT YOU NEED

- DOS 4.x or later. This product is also compatible with a DOS window running under Windows 95, Windows NT 3.51 or later, or Windows 3.1x.
- At least 10 Mbytes of free disk space.
- A minimum of 4 Mbytes of RAM available for the IAR applications.

INSTALLATION

1 Insert the first installation disk.

2 At the MS-DOS prompt type:

```
a:\install ↵
```

3 Follow the instructions on the screen.

When the installation is complete:

4 Make the following changes to your `autoexec.bat` file:

Add the paths to the IAR Systems executable and user interface files to the `PATH` variable; for example:

```
PATH=c:\dos;c:\utils;c:\iar\exe;c:\iar\ui;
```

Define environment variables `C_INCLUDE` and `XLINK_DFLTDIR` specifying the paths to the `inc` and `lib` directories; for example:

```
set C_INCLUDE=c:\iar\inc\
set XLINK_DFLTDIR=c:\iar\lib\
```

5 Reboot your computer for the changes to take effect.

6 Read the Read-Me file, named *`product`*`.doc`, for any information not included in the guides.

RUNNING THE TOOLS

Type the appropriate command at the MS-DOS prompt.

For more information refer to the chapter *Getting started* in the *Command Line Interface Guide*.

WINDOWS WORKBENCH VERSIONS

This section explains how to install and run the Embedded Workbench.

WHAT YOU NEED

- Windows 95, Windows NT 3.51 or later, or Windows 3.1x.
- Up to 15 Mbytes of free disk space for the Embedded Workbench.
- A minimum of 4 Mbytes of RAM for the IAR applications.

If you are using C-SPY you should install the Workbench before C-SPY.

INSTALLING FROM WINDOWS 95 OR NT 4.0

1 Insert the first installation disk.

2 Click the **Start** button in the taskbar, then click **Settings** and **Control Panel**.

3 Double-click the **Add/Remove Programs** icon in the **Control Panel** folder.

4 Click **Install**, then follow the instructions on the screen.

RUNNING FROM WINDOWS 95 OR NT 4.0

1 Click the **Start** button in the taskbar, then click **Programs** and **IAR Embedded Workbench**.

2 Click **IAR Embedded Workbench**.

INSTALLING FROM WINDOWS 3.1x OR NT 3.51

1 Insert the first installation disk.

2 Double-click the **File Manager** icon in the **Main** program group.

3 Click the **a** disk icon in the **File Manager** toolbar.

4 Double-click the **setup.exe** icon, then follow the instructions on the screen.

RUNNING FROM WINDOWS 3.1x OR NT 3.51

1 Go to the Program Manager and double-click the **IAR Embedded Workbench** icon.

RUNNING C-SPY

Either:

1 Start C-SPY in the same way as you start the Embedded Workbench (see above).

Or:

1 Choose **Debugger** from the Embedded Workbench **Project** menu.

UNIX VERSIONS

This section describes how to install and run the UNIX versions of the IAR Systems tools.

WHAT YOU NEED

◆ HP9000/700 workstation with HP-UX 9.x (minimum), or a
Sun 4/SPARC workstation with SunOS 4.x (minimum) or
Solaris 2.x (minimum).

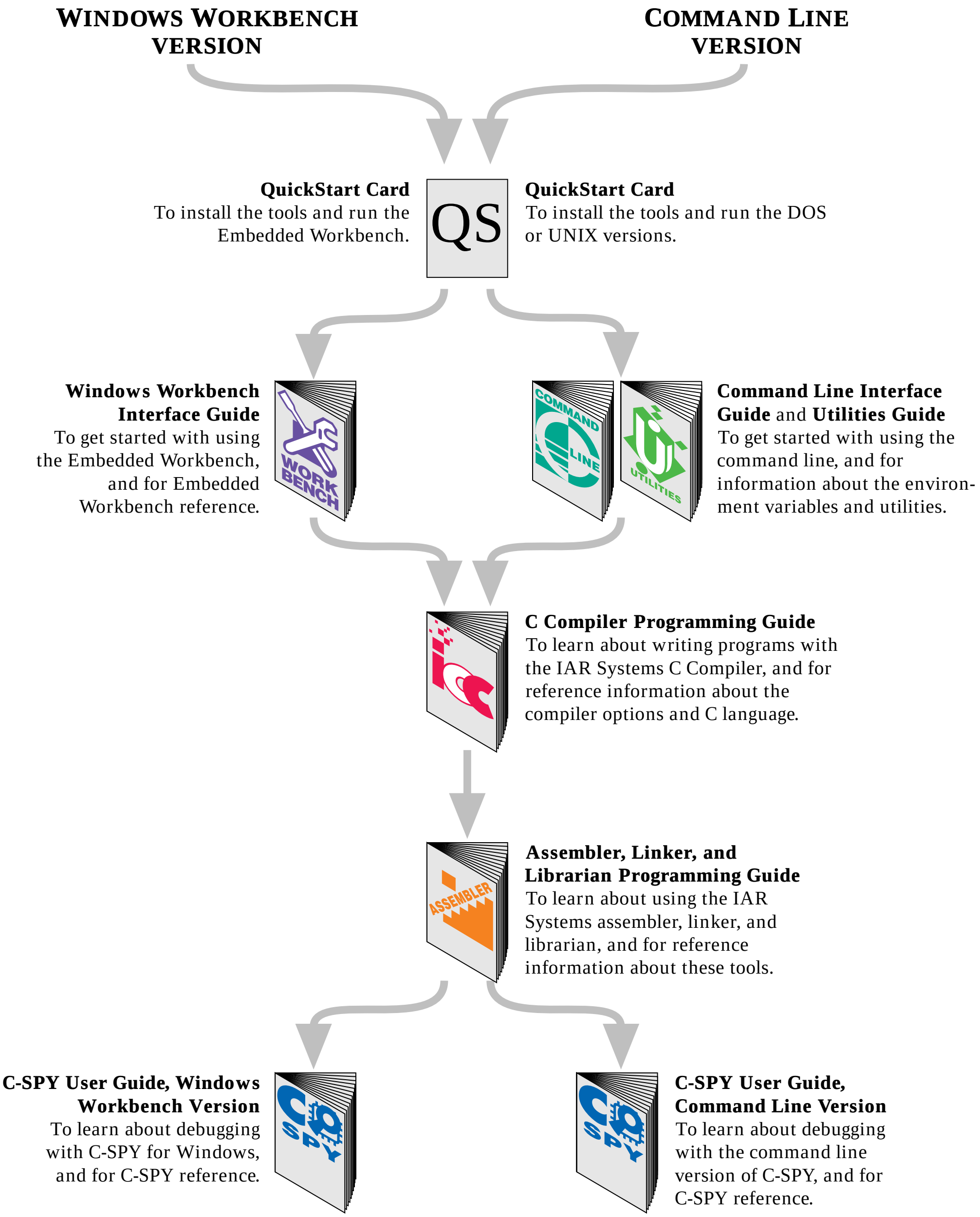
INSTALLATION

Follow the instructions provided with the media.

