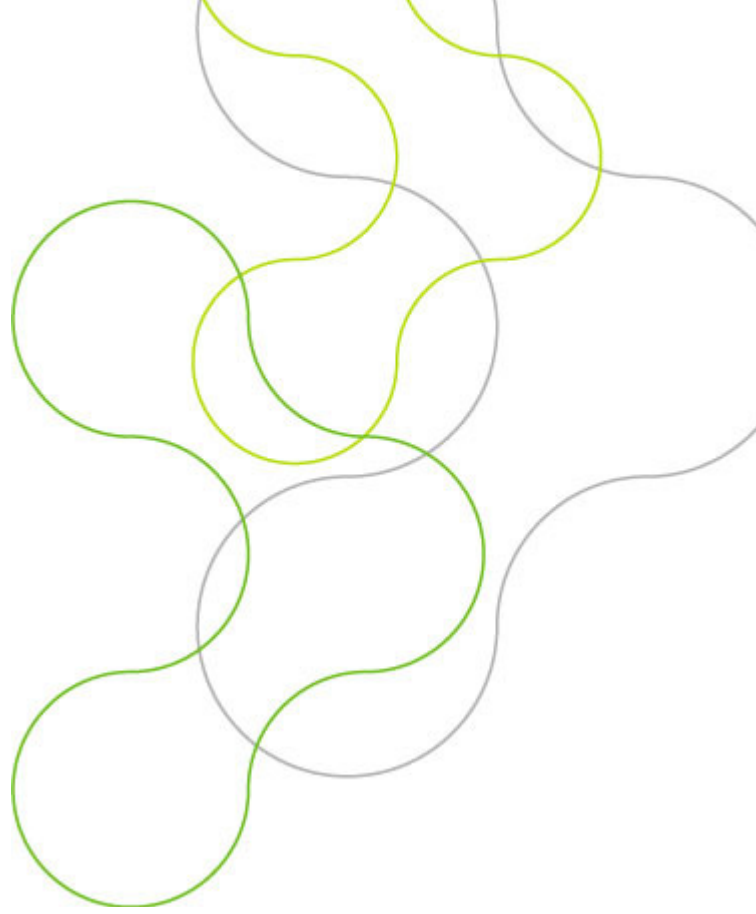




Frequently Asked Questions



Version 1.9
19 February 2009

Copyright Notice

Copyright 1999-2009 Gresham Software Labs Pty. Limited

Copyright 1996-2005 Insession Pty. Limited

Copyright 2008 ACI Worldwide (Pacific) Pty. Limited

All rights reserved. Information in this document is subject to change without notice. You are permitted to reproduce, transmit, and distribute (but not translate) this document, in its entirety, without alteration, providing that this complete Copyright Notice is included with all copies.

Trademarks

TAL2C™ is a trademark of ACI Worldwide (Pacific) Pty Limited. Compaq, Expand, Guardian, Himalaya, NonStop, ServerNet, SNAX, and Tandem are trademarks of Hewlett-Packard Company. Microsoft is a registered trademark, and MS-DOS, Windows, and Windows NT are trademarks, of Microsoft Corporation. Intel and Pentium are trademarks of Intel Corporation. IBM and IBM PC are registered trademarks of IBM Corporation. PKWARE and PKZIP are registered trademarks of PKWARE, Inc. Unix is a registered trademark, and X/Open is a trademark, of X/Open Company Limited.

All other brand names and product names may be trademarks or registered trademarks of their respective companies.

Acknowledgements

TAL2C is based in part on Sather libraries distributed free of charge by the International Computer Science Institute, 1947 Center Street, Suite 600, Berkeley, CA 94706 and which may be obtained by anonymous ftp from icsi.berkeley.edu.

The authors of the Sather library code used in TAL2C are:

