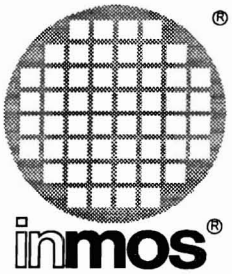

IMS D4314A Sun 4
IMS D6314A VAX/VMS
IMS D7314A IBM–PC
ANSI C Toolset
Release Notes

INMOS Limited

SGS-THOMSON MICROELECTRONICS
INMOS is a member of the SGS–THOMSON Microelectronics Group

72 TDS 349 02 January 1993

The C compiler implementation was developed from the Perihelion Software "C" Compiler and the Codemist Norcroft "C" Compiler.

INMOS Document Number: 72 TDS 349 02

Contents

1 Release notes

1.1 Introduction

This version of the INMOS ANSI–C toolset (D4314A, D6314A, D7314A) contains some significant enhancements over the earlier ones (D4214B, D5214B, D6214A, D7214C).

This document describes the major differences, provides general installation information and lists the known errors. These notes are intended as a guide — the manuals define the functionality in detail.

1.1.1 Licenses

Some variants (e.g. D4314A) of this product are supplied with a license manager. Please read your delivery manual before attempting to install and use the software.

1.1.2 Notation

For ease of reading, all options are given in the form `-x` as they would be written for Unix systems. For MS–DOS and VAX/VMS, the corresponding option would be written as `/x`.

1.2 Hosts

The toolset is supplied in three variants:

- **D4314A** Sun 4 running SunOS 4.1.1 or later.
- **D6314A** VAX under VMS.
- **D7314A** for IBM–PC 386 compatibles with a minimum of 4M bytes running DOS 5.

The Advanced Toolset (Dx300) is designed to operate in conjunction with the Dx314 toolset.

Note: This and later versions of the toolset do not support either the Sun 3 or the PC–AT 286. However, the D5214 and D7214 will continue to be available for some time. Please contact your distributor for further details.

1.3 `iserver` — important changes

The version of the host server with this release is a considerable change over the previous version.

There is just one program per toolset and you no longer need to decide which one to use. To handle the different possible methods of accessing transputers, it now obtains information from a connection database. This must exist and its name provided in the environment variable `ICONDB`. Examples of such a database are supplied under the `connect` subdirectory and you should edit a suitable one for your installation.

The `TRANSPUTER` environment variable has changed its meaning, and its value must be changed to reflect the entry in the connection database that you wish to use by default.

Please see the chapter on the `iserver` in the Tools Reference Manual.

1.4 Language extensions and changes

The language supported continues to be ANSI–C, and the products will be validated in due course. This release successfully passes all the tests in the Plum–Hall validation suite version 3.

1.4.1 INMOS extensions

A command line option `FC` to the compiler causes it to default the type `char` to `signed char`. It is otherwise `unsigned` as in earlier releases.

1.4.2 Changes from previous version

The definition of type `Channel` is changed to provide the optimizer with better information. If a program restricts its use of channels to the documented uses of the header files, then the program will continue to work.

1.5 Optimization

There is a globally optimizing compiler provided in this toolset. It generates code for any 32–bit transputer, but includes neither 16–bit support nor debugging information.

The optimizing version of the compiler also passes all the Plum–Hall validation suite version 3.

The options are fully compatible with the standard compiler, but any options that are not supported are either warned about or ignored.

The compiler can be requested to concentrate on speed or space. It applies a transformation to a program only if it is known that the transformation will result in an increase in speed or a saving in space as appropriate. The default is to optimize for speed.

There are three levels of optimization. Level zero is the base level. Level one adds the local optimizations. Level two includes the global optimizations. The default is level one.

Local optimizations use information obtained only from the local environment of the source construct being transformed. No information is gathered from other parts of the program.

Global optimizations are based on all the source code within a C function and all known information about static variables. This type of optimization can move expressions and statements from one place to another.