RUNNING THE TOOLS

Type the appropriate command at the UNIX prompt. For more information refer to the chapter *Getting started* in the *Command Line Interface Guide.*

DOCUMENTATION ROUTE MAP



WINDOWS WORKBENCH VERSION

COMMAND LINE VERSION

QuickStart Card
To install the tools and run the Embedded Workbench.

QuickStart Card
To install the tools and run the DOS or UNIX versions.

Windows Workbench Interface Guide
To get started with using the Embedded Workbench, and for Embedded Workbench reference.

Command Line Interface Guide and **Utilities Guide**
To get started with using the command line, and for information about the environment variables and utilities.

C Compiler Programming Guide
To learn about writing programs with the IAR Systems C Compiler, and for reference information about the compiler options and C language.

Assembler, Linker, and Librarian Programming Guide
To learn about using the IAR Systems assembler, linker, and librarian, and for reference information about these tools.

C-SPY User Guide, Windows Workbench Version
To learn about debugging with C-SPY for Windows, and for C-SPY reference.

C-SPY User Guide, Command Line Version
To learn about debugging with the command line version of C-SPY, and for C-SPY reference.

INTRODUCTION

The IAR Systems range of integrated development tools provides C compilers, assemblers, and debuggers to support a wide choice of target microprocessors. In addition, the range includes an XLINK Linker and XLIB Librarian for use across all targets.

C COMPILER

The IAR Systems C Compiler for the AT90S Series microprocessors offers the standard features of the C language, plus many extensions designed to take advantage of the AT90S-specific facilities. The compiler is supplied with the IAR Systems AT90S Assembler, with which it is integrated and shares linker and librarian manager tools.

It provides the following features:

LANGUAGE FACILITIES

- Conformance to the ANSI specification.
- Standard library of functions applicable to embedded systems, with source optionally available.
- IEEE-compatible floating-point arithmetic.
- Powerful extensions for AT90S-specific features, including efficient I/O.
- LINT-like checking of program source.
- Linkage of user code with assembly routines.
- Long identifiers – up to 255 significant characters.
- Up to 32000 external symbols.

PERFORMANCE

- Fast compilation.
- Memory-based design which avoids temporary files or overlays.
- Rigorous type checking at compile time.
- Rigorous module interface type checking at link time.

CODE GENERATION

- Selectable optimization for code speed or size.
- Comprehensive output options, including relocatable binary, ASM, ASM + C, XREF, etc.
- Easy-to-understand error and warning messages.
- Compatibility with the C-SPY high-level debugger.

TARGET SUPPORT

- Tiny and small memory models.
- Flexible variable allocation.
- Interrupt functions requiring no assembly language.
- A #pragma directive to maintain portability while using processor-specific extensions.

DOCUMENTATION

The AT90S C Compiler is documented in the *AT90S C Compiler Programming Guide*.

ASSEMBLER

The IAR Systems AT90S Assembler is a powerful relocating macro assembler with a versatile set of directives.

The assembler incorporates a high degree of compatibility with the microprocessor manufacturer's own assemblers, to ensure that software originally developed using them can be transferred to the IAR Systems Assembler with little or no modification.

It provides the following features:

GENERAL

- One pass assembly, for faster execution.
- Integration with the XLINK Linker and XLIB Librarian.
- Integration with other IAR Systems software.
- Self explanatory error messages.

ASSEMBLER FEATURES

- Structured control directives.
- Support for AT90S processors.
- Up to 256 relocatable segments per module.
- 255 significant characters in symbols.
- Powerful recursive macro facilities.
- Number of symbols and program size limited only by available memory.
- Support for complex expressions with external references.
- Forward references allowed to any depth.
- Support for C language pre-processor directives and `sfrb`/`sfrw` keywords.
- Macros in Intel/Motorola style.

DOCUMENTATION

The AT90S Assembler is documented in the *AT90S Assembler, Linker, and Librarian Programming Guide*.

XLINK LINKER

The IAR Systems XLINK Linker converts one or more relocatable object files produced by the IAR Systems Assembler or C Compiler to machine code for a specified target processor. It supports a wide range of industry-standard loader formats, in addition to the IAR Systems debug format used by the C-SPY high level debugger.

XLINK supports user libraries, and will load only those modules that are actually needed by the program you are linking.

The final output produced by XLINK is an absolute, target-executable object file that can be programmed into an EPROM, downloaded to a hardware emulator, or run directly on the host computer using the IAR Systems C-SPY debugger.

XLINK provides the following features:

FEATURES OF XLINK

- Unlimited number of input files.
- Fast memory-based linking, or optional disk-based operation for programs with large amounts of code, data, or symbols.
- Linker commands can be entered on the XLINK command line, read from an extended command file, or a combination of the two, making XLINK easy and flexible to use.
- Searches user-defined library files and loads only those modules needed by the application.
- Symbols may be up to 255 characters long with all characters being significant. Both upper and lower case may be used.
- Global symbols can be defined at link time.
- Flexible segment commands allow full control of the locations of relocatable code and data in memory.
- Support for over 30 emulator formats.

DOCUMENTATION

The XLINK Linker is documented in the *AT90S Assembler, Linker, and Librarian Programming Guide*.

XLIB LIBRARIAN

The IAR Systems XLIB Librarian enables you to manipulate the relocatable object files produced by the IAR Systems Assembler and C Compiler.