Benedict A. Gomes
Holger Klawitter
Stephen M. Omohundro

TABLE OF CONTENTS

1	INTRODUCTION	5
1.1	Purpose of this document.....	5
1.2	How to Contact Gresham	5
2	GENERAL QUESTIONS	6
2.1	What is TAL2C?.....	6
2.2	How does TAL2C work?.....	6
2.3	How fast is TAL2C?	6
2.4	Where can I obtain TAL2C?.....	6
2.5	Are there regular TAL2C upgrades?	6
2.6	What documentation is available for TAL2C?.....	6
2.7	What levels of support are provided for TAL2C and by whom?.....	6
2.8	What types of licenses are available for TAL2C?.....	7
2.9	How much does TAL2C cost?.....	7
2.10	Can you provide references from customers using TAL2C?	7
3	HARDWARE AND SOFTWARE CONSIDERATIONS	8
3.1	What is the recommended hardware for running TAL2C?	8
3.2	How much disk space does TAL2C require during a conversion?.....	8
3.3	Which operating systems support TAL2C?	8
3.4	Is TAL2C a graphical program?.....	8
3.5	Does TAL2C run on a NonStop server?	8
3.6	Does TAL2C support pTAL-related keywords added to TAL?.....	8
3.7	What type of C code does TAL2C produce?.....	8
3.8	Can TAL2C generate C++ code?	8
3.9	What type of C compiler do I need to compile the C code produced by TAL2C?.....	9
3.10	Which operating systems support ANSI-compliant C/C++ compilers?.....	9
3.11	Which versions of the Tandem C compiler can I use to compile the C/C++ code produced by TAL2C?.....	9
4	DOING A CONVERSION	10
4.1	What do I have to do before converting a TAL application to C?.....	10
4.2	How do I actually convert a TAL program?	10
4.3	What do I have to do after converting a TAL application to C?	10
5	WHAT IS CONVERTED BY TAL2C?	11
5.1	Is code layout preserved?.....	11
5.2	Are TAL comments converted?	11
5.3	Are TAL variable names retained?.....	11
5.4	What happens to TAL variable names that already have an underscore (_) character in them?.....	11
5.5	How are TAL statements converted?	12
5.6	How are TAL data types converted?	12
5.7	Are TAL FIXED(n) constructs converted?.....	12
5.8	How are TAL subprocs converted?	12
5.9	Are TAL DEFINES converted?.....	12
5.10	How are Tandem file names treated during conversion?.....	12
5.11	What about files that are ?SOURCED in?	13
5.12	What about files that are ?SEARCHED in?	13
5.13	How do I get readable initialization strings in the C code?.....	13

5.14	How do I convert obsolete Tandem (Guardian) routines that are not directly supported in C?.....	14
6	WHAT IS NOT CONVERTED BY TAL2C?.....	15
6.1	Are hardware-specific statements converted?.....	15
6.2	Are hardware-specific functions converted?.....	15
6.3	Are struct types declared in arguments converted?.....	15
6.4	Is equivalencing to hardware registers supported?.....	15
7	HANDLING C COMPILER WARNINGS.....	16
7.1	When can C compiler warnings occur?.....	16
7.2	Turning off C compiler warnings.....	16
7.3	Warning 74.....	16
7.4	Warning 203.....	17
7.5	Warning 252.....	17
7.6	Warning 260.....	17
7.7	Warning 303.....	17
8	MISCELLANEOUS QUESTIONS.....	18
8.1	Can I do a conversion myself?.....	18
8.2	Can I do a conversion at the Gresham conversion lab?.....	18
8.3	Is a TAL2C conversion faster than a manual conversion?.....	18
8.4	Is a TAL2C conversion cheaper than a manual conversion?.....	18
8.5	How much do I need to know about TAL and C to use TAL2C?.....	18
8.6	Is the converted C code easy to maintain?.....	18
8.7	What about native-mode?.....	18
8.8	Can TAL2C convert TAL code with embedded SQL statements?.....	19
8.9	Can TAL2C convert TAL code with embedded SQL to C++?.....	19
8.10	If TAL code has embedded SQL statements, what is the best conversion option to achieve native-mode performance?.....	19
8.11	If my TAL code has variable procedures called from COBOL, how do I call the converted routines?.....	19
8.12	What is the purpose of the file called .defines?.....	19
8.13	What is the purpose of the file called .index_file?.....	19
8.14	What is the purpose of REGION_ files?.....	19
8.15	Can I delete any intermediate files created by TAL2C during a conversion?.....	19
8.16	Is the new C code faster than the original TAL code?.....	20
8.17	What about converted programs that do a lot of I/O?.....	20
8.18	How do I convert a TAL application to run on different platforms?.....	20
8.19	If I want to convert a TAL application to run on a different platform, how do I handle standard system calls?.....	20
8.20	Can I repeatedly convert the same TAL code with consistent results?.....	20
8.21	Can I continue to develop the TAL code and periodically convert it to C?..	20
8.22	Can incomplete TAL programs be converted?.....	20
8.23	Can I convert a TAL application in stages or must I convert all the TAL code at the same time?.....	21
8.24	What about converting small code segments of TAL code?.....	21
8.25	If I already have some manually converted C code, can I use TAL2C to automatically convert the rest of the TAL code and will the automatically converted code be compatible with the manually converted code?.....	21
8.26	Could I re-code TAL source code to improve the performance or readability of the new C code?.....	21
8.27	TAL2C displays messages on the screen but they scroll by too quickly for me to read them. What can I do?.....	21
8.28	If I redirect output, how can I tell where a TAL2C conversion is up to?.....	21

1 Introduction

1.1 Purpose of this document

The purpose of this document is to provide answers to frequently asked questions about how to use TAL2C to convert programs written using the HP NonStop™ server Transaction Application Language (TAL) to ANSI/ISO Standard C or C++.