For a full description of the types of optimizations performed, please see the *Optimizing Compiler User Guide* supplied.

A new pragma, `IMS_nosideeffects`, has been introduced to provide the optimizer with information about functions. This pragma indicates whether a function can have any side–effects on static or external variables. This enables the optimizer to determine whether a call to a function can change the values of other variables, and if not, make assumptions about the validity of previously computed expressions.

1.6 Virtual routing configurer

The configurer in this toolset contains the functionality of the virtual routing configurer as provided in the product Dx224A. The configurer can provide support for channels connected to processors that are not adjacent in the network of transputers. In this case, a small kernel is added to some of the processors and the connections are multiplexed with others from one processor to another. This is known as *'software through–routing'*.

If the user specifies all the placements of connections, then the configuration will be the same as before and there will be no extra processes created for support of software through routing.

There are facilities to control the number of links which are used for the through routing and hence the user can optimize the network by judicious placement of the processes.

The debugger `idebug` has been modified to make it easier to debug programs that have been configured with the virtual routing software than was the case with Dx224. However, there has been an increase in the size (about 10%) of the kernel that `idebug` loads during interactive debugging.

1.7 Embedded systems help

1.7.1 Memory map file

A new tool `imap` is provided which can produce a complete record of where all variables and functions reside in memory on every node of a network. It operates by

reading the map files produced by the compiler, linker and collector and computing the locations of all the known names.

It produces a list for each processor, the contents of the memory in increasing order of address, stating what the sections of memory are for. This is first at the gross level of main programs, then for each module within each program.

Then it lists all the known names in alphabetical order showing where they reside on each node. Optionally, the names which are internal to the C runtime library can be suppressed, in order to reduce the size of the listings.

The listings from the compiler and linker have been extended in order to supply **imap** with the information it needs. In addition, the workspace requirements of each function are given, as are the positions of automatic variables within the workspace.

1.7.2 Placing code and data

The configurer has new features to enable the user to place code and data areas at specified locations in memory. It can also be told not to use the on–chip memory and leave it free for the application.

A new attribute is added to a processor, **reserved** = *size*. This reserves *size* bytes from the lowest address and neither any part of the application, nor startup processes will be placed in the reserved area automatically. Hence, an application can ensure that our tools do not use on–chip memory, or any other contiguous area starting at *mint*.

A new attribute is added to a process, **location**. This has the sub-attributes **stack**, **code**, **static**, **heap** and **vector**. They specify the exact location, by address, of each of the named sections. It takes precedence over the **order** attribute which retains its meaning.

See the chapter on Configuration in the *Toolset User Guide* for further details.

1.7.3 Instruction access

A number of predefined functions are introduced to allow access to some of the transputer features in a more efficient fashion.

BlockMove like the **move** instruction.

BitCnt, **BitCntSum** map onto **bitcnt**.

BitRevNBits, **BitRevWord** reverse bits.

Move2D, **Move2DNonZero**, **Move2DZero** use the transputer's instructions for moving data in blocks.

`CrcByte`, `CrcWord` perform CRC checking.

Channel instructions The functions:

```
DirectChanIn
DirectChanInChar
DirectChanInInt
DirectChanOut
DirectChanOutChar
DirectChanOutInt
```

provide a way of accessing the channel communications instructions directly. However, when these are used on external links, there is no support for the software through routing and the debugging support is reduced.

`ProcReschedule`, `ProcGetPriority`, `ProcTime` are done inline.

The above functions are handled by a library call where the corresponding instructions are not present.

1.7.4 Reducing library entry overhead

In order to provide flexibility to the user to tailor the runtime system to a particular application, the source code of the startup routines are provided. These are fully commented and can be changed by the user to include only the functions that are actually required. Changes have been made to the compiler in order that the user can write all the code required in C. There is an assembler routine still present because it requires access to a linker function that is not possible through a high level language. However, this is encapsulated in a single module that is included where necessary.