XLIB provides the following features:

FEATURES OF XLIB

- Support for modular programming.
- Modules can be listed, added, inserted, replaced, deleted, or renamed.
- Segments can be listed and renamed.
- Symbols can be listed and renamed.

- Modules can be changed between program and library type.
- Interactive or batch mode operation.
- A full set of library listing operations.
- A command to display a directory of relevant files on disk.

DOCUMENTATION

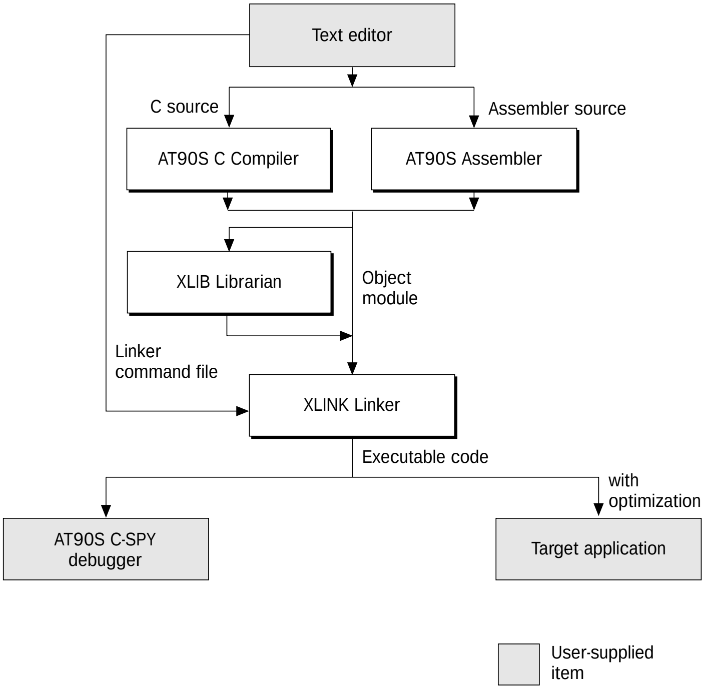
The XLIB Librarian is documented in the *AT90S Assembler, Linker, and Librarian Programming Guide*.

C-SPY DEBUGGER

An optional C-SPY debugger can be used with the IAR Systems tools, to run and debug AT90S object code programs.

DEVELOPMENT SYSTEM STRUCTURE

The following diagram shows how the IAR Systems tools are used to form a complete development system:



The text editor may be any standard ASCII editor, such as BRIEF, PMATE, or EMACS. The C compiler accepts C source files and produces code modules, normally in the IAR Systems proprietary Universal Binary Relocatable Object Format (UBROF).

The XLINK Linker then combines these code modules with modules created by the assembler, and library modules supplied as standard or created previously by the user using the library manager, XLIB.

The output of XLINK is either debuggable code for use in the C-SPY debugger or an alternative one, or final executable code for use in the target application. This executable code is in any of the many standard formats for use in emulators, EPROM, or ROM.

GETTING STARTED

This chapter lists the file extensions used by the AT90S tools. It also gives details of the files installed by the installation procedure, and explains how to run the tools from the command line.

FILE TYPES

The AT90S versions of the IAR Systems tools use the following default file extensions to identify different types of file:

Ext.	*Type of file*	*Output from*	*Input to*
`.doc`	ASCII documentation	–	Text editor
`.exe`	MS-DOS program	–	MS-DOS command
`.c`	C program source	Text editor	C compiler
`.h`	C header source	Text editor	C compiler `#include`
`.s90`	Assembler program source	Text editor	Assembler
`.xcl`	Extended command line	Text editor	XLINK and C compiler
`.r90`	Object module	C compiler and assembler	XLINK and XLIB
`.a90`	Target program	XLINK	EPROM, C-SPY, etc
`.d90`	Target program with debug information	XLINK	C-SPY, etc
`.lst`	List file	C compiler and assembler	–
`.map`	XLINK map	XLINK	–

The default extension may be overridden by simply including an explicit extension when specifying a filename.

Note that, by default, XLINK listings (maps) will have the `.lst` extension, and this may overwrite the listing file generated by the compiler. It is recommended that you explicitly name XLINK map files, for example `demo1.map`.

INSTALLED FILES

The installation procedure creates several directories to contain the different types of files used with the IAR Systems tools. The following sections give a description of the files contained in each directory.

The paths given in the following sections assume you chose the default installation directory, `c:\iar`.

Note that the list of files given here is provisional, and changes or additions may be made to reflect enhancements to the IAR Systems tools.

DOCUMENTATION FILES

Your installation may include a number of ASCII-format text files (`*.doc`) containing recent additional information. It is recommended that you read all of these files before proceeding.

ASSEMBLER FILES

The `c:\iar\aa90` subdirectory holds the tutorial, test, and document files for the AT90S Assembler.

DONGLE FILES

The `c:\iar\dongle` directory, and its subdirectories, contains dongle drivers for use with Windows NT and read me files for use in case of problems with the dongle protection system.

MISCELLANEOUS FILES

The `c:\iar\etc` subdirectory holds the source files for various library functions.

USER INTERFACE FILES

The `c:\iar\ui` subdirectory holds the user interface files.

EXECUTABLE FILES

The `c:\iar\exe` subdirectory holds the executable program files.

The installation procedure also includes an addition to the `autoexec.bat` `PATH` statement, directing MS-DOS to search the `exe` subdirectory for command files. This allows you to issue a command from any directory.

If you have installed the C-SPY debugger, this subdirectory will also contain csa90.exe, the AT90S C-SPY debugger; see the *AT90S C-SPY User Guide*.

C COMPILER FILES

The c:\iar\icca90 subdirectory holds the AT90S C Compiler demonstration files used in the tutorial chapters, as well as various source files for basic I/O library routines.

C INCLUDE FILES

The c:\iar\inc subdirectory holds C include files, such as the header files for the standard C library, as well as specific header files for SFRs.

The C compiler searches for include files in the directory specified by the C_INCLUDE environment variable. If you set this environment variable to the path of the C include subdirectory, as suggested in the installation procedure, you can refer to inc header files simply by their basenames.

LIBRARY FILES

The c:\iar\lib subdirectory holds library modules used by the C compiler.

XLINK searches for library files in the directory specified by the XLINK_DFLTDIR environment variable. If you set this environment variable to the path of the lib subdirectory, you can refer to lib library modules simply by their basenames.