For more detailed information about TAL2C, refer to the *TAL2C Product Overview* and the *TAL2C User's Guide*. These documents are available with the product and from the Gresham web site on the Internet. See "How to Contact Gresham".

1.2 How to Contact Gresham

Email: tal2c@gresham-computing.com
Web: <http://www.gresham-computing.com>
Post: PO Box 6409, North Sydney, NSW, 2060, Australia
Street: Level 3, 90 Mount St, North Sydney, NSW, 2060, Australia
Phone: +61 (0)2 9955 7660
Fax: +61 (0)2 9955 7687

2 General questions

2.1 What is TAL2C?

TAL2C is a software analyzer that automatically converts programs written in TAL to ANSI/ISO Standard C/C++.

It consists of a fast 32-bit TAL compiler with a C/C++ code generator.

TAL2C is not designed to be a TAL syntax checker. It expects the TAL code to be error-free before conversion begins. However if TAL2C encounters errors, it produces informative error messages.

2.2 How does TAL2C work?

TAL2C is a true compiler that understands TAL language constructs and TAL semantics.

During the compilation of the TAL code, intermediate C/C++ output is written to several files. During the final stage of the conversion, TAL2C automatically merges these files to create the final C/C++ program.

2.3 How fast is TAL2C?

The simple answer is very fast. TAL2C runs on a workstation. Depending on your workstation's CPU speed, TAL2C can convert thousands of lines per second.

Factors such as the size and organization of the TAL code, the number of subprocs, the number of #DEFINE statements, and the number of include files can affect the elapsed time to perform a conversion.

2.4 Where can I obtain TAL2C?

You can obtain TAL2C from Gresham on CD, or by downloading the product from the Gresham web site on the Internet. See "How to Contact Gresham".

2.5 Are there regular TAL2C upgrades?

No, TAL2C is a very stable product. Upgrades are periodically made available at the Gresham web site when problems are found. See "How to Contact Gresham".

2.6 What documentation is available for TAL2C?

In addition to this FAQ, comprehensive documentation is distributed with the product and is available from the Gresham web site on the Internet. See "How to Contact Gresham".

Documentation includes:

TAL to C/C++ Conversion - A Quick Tour: Ready-to-play self-running (10 minute) demonstration that shows you the key features and benefits of TAL2C.

TAL2C Product Overview: Provides an overview of TAL2C.

TAL2C User's Guide: Describes how to install, configure, run, license, and use TAL2C.

2.7 What levels of support are provided for TAL2C and by whom?

Gresham provides an active world-wide support service for TAL2C. The support service includes answering questions on usability, adding new features to TAL2C, making new

releases, researching and providing solutions for customer issues, and supplying fixes for actual bugs. Telephone support is available via a special support number during office hours and support is also available via email and fax. New releases are available to download 24 hours per day from the Gresham web site. Onsite consultancy can also be made available on a per project basis.

See "How to Contact Gresham".

2.8 What types of licenses are available for TAL2C?

There are two types of licenses available for TAL2C:

- The evaluation license is free and is valid for a predetermined period of time. When it expires, download a new version of TAL2C to obtain another evaluation license. This license lets you convert the test and demonstration TAL programs distributed with TAL2C so you can begin to evaluate the product. The terms of the initial evaluation license are distributed with the product.
- A production license allows you full use of TAL2C to completely convert your own TAL code and to use the C output code for your own business purposes. Before receiving a production license, you will be asked to sign a license agreement.

2.9 How much does TAL2C cost?

TAL2C is licensed per conversion project. The cost per project depends on various factors. These include:

- The size of the conversion project: The cost per line decreases as the size of the project increases. (Note that in a manual conversion that you do yourself, the cost per line increases as the size of the project increases.)
- Who converts the code: You or a Gresham consultant.
- Where the code is converted: At your premises or at the Gresham conversion lab.

When you have identified some TAL code for conversion, please contact Gresham to obtain the price.

2.10 Can you provide references from customers using TAL2C?

Yes. Please contact Gresham for more information. See "How to Contact Gresham".

3 Hardware and software considerations

3.1 What is the recommended hardware for running TAL2C?

- IBM-compatible Pentium 133MHz workstation (or faster)
- 64MB RAM (or more)
- 10MB disk space (to install the product)

3.2 How much disk space does TAL2C require during a conversion?

During a conversion, TAL2C requires at least an additional 2MB of free disk space for every 1MB used by the TAL source files (and other required source files).