In this way, a function that does not use any static (which implies it does not use the C I/O system, or any heap) can be started with about 40 bytes of overhead. By writing the application as if it were the startup code, the overhead is reduced to zero.

(**Note:** The new startup system works only when the program has been configured. It does not support the `-t` option on the collector.)

1.7.5 Other space saving

If a variable is defined as a `static const`, then it is placed in the text section if possible. This can save both initialization time and static space, especially where large tables are concerned.

On 16–bit processors, the inline code sequence for signed right shift on operands of type `long int` has been put into a library function to save space in the code.

1.7.6 Bootstrap sources

The sources of all the bootstraps and network loaders are provided with the toolset. These are commented in full, so that the user can modify them if necessary.

They can be found under the `source` subdirectory in this release.

It is recommended that any changes be made to the network loader and not to the bootstraps. There are interactions between the bootstraps that must be kept consistent, but the network loader can be changed and the information about it extracted from the resulting object module.

1.7.7 Dynamic loading of processes

A new set of library functions is provided that enables an application to load and execute a process that has been separately compiled and linked. The loaded process is created as an `.rsc` file, using the collector. Functions are provided that read the `.rsc` file and extract crucial information about the process, such as the size of static required and the location of the entry point. The application can then allocate the space required, load the file and call it.

Similar functions are provided to access an `.rsc` file that has been placed into ROM or is read from a channel.

By adapting the new dynamic code loading harness, the user can also tailor the loaded code to accept any parameters that are desired, rather than using only the ones which are supplied in this toolset.

1.7.8 Assembler

The assembly language that the compiler uses in its final phase has been made available to the users in a supported form. The language is documented in the reference manuals. It can be used to prepare bootstraps or other special purpose functions.

The assembler is invoked by a command line option on the compiler. If any preprocessing of assembly source is required, then two invocations of `icc` should be used, once to execute the preprocessor, the second to execute the assembler.

1.8 Sundry

When executing a program that has been configured, the function `exit` now terminates the server by default. The function `exit_terminate` still exists for compatibility, but behaves just like `exit`. If a configured process does not wish to terminate the server when it terminates, a new function has been introduced, `exit_noterminate`.

This release includes a set of new linker indirect files which replace and augment `startup.lnk` and `startrd.lnk`. The new linker indirect files are as follows:

- `cstartup.lnk` This file is used when linking with the full libraries for a configured program. It replaces `startup.lnk` but has slightly different `exit` semantics (see comments on `exit` above).

  In the Dx214 toolsets, the runtime start up code supplied two extra parameters to the function `main`. These were arrays of channels found in the configuration interface description. This facility is now available only through the use of the linker indirect file `startup.lnk`.

- `cnonconf.lnk` This file is used when linking with the full libraries for a non–configured program (i.e. using `icollect -t`). It replaces `startup.lnk`.

- `cstartrd.lnk` This file is used when linking with the reduced libraries for a configured program. It replaces `startrd.lnk`.

- `clibs.lnk` This file lists all library files making up the full library. It is useful in mixed language programming and when using the dynamic loading support provided in the library. There was no equivalent in the previous toolset.

- `clibsrd.lnk` As for `clibs.lnk` but for the reduced library.

`startup.lnk` and `startrd.lnk` are still present but their use is deprecated. They will not be supplied in future toolsets and the user is strongly advised to move over to using the new linker indirect files as soon as possible. For reasons of compatibility, when `startup.lnk` is used, `exit` behaves in the same manner as in the Dx214 toolsets.

The optimizing compiler does not warn by default about formal parameters that are not used. The warning is issued when the `-fh` option is used.

The stack of a process created by `ProcInit` can now reside anywhere within the transputers address space as long as it does not nest within an existing process stack. In Dx214, it had to be allocated from the heap.

New functions `ProcJoin` and `ProcJoinList` allow a process to wait for an asynchronous process to terminate before continuing.