LINKER COMMAND FILES

The c:\iar\icca90 subdirectory holds an example linker command file for each library module.

RUNNING THE C COMPILER

The AT90S C Compiler is run with the following command:

```
icca90 [options] [sourcefile] [options] ↵
```

These items must be separated by one or more spaces or tab characters.

PARAMETERS

options A list of options separated by one or more spaces or tab characters. For a full list of compiler command line options, see *C Compiler Options* in the *AT90S C Compiler Programming Guide*.

sourcefile The name of the source file.

If all the optional parameters are omitted the compiler will display a list of available options a screenful at a time; press ↵ to display the next screenful.

SOURCE FILE

Each invocation of the compiler processes the single source file named on the command line. Its name is of the form:

```
[path]filename.ext
```

For example, the filename `\project\program.c` has the path `\project\`, the filename `program`, and the extension `.c`. If you give no extension in the name, the compiler assumes `.c`. If you omit the path then the current directory is assumed.

INCLUDE FILES

Additional source files may be invoked from the main source file through the `#include` directive. The name of the include file may be specified in one of two ways.

Standard search sequence
To use the standard search sequence enclose the filename in angled brackets, as in:

```
#include <incfile.h>
```

The standard search sequence is as follows:

- The include filename with successive prefixes set with the `-I` option if any.

- The include filename with successive prefixes set in the environment variable named `C_INCLUDE` if present. Multiple prefixes may be specified by separating them with semicolons.

  For example:

  ```
  set C_INCLUDE=\usr\proj\;\headers\ ↵
  ```

- The include filename by itself.

Note that the compiler simply adds each prefix from `-I` or `C_INCLUDE` to the front of the `#include` filename without interpretation. Hence it is necessary to include any final backslash in the prefix.

Source file path
To search for the file prefixed by the source file path first, enclose the filename in double quotes, as in:

```
#include "incfile.h"
```

For example, with a source file named `\project\prog.c`, the compiler would first look for the file `\project\incfile.h`. If this file is not found, the compiler continues with the standard search sequence as if angle brackets had been used.

ASSEMBLY SOURCE FILE

The compiler is capable of generating an assembly source file for assembly using the appropriate IAR Systems Assembler. Its name is *sourcefile*`.s90`.

Assembly source file generation is controlled by the `-a` and `-A` options.

OBJECT FILE

The compiler sends the generated code to the object file whose name defaults to *sourcefile*`.r90`.

If any errors occur during compilation, the object file is deleted. Warnings do not cause the object file to be deleted.

LIST FILE

The compiler can generate a compilation listing, and its name defaults to *sourcefile*`.lst`.

EXTENDED COMMAND LINE FILE

In addition to accepting options and source filenames from the command line, the compiler can accept them from an extended command line file, or from the `QCCA90` environment variable.

By default, extended command line files have the extension `.xcl`, and can be specified using the `-f` command line option. For example, to read the command line options from `extend.xcl` and `extend2.xcl` enter:

```
icca90 -f extend -f extend2 ↵
```

The `QCCA90` environment variable can be set up using the MS-DOS `set` command. For example, typing:

```
set QCCA90=-z9 ↵
```

at the MS-DOS prompt or including this in the `autoexec.bat` file will cause the compiler to optimize for size in all compilations.

DONGLE SECURITY DEVICE

The AT90S C Compiler is supplied with a dongle, or hardware security device, and this needs to be present in order to use the compiler.

Before connecting the dongle you should turn off the PC, or the dongle may be damaged.

Plug the dongle into the parallel printer port on the PC, either LPT1 or LPT2, and tighten the locking screws at each end of the dongle to ensure that it is firmly connected.

If you need to connect a parallel device, such as a printer, to the port, plug this into the socket at the rear of the dongle.

ERROR RETURN CODES

The AT90S C Compiler returns status information to the operating system which can be tested in a batch file. The supported error codes for MS-DOS are listed below.

Code	*Description*
0	Compilation successful.
1	There were warnings.

Code	*Description*
2	There were non-fatal errors.
3	There were fatal errors (compiler aborted).

RUNNING THE ASSEMBLER

The AT90S Assembler is run with the following command:

```
aa90 [options] [sourcefile] [options] ↵
```

These items must be separated by one or more spaces or tab characters.

PARAMETERS

options Assembler options. For detailed information about the assembler command line options see *Assembler options reference* in the *AT90S Assembler, Linker and Librarian Programming Guide*.

sourcefile The source file with the default extension `.msa` or `.s90`.

If all the optional parameters are omitted the assembler will display a list of available options a screenful at a time; press ↵ to display the next screenful.

EXTENDED COMMAND LINE FILE

In addition to accepting options and source filenames from the command line, the assembler can accept them from an extended command line file, or from the `ASMA90` environment variable.

By default, extended command line files have the extension `.xcl`, and can be specified using the `-f` command line option. For example, to read the command line options from `extend.xcl` enter:

```
aa90 -f extend ↵
```

The ASMA90 environment variable can be set up using the MS-DOS `set` command. For example, typing:

```
set ASMA90=-ms ↵
```

at the MS-DOS prompt or including this in the `autoexec.bat` file will cause the assembler to use the small memory model for all assemblies.

ERROR RETURN CODES

When using the AT90S Assembler from within a batch file, you may need to determine whether the assembly was successful in order to decide what step to take next. For this reason the assembler returns the following error return codes:

Return codes	*Meaning*
0	Assembly successful.
1	There were warnings.
2	There were errors.

These codes can be used in conjunction with the `make` facility.

RUNNING XLINK

XLINK is run with the command:

```
xlink [options] objectfiles ↵
```

If no parameters are specified, a list of all the linker options will be displayed.

PARAMETERS

`options` A list of one or more command line options, in any order. For a full list of linker command line options, see *XLINK options reference* in the *AT90S Assembler, Linker, and Librarian Programming Guide*.

`objectfiles` A list of object files. The order in which you specify the object files determines the order in which they are linked.

Upper or lower case is important in the following two cases:

- In linker command line options; eg `-f` and `-F`.
- In symbol and segment names.

EXTENDED COMMAND LINE FILE

As an alternative to supplying options on the command line you can use the `-f` option to read commands from a file.