3.3 Which operating systems support TAL2C?

TAL2C runs on workstations running Microsoft Windows NT 4.0, 2000 or XP.

TAL2C is not supported with 16-bit Windows operating systems.

TAL2C may work with Microsoft Windows 3.x, 95, 98, ME, or NT 3.x. However, TAL2C has not been tested with these operating systems. Running TAL2C with these operating systems is neither recommended nor supported.

3.4 Is TAL2C a graphical program?

No. TAL2C is a command-line program that is executed from a 32-bit Command Prompt.

There are no plans to make TAL2C a graphical program.

3.5 Does TAL2C run on a NonStop server?

No. TAL2C runs on an IBM-compatible workstation.

You must transfer your TAL programs from the Tandem to the workstation where TAL2C is installed before beginning the conversion.

3.6 Does TAL2C support pTAL-related keywords added to TAL?

Yes. TAL2C supports some pTAL-related keywords added to recent versions of TAL (for example: WADDR, BADDR, and so on).

3.7 What type of C code does TAL2C produce?

ANSI/ISO Standard C.

3.8 Can TAL2C generate C++ code?

Yes. TAL2C includes a command-line option (`--plusplus`) that enables it to generate C++ syntax in the output. When this option has been used the C output can only be compiled successfully with a C++ compiler.

This does not mean that the converted code is object-oriented.

3.9 What type of C compiler do I need to compile the C code produced by TAL2C?

TAL2C generates ANSI-compliant C/C++ source. To compile the output, you will need an ANSI-compliant C/C++ compiler.

3.10 Which operating systems support ANSI-compliant C/C++ compilers?

ANSI-compliant C/C++ compilers are available for operating systems such as the NonStop Kernel, Microsoft Windows NT, Windows 2000, and Unix.

3.11 Which versions of the Tandem C compiler can I use to compile the C/C++ code produced by TAL2C?

If TAL2C has produced C code, you can compile the output by using either the Tandem NonStop C compiler (to produce CISC/accelerated object code) or the native-mode C compiler (to produce native-mode object code).

If TAL2C has produced C++ code, you must compile the output using either the Tandem C++ or native-mode C++ compiler.

4 Doing a conversion

4.1 What do I have to do before converting a TAL application to C?

Pre-conversion tasks include:

1. Ensure the TAL application successfully compiles with a TAL compiler. If the TAL code compiles with errors then TAL2C will also produce errors.
2. Back up the original TAL source code.
3. Transfer all the TAL source files (and other required source files) in the application from the Tandem to the workstation where TAL2C is installed.
4. Customize TAL2C, as required, for conversion. Customizing TAL2C is discussed in the TAL2C User's Guide.

4.2 How do I actually convert a TAL program?

Simply run TAL2C from the Windows NT 4.0 or Windows 2000 32-bit Command Prompt, specifying the name of the TAL program.

4.3 What do I have to do after converting a TAL application to C?

Post-conversion tasks include:

1. Transfer the new C source code (and any other required files) to the computer where you intend to build the new C executable. This might be a NonStop server, a Windows workstation, or a Unix system.
2. Compile, build, and run the new C executable.

5 What is converted by TAL2C?

5.1 Is code layout preserved?

Yes. The new C code uses similar layout to the original TAL. This makes the C easy to read, understand, and maintain. The C code is recognizable by the original TAL programmer(s).

A free "pretty print" program (called "indent") is available from Gresham should you want to adopt a different layout standard. You can download "indent" (source and executable code) from the Gresham web site on the Internet. See "How to Contact Gresham".

5.2 Are TAL comments converted?

Yes. All TAL comments are converted to C comments. Wherever possible, comments are kept in the same position in the new C code.

This includes single-line comments, end-of-line comments, and block comments. For example, the following TAL comments:

```
! Single-line comment
... TAL code ...                ! End-of-line comment
!*****
!* Block comment                *
```

are converted to the following C comments:

```
/* Single-line comment */
... C code ...                /* End-of-line comment */
/*****
/** Block comment                */
/*****
```

5.3 Are TAL variable names retained?

TAL variable names are retained in the new C code with a few exceptions (for example, the circumflex (^) character is converted to the underscore (_) character). For example:

abc^def is converted to abc_def

5.4 What happens to TAL variable names that already have an underscore (_) character in them?

The underscore is retained in the C code. For example:

abc_def is converted to abc_def

If the TAL code has two variable names where the only difference is the underscore and circumflex, both names will be the same in the C code. For example:

abc^def is converted to abc_def

abc_def is converted to abc_def

This is unlikely to be the result you want after conversion. To avoid potential variable name conflicts, change one or both names before conversion so that the variable names will be unique afterwards.