A new version of the `callc.lib` library is issued which is compatible with this release of the C runtime library. It should be used instead of the one released with the occam toolset, Dx205.

The debugger `idebug` may be used with programs which use software virtual routing. A new option has been introduced to display virtual links on a processor.

`idebug` can now debug boot–from–ROM, run in RAM programs.

The T426 is supported by all the tools.

A number of bugs are fixed. The most important ones are:

TS/1275, TS/1322 Line counts corrected when form feeds and vertical tabs are present in the source.

TS/1370 Signed integer arithmetic now follows exactly the order that the program defines; add and multiply are not associative.

TS/593, TS/1337 Pragma nolink problems resolved.

TS/1449, TS/1450 clock function clarification for high priority and 16–bit processors.

1.9 Cautions

When receiving two arrays of channels at the end of the standard parameters to `main`, the linker indirect file `startup.lnk` must be used. This method is deprecated and may disappear in the future.

There is a change in the default manner of booting programs from link. The Dx214 toolset used whatever space was available from low memory. Dx314 loads its bootstraps and loader into high memory, as defined by the configuration language 'memory' attribute. To retain the Dx214 behaviour, a new option `-bm` has been added to the collector.

When collecting with the `-t` option, the value of IBOARDSIZE must not be greater than the actual memory present on the hardware. When collecting with the `-t` and `-m` options, the value of the `-m` option must not be greater than the actual amount of memory present.

When mixing languages, C and occam, using Dx314 and Dx205, virtual channels should NOT be used. Post-mortem debugging is still possible for mixed language programs but interactive debugging cannot be used in general. A new version of the occam toolset (Dx305) will be released shortly which will be fully compatible with this version of the C toolset.

There are two hosts that are discontinued with this release, viz. the PC–286 (AT compatible plus TRAM) and the Sun 3. The tools in Dx713 execute directly on the PC rather than on a transputer attached to the PC.

The option `-t` on the collector is strongly deprecated. The advanced toolset (Dx300) will not support programs that have been generated using it. To a large extent, there are alternative ways of getting equivalent functionality in this release. Any shortcomings will be remedied in a later version of the toolset.

The optimizing compiler does not perform certain types of optimization when a program uses the process library. In particular, it cannot perform some workspace allocation optimizations nor tail call elimination.

On MS–DOS, the previous toolsets used a screen driver called `BANSI.SYS`. This has been removed, along with its accompanying `PCBANSI.ITM`, since it is incompatible with later versions of both DR–DOS and MS–DOS.

The functions that were provided for compatibility with 3L have been removed. We no longer support the header file `conndxll.h`.

The tools `icvlink` and `icvemit` have been removed from the toolset.

1.10 Bugs fixed

The following bugs in previous version of the toolset are now fixed.

- (Ref: Bulletin 373) If a function invocation uses workspace for some of its parameters and one of the parameters contains a bitwise 'or' operator and one of the operands to the bitwise 'or' is a function invocation which passes some of its parameters in workspace, then incorrect code was generated. This is now fixed.
- (Ref: Bulletin 374) The function `ProcPriPar` on a 16–bit processor now executes correctly. The need to align the workspace to a 4–byte boundary has been removed. It must still be aligned to a word boundary on all processors.
- (Ref: Bulletin 379) The function `ProcPriPar` now correctly returns at the caller's priority.
- (Ref: Bulletin 386) The configurer now takes into account 3 words below workspace for scheduling purposes, hence removing the potential problem when overlaying part of the configuration processes in the user's stack space.
- (Ref: Bulletin 395) When calling C from occam, it is no longer necessary to initialize `_IMS_heap_base` and `_IMS_heap_front`. That is now done by the `callc` library. In fact, these variables MUST NOT be initialized by the user.
- (Ref: TS/1054) When debugging a C program with `idebug`, access to the pre–defined variable `errno` had to be made using the name `_IMS_errno`. This is no longer true. The name `errno` is now known to the debugger.
- (Ref: TS/1697) If two processes concurrently operated with floating point values on a T8 processor and one of those processes performed a condition expression, then it could have received a corrupted result to the condition operation. This was true whether the processes were part of the same program, or were separate processes at the configuration level. The reason was that during a conditional expression, the floating point stack was not emptied before the jumps, and so another process could have corrupted the stack if there was a timeslice de–schedule on the jump.