ERROR RETURN CODES

XLINK returns status information to the operating system which can be tested in a batch file. The supported error codes for MS-DOS are listed below:

Code	*Description*
`0`	Linking successful.
`1`	There were warnings generated during the link unless the XLINK `-w` option was specified, in which case XLINK returns a `0` on warnings.
`2`	There was a non-fatal error.
`3`	Fatal error detected (XLINK aborted).

These error codes can be used as follows in a batch file:

```
.
xlink -f testlnk
if errorlevel 3 goto errors
if errorlevel 1 goto warnings
echo    the link was successful!
.
.
:warnings
echo    the link ended with a warning or non-fatal error
…
.
:errors
echo    the link ended with a fatal error …
.
```

RUNNING XLIB

You can run XLIB in either interactive mode or batch mode.

INTERACTIVE MODE

To run XLIB in interactive mode type:

`xlib` ↵

The librarian will then prompt:

```
*
```

At the XLIB `*` prompt you can enter any of the XLIB commands, including all of the necessary parameters to complete the command, all on one line.

You may also type just the name of the command and XLIB will prompt you for the necessary parameters one at a time. Using this mode of operation, you do not need to remember the order of parameters for each command.

For a list of XLIB commands refer to the *AT90S Assembler, Linker, and Librarian Programming Guide*.

Command syntax
Commands and parameters should be separated by commas, ↵, or spaces, and prompts are issued in the absence of a parameter.

BATCH MODE

To run XLIB in batch mode type:

```
xlib commandfile [p0,p1…p9] ↵
```

where *`commandfile`* specifies a text file containing a sequence of XLIB commands.

Parameters
Optional parameters may be added after the filename and must be separated with spaces or commas. You can set a parameter to its default value by writing `,,`. Within the command file, parameters will be substituted at every occurrence of a `\` (backslash) followed by a decimal number. For example:

```
LIST-MOD \4 PRN
```

If `COMFIL` contains a line like the one above, the command:

```
xlib COMFIL,ABC G,,78,CODE SOME ↵
```

would be interpreted as:

```
LIST-MOD CODE PRN
```

A default value, enclosed in single quotes, may be added immediately after a substitute parameter; for example `'Parm'`. If the requested parameter is missing in the XLIB invocation line or is written as `,,`, the default string will replace the `\`*n* parameter.

An example of a line with a substitute parameter and a default value is:

```
LIST-MOD \4'OBJ' PRN
```

Interactive parameters
Interactive parameters, written `\?`, will make XLIB stop and wait for input. The interactive parameters may also have a user prompt string attached immediately after the question mark.

For example:

```
LIST-MOD \?'YOUR FILE PLEASE: ' PRN
```

The `\?` parameter is replaced with the input, excluding the return.

Note that when XLIB is used in batch mode with a command file, it operates completely silently except when errors occur. This batch operation can be further controlled by the commands `ON-ERROR-EXIT`, `ECHO-INPUT`, and `REMARK`.

TUTORIAL

This tutorial illustrates how you might use the IAR Systems tools to develop a simple project consisting of two C source files and one C header file. It shows how to develop a simple C program, compile it, and run it on the C-SPY debugger.

Before reading this chapter you should:

- Have installed the IAR Systems tools, as described in the chapter *Installation and documentation route map*.
- Be familiar with the architecture and instruction set of the AT90S processor.
- Be familiar with the AT90S C Compiler. For more information refer to the *AT90S C Compiler Programming Guide*.

Note that the pathnames given in this tutorial assume that you have installed the IAR Systems tools in the default installation directory, `c:\iar`.

USING C-SPY

This tutorial assumes that you are using the C-SPY debugger with the IAR Systems tools. If your installation does not include C-SPY you may still follow this tutorial by examining the list files, or by using an alternative debugger. The `.lst` and `.map` files show which areas of memory to monitor.

STARTING A NEW PROJECT

It is a good idea to keep all the files for a particular project in one directory, separate from other projects and the system files.

The tutorial files are installed in the `icca90` directory. Select this directory by entering the command:

```
cd c:\iar\icca90 ↵
```

During this tutorial, you will work in this directory, so that the files you create will reside here.

EDITING A FILE

You can edit the files in a project using any standard text editor.

For example, edit the file demo.c, provided in the icca90 directory, by typing:

```
edit demo.c ↵
```

The contents of this file are shown below:

```
/*    DEMO.C    -   C-SPY Demo Program   */

#include "stdio.h"
#include "defns.h"
void demo_two(int i);

int d;

void main(void)
  {
    int i;

    for (i = 0, d = 1; i < TWO_POWER; i++)
      d *= 2;
    printf("2 to the power of %d is %d\n",
                 TWO_POWER, d);
    demo_two(3);
  }
```

The routine demo_two is defined in the file demo_two.c, and the constant TWO_POWER is defined in the include file defns.h. These files are also provided in the icca90 directory.

Use the editor to introduce an error into the program so that you can see the error handling features provided by the C compiler. Change the i++ at the end of line 11 to j++, and save the file.

COMPILING THE PROJECT

To compile the source file you have edited enter the command:

```
icca90 demo -v1 -A -r -ms ↵
```

The following table explains each of the compiler options used here:

Option	*Description*
`-v1`	Generates code for the AT90S C Compiler with maximum 64 Kbytes data memory and 8 Kbytes program memory.
`-A`	Generates assembler source.
`-r`	Allows the code to be debugged using C-SPY.
`-ms`	Selects the small memory model.

The compiler produces the following output:

```
          IAR AT90S C-Compiler Vx.xx
       (c) Copyright IAR Systems 1996

    for (i = 0, d = 1; i < TWO_POWER; j++)
--------------------------------------^
"demo.c",13  Error[100]: Undeclared identifier: 'j'
Errors: 1
Warnings: none
```

This indicates the line causing the error, and the position in the line.

OTHER ERRORS

If you get the error:

```
Bad command or file name
```

you probably have not got the exe directory, containing the C compiler program, in your MS-DOS `PATH` statement. Type:

```
PATH ↵
```

and check that the path `c:\iar\exe` is included in the path list.

If you get the error:

```
Failed to open #include file 'stdio.h'
```

you have not set up the `C_INCLUDE` environment variable correctly.