5.5 How are TAL statements converted?

TAL statements are converted as expected. For example:

TAL IF statements are converted to C `if` statements

TAL WHILE statements are converted to C `while` statements

TAL DO statements are converted to C `do` statements

TAL CASE statements are converted to C `switch` statements

TAL CALL statements are converted to C function calls

5.6 How are TAL data types converted?

TAL data types are converted as expected. For example:

TAL INTs are converted to C `shorts`

TAL STRINGS are converted to C unsigned chars (`uchar`)

TAL LITERALS are converted to C `enums`

TAL STRUCTS are converted to C `structs`

TAL ARRAYS are converted to C arrays

5.7 Are TAL FIXED(n) constructs converted?

Yes. But the C compiler that you use must have a 64-bit datatype (this is `long long` if using the Tandem C compiler or `__int64` if using the Microsoft C compiler).

5.8 How are TAL subprocs converted?

TAL subprocs are converted to C static functions. Variables from the parent procedure that are used in the subproc are passed as parameters to the function.

5.9 Are TAL DEFINEs converted?

Virtually all TAL `DEFINE` statements are automatically converted to C `#define` statements.

However it is not possible to automatically convert all TAL `DEFINES` to C `#defines`. This is because C `#defines` do not support all constructs supported by TAL `DEFINES`. To avoid this problem if it occurs, TAL2C has a special switch (`--~@`) to enable troublesome `DEFINES` to be expanded inline.

5.10 How are Tandem file names treated during conversion?

Tandem file names are mapped to workstation file names during conversion.

Tandem file names do not have extensions. Workstation file names generally have extensions. TAL2C uses a configuration file to map Tandem file names to workstation file names with extensions. By default the output file name is the same as the input file name, with an extension. For example:

Input header file name: `myfile`

Output header file name: `myfile.h`

You can customize the configuration file to map file names as you wish.

5.11 What about files that are `?SOURCED` in?

TAL files that are `?SOURCED` in are `#included` in the C code. Section names can be preserved.

5.12 What about files that are `?SEARCHED` in?

If your TAL code contains `?SEARCH` directives and you want TAL2C to generate output for the libraries referenced in the `?SEARCH`, you must replace the `?SEARCH` directives with `?SOURCE` directives before conversion. These new `?SOURCE` directives should refer to the source files used to create your libraries instead of the object referred to by the `?SEARCH`.

This is because TAL2C needs to read the files containing the original TAL source in order to convert them, whereas the TAL compiler merely passes the object file names to its bind stage. TAL2C cannot produce C/C++ source from object libraries.

5.13 How do I get readable initialization strings in the C code?

Variables declared as INT P-arrays can be initialized in TAL with strings. However, these cannot be converted directly to C because short arrays in C cannot be initialized with strings. By default, TAL2C generates C that is correct but which may be difficult to read.

If the program that accesses the INT P-array does not rely on the integer nature of the array, then TAL2C has three options to make the output code much more readable.

`--pstring`

INT P-arrays initialized with only strings are converted to arrays of unsigned char (uchar)

`--pstring_any`

INT P-arrays initialized with any strings are converted to arrays of uchar

`--pstring_all`

All INT P-arrays are converted to arrays of uchar

When INT P-arrays are converted to arrays of uchar they are initialized with readable strings.

5.14 How do I convert obsolete Tandem (Guardian) routines that are not directly supported in C?

Some examples of obsolete routines are `GETPOOL`, `PUTPOOL`, and `USESEGMENT`.

There are two methods to convert obsolete Tandem (Guardian) routines.

First (the recommended method), change the obsolete calls to the current calls in the TAL code. For example:

Change `GETPOOL` to `POOL_GETSPACE_`

Change `PUTPOOL` to `POOL_PUTSPACE_`

Change `USESEGMENT` to `SEGMENT_USE_`

You may need to rework the surrounding code to accommodate the slightly different parameter specifications. Then use TAL2C to convert the TAL.

Second, convert the TAL code with the obsolete calls then rework the C code after conversion.

6 What is not converted by TAL2C?

6.1 Are hardware-specific statements converted?

No. Platform-specific statements, such as `CODE`, `STACK`, and `STORE`, have no equivalent in C and are not converted. TAL2C issues a warning if it encounters these statements. If your TAL uses platform-specific statements such as these you must re-code the TAL before conversion or rework the output after conversion.

6.2 Are hardware-specific functions converted?

No. Hardware-specific functions such as `$overflow`, `$rp`, `$type`, and `$usercode` have no equivalent in C and are not converted. If your TAL uses hardware-specific functions such as these you must re-code the TAL before conversion or rework the output after conversion.