  Examples of the type of code that were at risk were:

```
double df(double x)
{
  return (x < 0.0) ? -x : x;
}
```

and

```
float q(int c, float d)
{
  return (c ? -d : d) + ((d * d) + (d * d));
}
```

This has now been fixed.

- (Ref: TS/1706) In a program with more than one incomplete declaration of an **enum** type before its completion, the enumeration values and type information were not accessible by the debugger. This was due to a bug in the C compiler and has now been fixed.

1.11 Installation

1.11.1 If you are replacing an existing ANSI–C toolset

This toolset completely replaces the previous ANSI–C toolsets D4214B, D6214A and D7214C. The installation of this toolset will generate a new directory and sub-directories. The older one should be deleted.

If you have made any changes to the previous toolset structure, these should be recorded and re–implemented in the new structure.

1.11.2 If you are using occam and ANSI–C together

The occam and ANSI–C toolsets have some common components, such as the linker, librarian, debugger, eprom tools, and the system library **sysproc.lib**. The installation procedures for each toolset create separate directory structures. However, if you are intending to mix languages, it is important to know which version of the tools to use.

When choosing a tool, *always use the latest version.* Tools are designed to be *backwards compatible* so that they will work with previous releases of the toolsets. A tool will not necessarily work in conjunction with tools from a later toolset, especially if there is a later version of the tool in the later toolset.

Consult the documentation on the Release Notes for each toolset to determine which is the later release. If you wish to use both toolsets, put the *later* release on your search path *before* the earlier one. If you decide to combine the tools into a common directory, then, for the tools and libraries which are in both toolsets, always use the later one.

1.11.3 C++ considerations

If you are using the INMOS C++ product, Dx217, then by default, when running the linker on the users behalf, **iccxx** refers to **startup.lnk**. This may be over–rid-

den by the presence of a `.ilk` file (specified by the user) in order to obtain the newly introduced replacement `cstartup.lnk`.

1.11.4 Transputer bootable files

This toolset does not contain any of the tools as transputer bootables except for `idebug`, `iskip` and `idump`. These three need a transputer to execute.

None of the other tools are provided as transputer executable files. They execute on the host.

For an IBM PC compatible host, D7214 relied on a transputer to execute the compiler. This is not true in D7314; the requirements of the PC are now that it has an Intel 386 or 486 compatible processor, runs DOS 5.0 or later, and has at least 4 MB of memory.

1.12 Errata

This release has the following known errors.

1.12.1 C compiler — `icc`

Unless where otherwise stated, the problems described here apply to both the optimizing and non–optimizing compilers.

- (Ref: TS/1164) The compiler may generate an internal error and terminate compilation when attempting to compile an invalid program with the wrong nesting level of braces in a static initialization.

  No problems are known when the syntax of the initializer is corrected.

- (Ref: TS/1303) If a label has the same identifier as a type name, then the compiler generates invalid error messages and does not complete the compilation. **Workaround:** Change the name of the label or the type name so that they are different.

- (Ref: TS/1353) When an include preprocessor statement names a file using the Unix directory syntax (i.e. with / as a separator), but the program is being compiled on VAX/VMS, then the filename may not be correctly adjusted when searching ISEARCH. The compiler will report that the file cannot be found. **Workaround:** Change the source so that the file name uses VAX/VMS syntax for subdirectories.

- (Ref: TS/1406) The compiler will abort compilation with an internal error in some cases where spurious casts are used in expressions. For example:

  ```
  int array[2];
  (void) &(int *)array;
  ```

  Workaround: Remove the unnecessary casts.