Type:

```
set ↵
```

and check that the list of environment variables includes:

```
C_INCLUDE=c:\iar\inc\
```

Note that the path must end in '\'.

CORRECTING THE ERROR

To correct the error edit the source file and correct `j++` to `i++`. Then recompile the program.

It should compile this time without an error to create an object module, `demo.r90`, and an assembler source file, `demo.s90`.

Compile the other source file, `demo_two.c`, in the same way.

LINKING THE PROJECT

Before being able to link the files you have compiled, you need to choose a linker command file to use with the project. This specifies details of the system's memory map and defines the segments to be used for the target code.

For this tutorial we will use the supplied command file `lnk8414s.xcl`, from the `lib` directory. Copy it to your project directory by typing:

```
copy \iar\lib\lnk8414s.xcl ↵
```

This linker command file supports the AT90S processor and the small memory model.

Link the programs by typing:

```
xlink demo demo_two -f lnk8414s -rt -x -l demo.map ↵
```

The following table explains each of the linker options used here:

Option	*Description*
`-f lnk8414s`	Specifies the linker command file `lnk8414s`.
`-rt`	Generates debugging information for C-SPY, and includes the C-SPY terminal I/O routine.
`-x`	Creates a map file.
`-l demo.map`	Specifies the name for the map file.

ERRORS

If you get the error:

```
Unable to open file lnk8414s.xcl
```

you have not copied the supplied linker command file to your project directory.

If you get the error:

```
Unable to open file cl1s.r90
```

you have not set up the `XLINK_DFLTDIR` environment variable correctly. Type:

```
set ↵
```

and check that the list of environment variables includes:

```
XLINK_DFLTDIR=c:\iar\lib\
```

Note that the path must end in ‘\’.

EXAMINING THE MAP FILE

The result of linking is a code file `aout.d90` and a map file `demo.map`.

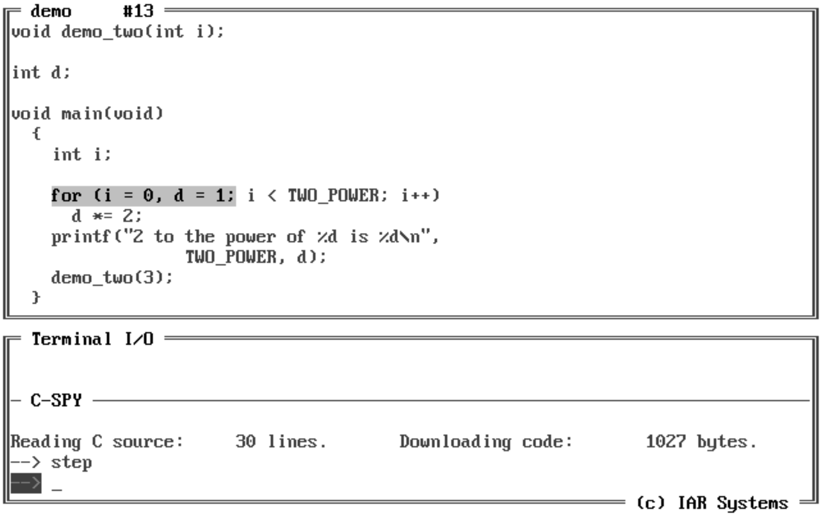
You can examine the map file, using a text editor, to see how the code is mapped to physical addresses.

DEBUGGING THE PROJECT

If you have the C-SPY debugger you can run the object code using C-SPY. Run C-SPY on the object file `aout.d90` by typing:

```
csa90 aout -v1 ↵
```

Then type `STEP` ↵, or press F2, to start executing the code. The source will be displayed on the screen with the first executable statement highlighted:



```
demo     #13
void demo_two(int i);

int d;

void main(void)
  {
    int i;

    for (i = 0, d = 1; i < TWO_POWER; i++)
      d *= 2;
    printf("2 to the power of %d is %d\n",
                 TWO_POWER, d);
    demo_two(3);
  }
```

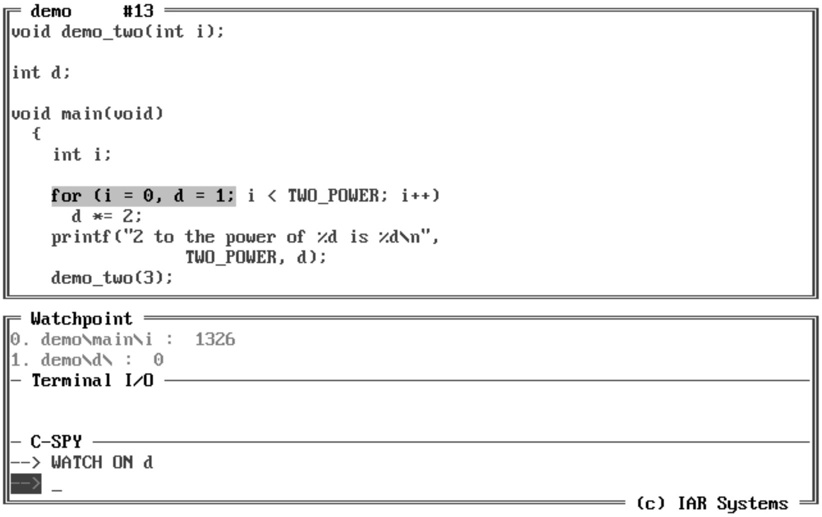
```
Terminal I/O


- C-SPY

Reading C source:     30 lines.       Downloading code:       1027 bytes.
--> step
--> _
                                                              (c) IAR Systems
```

WATCHING VARIABLES

To keep track of a variable you can set a watchpoint on it.

For example, to watch the values of the variables `i` and `d` as you step through the program, type:

```
WATCH ON i ↵
WATCH ON d ↵
```



```
 demo     #13
void demo_two(int i);

int d;

void main(void)
  {
    int i;

    for (i = 0, d = 1; i < TWO_POWER; i++)
      d *= 2;
    printf("2 to the power of %d is %d\n",
                 TWO_POWER, d);
    demo_two(3);

 Watchpoint
0. demo\main\i :  1326
1. demo\d\ :  0
- Terminal I/O


- C-SPY
--> WATCH ON d
--> _
                                                  (c) IAR Systems
```

Now type STEP ↵ again to step through the program and see the variables i and d change in the Watchpoint window.