6.3 Are struct types declared in arguments converted?

No. C does not allow struct types to be declared in arguments.

To avoid this, you can create a structure template in the TAL code and then use the structure template instead.

6.4 Is equivalencing to hardware registers supported?

No. C does not support equivalencing to hardware registers or offsets to equivalents, because they are specific to the hardware architecture.

If your TAL uses this feature you must re-code the original TAL or rework the C code after conversion.

7 Handling C Compiler Warnings

7.1 When can C compiler warnings occur?

C compiler warnings can occur after you convert TAL code to C, and then compile the converted C code. If a warning message occurs, note the message number and refer to the *C/C++ Programmer's Guide* for more information.

Sometimes you can ignore warning messages, in which case you may want to use the `nowarn` pragma to turn off all or specific warning messages. Sometimes you may have to rework the original TAL code, run TAL2C again, and then compile the converted C code again. Sometimes you can simply change the converted C code without changing and reconverting the original TAL code.

7.2 Turning off C compiler warnings

To turn off C compiler warnings for a certain warning, use the `nowarn` pragma.

You can use the `nowarn` pragma when you run a C compiler. For example, to turn off warnings for warning message 74, use commands similar to these:

```
/* Using the Tandem TNS C compiler */
c /in ctalc, out ctall/ ctalo; xmem, nowarn(74)

/* Using the Tandem native-mode C compiler */
nmc /in ctalc, out ctall/ ctalo; extensions, nowarn(74)
```

You can use the `nowarn` pragma around certain segments of C code. For example, to turn off warnings for warning message 74 around some C code, specify statements such as these:

```
#pragma nowarn(74)
<some C code>
#pragma warn(74)
```

7.3 Warning 74

Warning 74 (initializer data truncated) is generated by the Tandem TNS C compiler.

This can occur when converting a string variable. For example, the warning occurs when converting the following TAL code:

```
STRING
Carriage^Control^s[0:1] := ["  "]
```

to the following C code:

```
_lowmem uchar carriage_control_s[2] = "  ";
```

In this case, the C ANSI standard says that the C code is legal. The Tandem native-mode C and C++ compiler gives no warning.

The solution is to use the `nowarn(74)` pragma.

7.4 Warning 203

Warning 203 (statement is unreachable) is generated by the Tandem native-mode C and C++ compiler.

It occurs when code appears to be inaccessible (the Tandem native-mode C compiler is stricter than the TAL compiler).

Examples include a loop with no exit conditions (typically the message processing loop) or a `break` statement inside a `case` structure that cannot be reached because of preceding `return` statements.

The solution is to restructure the code.

7.5 Warning 252

Warning 252 (argument of type %t1 is incompatible with parameter of type %t2) is generated by the Tandem native-mode C and C++ compiler.

It occurs whenever an unsigned character array was passed to a Guardian procedure that is expecting a string. `CEXTDECS` declares these arguments to be of type `char` and the Tandem native-mode C compiler issues the warning if an unsigned `char` is passed. The code should work correctly (according to the Tandem *C/C++ Programmer's Guide*, both types have the same range of values).

The solution is to either alter the data type or use the `nowarn(252)` pragma.

7.6 Warning 260

Warning 260 (subscript out of range) is generated by the Tandem native-mode C and C++ compiler.

It occurs whenever the generated macro `BLOCK_MOVE_STR` was used to replace a TAL `string` move. This appears to be caused by a bug in some versions of the Tandem native-mode C compiler. The same code compiles cleanly with TNS and Microsoft C compilers. The code functions correctly.

The solution is to use a different version of the compiler or use the `nowarn(260)` pragma.

7.7 Warning 303

Warning 303 (truncation of pointer size performed) is generated by the Tandem TNS C compiler.

It can occur when the C compiler has to convert a 32 bit pointer to a 16 bit pointer on a function call. This occurs because many system procedure calls that had 16-bit pointers have been superseded by calls requiring 32-bit pointers.

The solution is to replace procedure calls requiring 16-bit pointers by procedure calls requiring 32-bit pointers when possible.

8 Miscellaneous questions

8.1 Can I do a conversion myself?

Yes. Two conversion options are available:

- You can license TAL2C "unmanned" and manage the conversion yourself.
- You can license TAL2C "manned" and a Gresham consultant will travel to your premises to supervise the conversion. See "How to Contact Gresham".

8.2 Can I do a conversion at the Gresham conversion lab?

Yes. Two conversion options are available:

- You can send your TAL source to the Gresham conversion lab and a Gresham consultant will do the conversion for you at the lab.
- You can bring your TAL source to the Gresham conversion lab and a Gresham consultant will supervise your conversion at the lab.