- (Ref: TS/1429) If the **size** option is used on an assembler instruction in a **__asm** block with a non–constant expression, bad code is generated. No diagnostic is produced. For example:

```
int x;
__asm {
size x ldl 1; /* size expression (x) should be constant */
      }
```

  does not indicate an error, but should do.

- (Ref: TS/1542, OPT/163) If a union tag is used in a definition of an incomplete type array with initializer and the tag has not been defined, then the compiler may abort with a segmentation fault or other system error. For example:

```
union { int i;              /* tag omitted in declaration */
        char *j;
      };

union u array[] = { 13,     /* 'u' not defined */
                    27,
                    -9,
                    43,
                  };
```

- (Ref: TS/1554) If the primary file name given to the compiler contains a hyphen symbol, the compiler does not process the command line correctly. **Workaround**: rename any C source files not to include hyphens.

- (Ref: TS/1613) Spurious messages can be generated for preprocessor directives that occur within the non–included clauses of preprocessor **#if** constructs. These are rare and usually apply to invalid or dubious conditions should they have arisen in normal text anyway. They can be safely ignored. This error only occurs with the non–optimizing compiler.

- (Ref: TS/1630) The assembler may not diagnose some cases of errors arising from the evaluation of expressions (e.g. overflow). Exceptions can never happen when the assembler is invoked during the compilation of a C program. It occurs only when the compiler is invoked to do an assembly using the **–as** command line option.

- (Ref: TS/1637) If a variable is first declared with file scope and internal linkage (i.e. is "static"), and then subsequently declared with file scope and external linkage (i.e. with "extern") and given an initializer on the second declaration, then the name is given external linkage, rather than internal linkage according to the ANSI standard.

  For example:

```
static const int a[5];
extern const int a[5]    = { 1, 2, 3, 4, 5 };
```

The identifier `a` is given external linkage (i.e. it is visible to the linker), whereas it should have internal linkage (i.e. should not be visible to the linker).

- (Ref: TS/1640) If an incomplete structure declaration occurs in a prototype for a function pointer declaration as a member in a structure declaration, then an attempt to declare another structure with the same tag name may cause the compiler to abort with a core dump or other system error. Note that the scope of such an incomplete declaration is the prototype only; it cannot be completed by any further structure declaration. In addition, no function can be declared that matches the function pointer prototype, since the incomplete structure declarations are not equal.

- (Ref: TS/1665) The hash '#' preprocessing operator does not add back-slash characters into the substituted string as it should according to the standard definition. There is no work around.

  For example:

```
#define X(a)   #a

if ( strcmp( X( '\'' ), "'\\''" ) == 0 )
```

  generates:

```
if ( strcmp(  "'\''" , "'\\''" ) == 0 )
```

  instead of:

```
if ( strcmp(  "'\\''" , "'\\''" ) == 0 )
```

- (Ref: TS/1666) If any errors are found during the assembly phase of the compiler (including if it is used alone through the `-as` command line option), then the object file is not removed.

- (Ref: TS/1725) If a `#pragma IMS_translate` is used to create the same alias name for more than one function, then the compiler will produce an error during the assembler phase concerning duplicate names. This can occur either if the alias is to the name of a function without a translation alias, or if two functions are aliased to the same name.

- (Ref: TS/1727) When preprocessing source only (i.e. using the `-pp` option), some pragmas may produce warning messages if they refer to elements of the C language source. These messages should be ignored.

  For example, using `icc -pp` on the following source:

```
void p(void);
#pragma IMS_nolink(p)
```

  will generate a warning message.

- (Ref: TS/1728) There is a restriction on the value of the `size` option on a `patch` directive in the assembler. The value must be positive and not greater than 255. This is not documented, nor is it checked. If the value is outside this range, the behavior of the assembler is undefined.