SETTING A BREAKPOINT

You can execute a program up to a specific statement by setting a breakpoint at that statement.

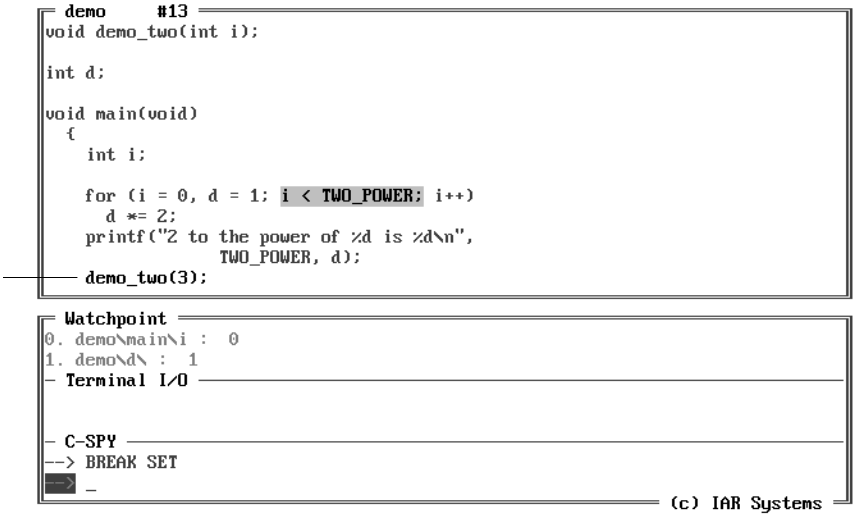
Type Ctrl N, to select the next statement in the program, until the statement:

```
demo_two(3);
```

is highlighted. Then set a breakpoint at this statement by typing:

```
BREAK SET ↵
```

The statement will be displayed in bold to indicate that it is a breakpoint:

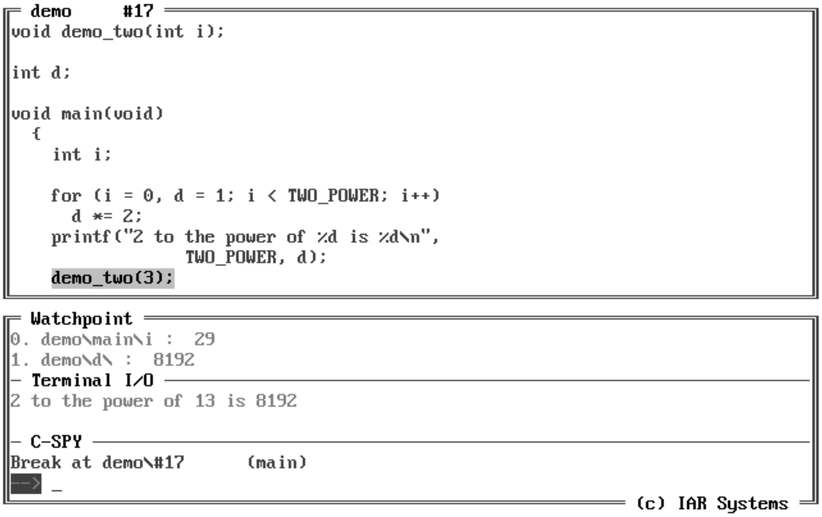


```
demo     #13
void demo_two(int i);

int d;

void main(void)
  {
    int i;

    for (i = 0, d = 1; i < TWO_POWER; i++)
      d *= 2;
    printf("2 to the power of %d is %d\n",
                  TWO_POWER, d);
    demo_two(3);

Watchpoint
0. demo\main\i :  0
1. demo\d\ :  1
- Terminal I/O


- C-SPY
--> BREAK SET
--> _
                                                (c) IAR Systems
```

Breakpoint

Then type GO ↵ to execute up to the breakpoint.

The output from the program will be displayed in the Terminal I/O window:



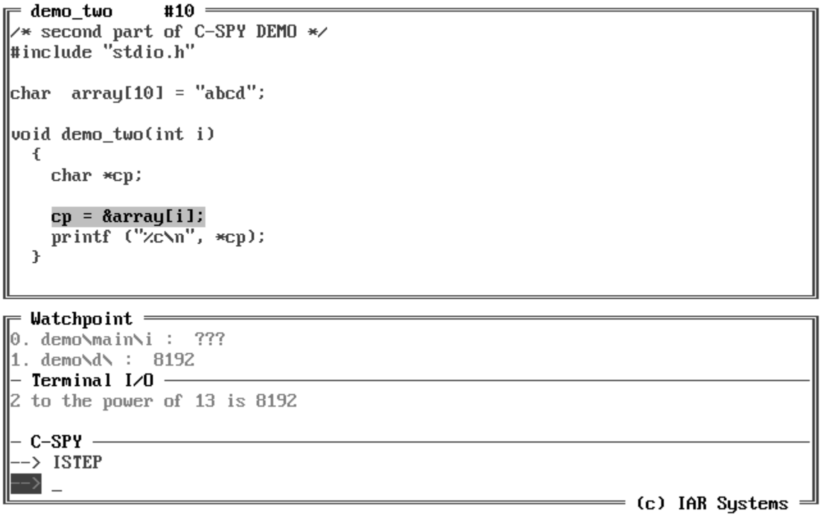
```
demo     #17
void demo_two(int i);

int d;

void main(void)
  {
    int i;

    for (i = 0, d = 1; i < TWO_POWER; i++)
      d *= 2;
    printf("2 to the power of %d is %d\n",
                  TWO_POWER, d);
    demo_two(3);

Watchpoint
0. demo\main\i :  29
1. demo\d\ :  8192
- Terminal I/O
2 to the power of 13 is 8192

- C-SPY
Break at demo\#17      (main)
--> _
                                                (c) IAR Systems
```

Now give the command:

`ISTEP ↵`

to step into the routine `demo_two`.

The source window will then automatically display the second Source file containing the routine `demo_two`:



```
demo_two     #10
/* second part of C-SPY DEMO */
#include "stdio.h"

char  array[10] = "abcd";

void demo_two(int i)
  {
    char *cp;

    cp = &array[i];
    printf ("%c\n", *cp);
  }

Watchpoint
0. demo\main\i :  ???
1. demo\d\ :  8192
- Terminal I/O
2 to the power of 13 is 8192

- C-SPY
--> ISTEP
--> _
                                              (c) IAR Systems
```

If you continue to step you will exit from the program, and the C-SPY window will display `program EXIT`.