See "How to Contact Gresham".

8.3 Is a TAL2C conversion faster than a manual conversion?

Yes. Conversion using TAL2C will always be faster than manual conversion (except for converting very simple TAL programs). Please contact Gresham if you need assistance with estimating the elapsed time to complete your conversion project.

8.4 Is a TAL2C conversion cheaper than a manual conversion?

Yes. Conversion using TAL2C will always be cheaper than manual conversion (except for converting very simple TAL programs). Please contact Gresham if you need assistance with estimating the cost of performing a conversion. If you have a conversion project in mind and you have identified the programs you want to convert, Gresham can provide a written quotation for TAL2C.

8.5 How much do I need to know about TAL and C to use TAL2C?

Programming experience of at least one is preferable. A knowledge of TAL is more important if the original TAL code requires changes.

8.6 Is the converted C code easy to maintain?

Yes. It is easily recognizable by the original TAL programmers. Additionally it is very easy to match the old TAL code with the new C code because layout is preserved and comments are kept in context wherever possible.

8.7 What about native-mode?

TAL2C produces C that you can compile on a Tandem using a native-mode C compiler.

The converted C code can run significantly faster in native-mode, depending on the number of Guardian system calls made. As the number of system calls increases, the less performance improvement is visible (because the Guardian code is already in native-mode).

8.8 Can TAL2C convert TAL code with embedded SQL statements?

Yes.

This is not useful, however, unless you intend to compile the resulting C code with a C compiler that supports embedded SQL. The Tandem C and native-mode C compilers support embedded SQL. The Tandem C++ and native-mode C++ compilers do not support embedded SQL.

8.9 Can TAL2C convert TAL code with embedded SQL to C++?

Yes. TAL2C will convert the TAL code to C++ but the Tandem C++ compilers do not support it.

If you want TAL2C to convert TAL code with embedded SQL to C++, use a workaround. For example, split the TAL code into two parts - code with and without embedded SQL. Convert the TAL code with the embedded SQL to C, then convert the TAL code without the embedded SQL to C++ by using the TAL2C `--plusplus` command-line option. Finally, rework the converted C++ code to call the converted C code.

8.10 If TAL code has embedded SQL statements, what is the best conversion option to achieve native-mode performance?

pTAL does not allow embedded SQL. Therefore your best conversion option is to use TAL2C to convert the TAL code to C and then to compile the resulting C code using the Tandem native-mode C compiler.

8.11 If my TAL code has variable procedures called from COBOL, how do I call the converted routines?

You cannot directly call native-mode C routines that are variable or extensible from native mode COBOL programs.

There is a work-around available using pTAL stubs if this situation is unavoidable. Create a variable pTAL procedure for the corresponding variable C routines. Each pTAL procedure simply calls the corresponding C routine. Your COBOL programs can then call the pTAL procedure.

8.12 What is the purpose of the file called .defines?

TAL2C creates this intermediate file during a conversion for processing `DEFINE` statements.

8.13 What is the purpose of the file called .index_file?

TAL2C creates this intermediate file during a conversion to keep track of multiple files over multiple runs.

8.14 What is the purpose of REGION_ files?

TAL2C creates one or more intermediate files whose name begins with `REGION_` during a conversion. These files contain segments of C code. TAL2C concatenates these files during the final stage of a conversion, to create the final C output file.

8.15 Can I delete any intermediate files created by TAL2C during a conversion?

Intermediate files include `.defines`, `.index_file`, and various `REGION_` files.

You should not delete any intermediate files created by TAL2C during a conversion project unless you intend to begin the conversion project again from the beginning. In this case, you can delete all intermediate files.

A batch file called `cleanup.bat` is available to help you clean up after a conversion. This file removes all TAL2C intermediate files from the current directory.

8.16 Is the new C code faster than the original TAL code?

Yes. Best results are obtained with the Tandem native-mode C compiler.

Performance is comparable between accelerated Tandem C and accelerated TAL.

8.17 What about converted programs that do a lot of I/O?

The converted code will still run faster but the improvements may not be obvious because of the I/O overheads.

8.18 How do I convert a TAL application to run on different platforms?

In general there is no difference in the way you use TAL2C to convert TAL to C for different platforms.

However if a TAL application includes hardware-specific constructs and you intend to build the C for a non-Tandem platform, you must rework the hardware-specific code.

You can either rework the original TAL code and run TAL2C again or change the C code.

(Additionally, you must not use the `--tandem` command-line option.)

8.19 If I want to convert a TAL application to run on a different platform, how do I handle standard system calls?

C functions corresponding to some Guardian procedure calls, such as date/time and pool routines, are available separately from Gresham upon request.