- (Ref: TS/1752) If an attempt is made to define a structure type containing a member which is an array whose elements are of a type which is incomplete, then the compiler will silently accept it. It should not do so, as this is an error. Note that an error message is generated if the debugging flag `-g` is used. An example of the case is:

```
typedef struct {
        struct PROV Full [1];
        } JUNK;
```

  where the type of `struct PROV` has not been completed yet.

- (Ref: TS/1864) The optimizing compiler may give a fatal internal error when generating code to assign to an element of a typedef type which has an underlying array type.

  For example, the following fails.

```
typedef struct mem_structure_s
  {
      int type;
  } mem_structure_t[5];

static void  compute_structure_data( void )
{
  mem_structure_t  *struc;      /* !! */
  int               i;

    i = 4;
    (*struc)[i].type          = 2;
}
```

  Workaround. The declaration that causes the trouble is the one marked with exclamation points. It can be replaced by an explicit version of the type, rather than using the typedef name. For example, the line above can be replaced by:

```
  struct mem_structure_s (*struc)[5];
```

- (Ref: OPT/142) If the optimizer has re–arranged the source so that expressions are executed at some other line number, then any messages that the expression may cause at compile time (e.g. a divide by zero exception) may be given more than once and refer to the statement to which it has moved. This applies only to the optimizing compiler.

1.12.2 Debugger — `idebug`

- (Ref: TS/1477) The object files for certain types of processor are identical to those of others, and the debugger cannot distinguish them. If the pro-

gram is configured, there is no problem. If you *must* use the non–configured case (i.e. `-t` option to the collector), then do the following for debugging:

For T222 there is no problem. The debugger can treat T222 as a T212, which is what the object file is marked as.

For T400, T426 Debug in TA or TB mode and commit to T400 or T426 when debugged.

For T801 Debug in TA mode and commit to T801 when debugged.

- (Ref: TS/1862) Low priority user processes using software virtual links are promoted temporarily to a high priority process as they perform the communication operation. If you locate to such a process via the idebug Monitor page Display software virtual links option, idebug cannot tell if they were originally at high or at low priority: idebug will always locate to what it believes is a high priority process. For further details consult the idebug reference chapter Monitor page 'Z' option.

1.12.3 Collector — `icollect`

- (Ref: TS/2052) When building boot-from-ROM applications the collector `icollect`, by default, uses an overlaying network loader. It is possible that this may cause a conflict of memory usage, which causes the application to fail.

  Workaround. Select the non-overlaying ROM network loader by using the '`bm`' option on the collector.

1.12.4 File lister — `ilist`

- (Ref: TS/1477) The file lister cannot distinguish between some processor types when listing some file formats. In these cases, the T400 and T426 processors are shown as a T425, the T801 is shown as a T805 and the T222 as a T212. The code the compilers generate is identical for these alternative processor types and so the object files do not mark any further differences.

1.12.5 Makefile generator — `imakef`

- (Ref: TS/1544) The makefile generator may produce a core dump or other system error when building a makefile for a `.liu` file when the `.lbb` file is missing. This affects occam users. Libraries of C modules do not have `.liu` files created automatically.

- (Ref: TS/1572 and TS/1822) The makefile generator and the C compiler differ in their use of nested directories when handling include files. This is

part of a more general problem that will not be fixed until a subsequent release. **Workaround**: Do not use `imakef` for such C programs, or at least use it once to obtain a skeleton makefile which is then edited to get it right. The makefile should then be maintained by hand.

- (Ref: TS/1687) When references are made from `.pgm` files to files on remote file systems, then the names generated for those files in makefiles and linker indirect files may be incorrect. There is no work around, but it applies only to occam programs.

1.12.6 File server — `iserver`

- (Ref: IS/53) When a request is made for an environment variable, whose content is longer than 507 bytes, the server attempts to send more bytes in the return message than the runtime systems can accept. Typically, the error flag on the transputer will be set.

  The maximum length of an environment variable should be 507 bytes.