To exit from C-SPY, and return to MS-DOS, type:

`QUIT ↵`

and reply `Y` to confirm.

WHAT NEXT?

That completes this brief guided tour of the IAR Systems tools.

For more information about using the IAR Systems tools refer to the *AT90S C Compiler Programming Guide* and the *AT90S Assembler, Linker, and Librarian Programming Guide*.

XLINK ENVIRONMENT VARIABLES

XLINK uses a number of environment variables which can be defined in the PC host environment using the MS-DOS `set` command. These variables can be used to create defaults for various XLINK options so that they do not have to be specified on the command line.

Except for the `XLINK_ENVPAR` and `XLINK_TFILE` environment variables, the default values can be overruled by the corresponding command line option. For example, the `-FMPDS` command line argument will supersede the default format selected with the `XLINK_FORMAT` environment variable.

To make these settings automatic, you can place the `set` commands in your system's `autoexec.bat` file (or in your login script if you are running on a network).

XLINK_COLUMNS

Sets the number of columns per line.

DESCRIPTION

Use `XLINK_COLUMNS` to set the number of columns in the listing. The default is 80 columns.

EXAMPLE

To set the number of columns to 132:

```
set XLINK_COLUMNS=132 ↵
```

XLINK_CPU

Sets the target CPU type.

DESCRIPTION

Use `XLINK_CPU` to set a default for the `-c` option so that it does not have to be specified on the command line.

EXAMPLE

To set the target CPU type to AT90S:

```
set XLINK_CPU=a90 ↵
```

RELATED COMMANDS

This is equivalent to the XLINK `-c` command; see *-c* in the *AT90S Assembler, Linker, and Librarian Programming Guide*.

XLINK_DFLTDIR

Sets a path to a default directory for object files.

DESCRIPTION

Use `XLINK_DFLTDIR` to specify a path for object files. The specified path, which should end with `\`, is prefixed to the object filename.

EXAMPLE

To specify the path for object files as `c:\iar\lib:`

```
set XLINK_DFLTDIR=c:\iar\lib\ ↵
```

XLINK_ENVPAR

Creates a default XLINK command line.

DESCRIPTION

Use `XLINK_ENVPAR` to specify XLINK commands that you want to execute each time you run XLINK.

EXAMPLE

To create a default XLINK command line:

```
set XLINK_ENVPAR=-FMOTOROLA ↵
```

RELATED COMMANDS

See also *-f* in the *AT90S Assembler, Linker, and Librarian Programming Guide*, which reads linker commands from a file.

XLINK_FORMAT

Sets the output format.

DESCRIPTION

Use `XLINK_FORMAT` to set the format for linker output. For a list of the available output formats see *XLINK output formats* in the *AT90S Assembler, Linker, and Librarian Programming Guide*.

EXAMPLE

To set the output format to `MOTOROLA`:

```
set XLINK_FORMAT=MOTOROLA ↵
```

RELATED COMMANDS

This is equivalent to the XLINK `-F` command; see *-F* in the *AT90S Assembler, Linker, and Librarian Programming Guide*.

XLINK_MEMORY

Specifies whether XLINK is file-bound or memory-bound.

DESCRIPTION

If set to 0 the linker is file-bound; otherwise it is memory-bound.

EXAMPLE

To specify that XLINK is file-bound:

```
set XLINK_MEMORY=0 ↵
```

RELATED COMMANDS

This is equivalent to the XLINK `-m` command; see *-m* in the *AT90S Assembler, Linker, and Librarian Programming Guide*.

XLINK_PAGE

Sets the number of lines per page.

DESCRIPTION

Use `XLINK_PAGE` to set the number of lines per page (20–150). The default is a listing with no page breaks.

EXAMPLES

To set the number of lines per page to 64:

```
set XLINK_PAGE=64 ↵
```

RELATED COMMANDS

This is equivalent to the XLINK `-p` command; see *-p* in the *AT90S Assembler, Linker, and Librarian Programming Guide*.

XLINK_TFILE

Specifies the temporary file.

DESCRIPTION

Use `XLINK_TFILE` to set the name and location of the temporary file which is used when the XLINK `-t` command is specified; see *-t* in the *AT90S Assembler, Linker, and Librarian Programming Guide*.

EXAMPLE

To specify a temporary file `e:\xlink.tmp`:

```
set XLINK_TFILE=e:\xlink.tmp ↵
```

XLIB ENVIRONMENT VARIABLES

XLIB supports a number of environment variables which can be set using the MS-DOS `set` command. These variables can be used to create defaults for various XLIB options so they do not have to be specified on the command line.

To make these settings automatic, you can place the `set` commands in your system's `autoexec.bat` file (or in your login script if you are running on a network).

XLIB_COLUMNS

Sets the number of columns.

DESCRIPTION

Use `XLIB_COLUMNS` to set the number of columns for listings (80–132). The default is 80 columns.

EXAMPLE

To set the number of columns to 132:

```
set XLIB_COLUMNS=132 ↵
```

XLIB_CPU

Sets the CPU type.

DESCRIPTION

Use `XLIB_CPU` to set the CPU type so that the `DEFINE-CPU` command does not need to be entered at the beginning of an XLIB session.

EXAMPLE

To set the CPU type to AT90S:

```
set XLIB_CPU=a90 ↵
```

XLIB_PAGE

Sets the number of lines per page.

DESCRIPTION

Use `XLIB_PAGE` to set the number of lines per page (10–100) for the list file. The default is a listing with no page breaks.

EXAMPLE

To set the number of lines per page to 66:

```
set XLIB_PAGE=66 ↵
```

XLIB_SCROLL_BREAK

Sets the scroll pause.

DESCRIPTION

Use `XLIB_SCROLL_BREAK` to make the XLIB output pause and wait for the ↵ key to be pressed after the specified number of lines (16–100) on the screen have scrolled by.

EXAMPLE

To pause every 22 lines:

```
set XLIB_SCROLL_BREAK=22 ↵
```