However, you must manually convert calls that rely on Guardian functionality, such as transactions and checkpointing.

8.20 Can I repeatedly convert the same TAL code with consistent results?

Yes. In some cases, customers have found bugs in their TAL code that only become apparent during conversion. After making fixes to their TAL code, they successfully re-ran TAL2C.

8.21 Can I continue to develop the TAL code and periodically convert it to C?

Yes. You can treat the TAL code as your master code base and use TAL2C to convert the TAL to C when you want to.

8.22 Can incomplete TAL programs be converted?

Only if they compile. Because TAL2C is a compiler/converter, a clean TAL compilation is necessary before TAL2C is able to compile and convert the code.

8.23 Can I convert a TAL application in stages or must I convert all the TAL code at the same time?

You can convert a TAL application in stages if you can successfully compile the TAL code at each stage.

8.24 What about converting small code segments of TAL code?

You can convert a TAL code segment if you can successfully compile the segment.

8.25 If I already have some manually converted C code, can I use TAL2C to automatically convert the rest of the TAL code and will the automatically converted code be compatible with the manually converted code?

Yes and yes, if both sets of code maintain the same naming conventions, files structures, and so on.

8.26 Could I re-code TAL source code to improve the performance or readability of the new C code?

It depends. If the TAL code was designed for portability then it is unlikely that re-coding will yield any benefits.

However, when TAL2C encounters TAL code that could result in inefficient C it displays a warning. In these cases you might consider reworking the TAL to avoid the warning and improve efficiency.

8.27 TAL2C displays messages on the screen but they scroll by too quickly for me to read them. What can I do?

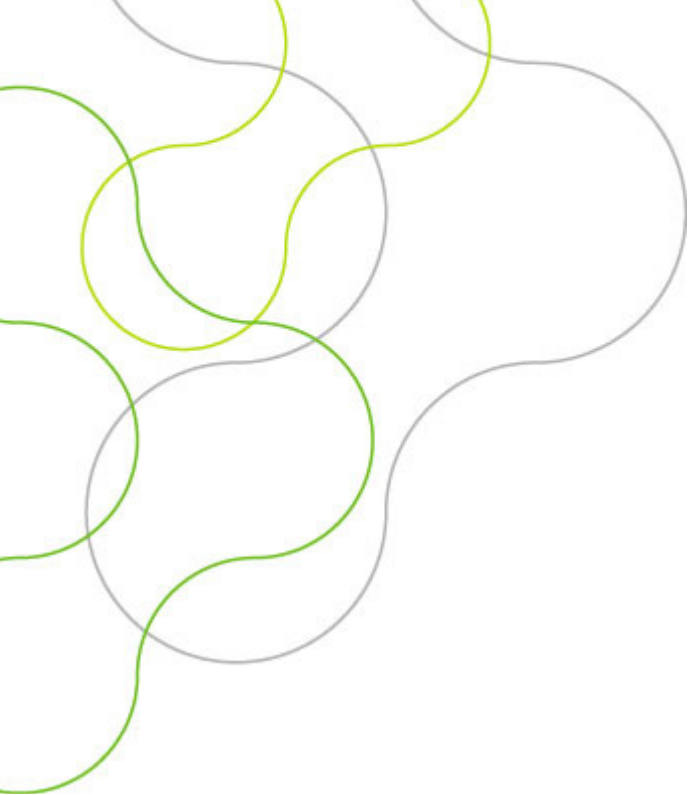
Either, you can redirect the messages to a text file on the workstation and read the messages in the file when TAL2C has finished. The following example redirects error output to a file called `errlog.txt`:

```
tal2c myTAL 2>errlog.txt
```

Or, you can choose the Command Prompt Properties>Layout tab, increase the Height value of the Screen Buffer Size, then scroll up and down to read the messages on the screen when TAL2C has finished.

8.28 If I redirect output, how can I tell where a TAL2C conversion is up to?

TAL2C includes a command-line option (`-z`) that enables it to display progress messages on the screen, regardless of where conversion messages are being sent.



About Gresham

Gresham Computing plc (LSE:GHT) specializes in the provision of real-time financial solutions to banks and corporates, and has a well-deserved reputation for technical excellence, reliability and a strong service culture. Our storage division helps the largest data users to better manage the unrelenting growth of data.

Further information

For more information on how TAL2C can help your company visit

www.gresham-computing.com

or you can email us at

tal2c@gresham-computing.com

Alternatively you can contact our offices directly.

Europe, Middle East and Africa

T +44 (0)20 7653 0200

Americas

T +1 416 620 6683

Asia Pacific

T +61 2 9955 7660