A Layman's Guide to TPF C

Author	:	Simin Marsden
Original Version	:	1.0, December 2002
Last Update	:	9/12/2002
Issued by	:	Simin Marsden
Reviewed by	:	Ruud Schelvis, Keith Payne

1 Introduction

The aim of this document is to indicate to Transaction Processing Facility (TPF) or Airline Control System (ALCS) professionals the possible advantages of using the C language. For simplicity we shall just refer to TPF in this document and recognise that ALCS is included in that.

As the business logic for airline reservation systems gets more complex the old Assembler code is turning more and more into a bowl of spaghetti. The "original" people who wrote the "original" code are long gone and the junior programmers modifying a piece of code have no idea what damage they can cause down the line because of the misuse of a work area. Traditionally good documentation is produced during the initial design and development of a TPF package. However the documentation is not always updated during later modifications to the system, especially if the modifications are done in a hurry to solve problems.

The speed (or lack of it) of application development and the slow learning curve of application programmers has been a concern to the IT management in TPF environment for decades. It takes around one year for an application programmer to become reasonably productive. Even an experienced TPF programmer cannot produce code at the same pace as programmers working on other platforms. Assembler was seldom included in formal IT education and it definitely is a very foreign concept to the new graduates now.

Although it is not the "magic solution", using C language can help with the above issues. It can result in more standard and structured code and could improve programmer productivity in both development and maintenance phases. The development and debugging tools have a more familiar look to today's IT professionals. It is possible to port an application to the TPF system that is compliant with the Portable Operating System Interface for Computer Environments (POSIX) standards. At the moment IBM is committed to the 64-bit release of TPF. This would make the use of C/C++ advantageous over assembler. In the Fall 2002 TPF User Group Meeting, IBM has also announced that TPF is becoming a C/C++ machine with open tooling from the application's standpoint (like with UNIX).

There will always be traditional assembler code to be maintained; but some TPF installations are developing all new functionality using ISO-C. The use of C++ is still in an experimental stage, since its performance impact can be extreme. There are also attempts to re-write existing packages using ISO-C.

Before the standard C support, the TPF system supported a non-standard implementation of a C language subset known as Target (TPF). Support of the latter is no longer enhanced or used by IBM, although it is still available for legacy applications that have not been updated or migrated to standard C support. Many installations have already converted their Target-C applications into ISO-C.

2 What is TPF-C

TPF-C is IBM C/C++ language support for Transaction Processing Facility (TPF) application programming, which permits application programmers to write TPF programs in C or C++ language. TPF C language support conforms to ANSI and ISO standard C as defined in ANSI/ISO 9899-1990 (ISO-C). IBM TPF C/C++ language support requires the use of the C/C++ compiler to be installed and available. Both the MVS and VM versions of this compiler support the TPF system. C/C++ language support consists of modifications made to the TPF operating system to interface with IBM C/C++ compiler-generated code and a set of TPF-specific library functions.

What is the difference between a classic TPF Assembler Segment and a Load Module? A classic TPF assembler segment is limited to size 4 KB. Once it has been assembled by HL-ASM it is ready to be loaded into the TPF system and it is not link-edited. A load module instead is not limited to size 4KB; it can go up to 16MB. It is link-edited and may include different object files. The source code for the object files within a load module can be written in Assembler, C or C++. A load module can be an application module (Dynamic Link Module-DLM), a run-time non-dynamic library or a Dynamic Link Library (DLL).

The application code is loaded into TPF system in form of DLM's. A typical DLM contains:

- Start up code to set up C environment
- Objects such as application code compiled with ISO/C compiler or application code assembled with HL-ASM
- Stubs to resolve references outside the DLM

Libraries contain library functions that can be called by applications. The following are some standard non-dynamic libraries:

- CISO- Standard C library
- CTAL-Standard TPF API library
- CTDF-TPFDF library
- CTBX-ISO-C general purpose toolbox library

Installations can decide to create their own libraries, which contain common functions that are used by many applications.

2.1 Benefits of TPF-C

2.1.1 Application programmer productivity benefits

The TPF-C development environment with Visual Age Editor and Source Level Debugger is a much more "contemporary" development environment and can increase programmer productivity especially for junior programmers. Because C language statements are concise statements and the compiler does extensive syntax checking, fewer errors are introduced into application programs, increasing the probability that programs will run correctly the first time they are executed. IBM currently has plans to rewrite the Visual Age product based on open standards (Eclipse). The Eclipse Platform is designed for building Integrated Development Environments (IDE's) that can be used to create applications as diverse as web sites, embedded Java TM programs, C++ programs, and Enterprise JavaBeans TM.

The TPF-C code is by nature structured and "bounded". A programmer can code a piece of code assigned to him/her without the need to know the details of the whole package. He/she does not need to worry about finding that free byte in the work area and try to make sure that it is not used 10 programs down the line. C is a call-by-value language; it means that a function always receives a copy of the arguments passed to the function. This property of the C language makes it difficult to override variables of work areas by mistake. It is possible to call a DLM from a traditional assembler program and vice versa. Enhancements to an old package can be written in ISO-C with less worry about the register and/or work area usage in the old application. Because an application programmer is not required to code registers explicitly, the likelihood of certain errors caused by corruption of main storage is greatly reduced. Storage for C variables is allocated by the system, reducing the need for programmers to manage data in core blocks. This makes it easier to "copy" code from an old program doing a similar job to a new program without the worry of register/work area usage.

If the installation applies good naming conventions for variables, arguments, data structures and functions, the code is much easier to read and understand than the BAL Assembler code. Application programs can be easily modularised by taking advantage of the C *function* concept.

The standard C library provides powerful functions for string manipulation etc. for the use of application programmer. A single function can do the work of lines and lines of assembler code. Use of the standard library functions makes the code more standard. I.e. there is often only one standard way of doing a piece of work. This nature of C language makes it possible for some installations to generate code from the structured charts. Another advantage of library functions is that they are easy to maintain and reuse in many applications. If the function needs to be updated only the function needs to be recompiled, and only the library load module needs to be re-linked and loaded.

The use of C++ and class libraries may bring the benefits of Object Oriented programming into TPF. However there are not many installations at the moment exploring these features. One very good usage of C++ in TPF environment would be the XML parser. XML4C parser 3.5.1 (APAR PJ28176) is a port of XML Parser for C++ (XML4C) Version 3.5.1 to the TPF 4.1 system. The parser is XML Version 1.0 compliant and allows TPF 4.1 applications written in C++ language to parse XML documents.

2.1.2 Things to be careful about

The TPF IT manager's dream of getting C programmers off the market to develop TPF-C code the next day is only a dream. Every programmer must be educated about the functioning of the TPF system and assembler before they are allowed to load new code to the system. Management often believes that C code automatically gets written to high quality, whereas the quality has to be written in (by the programmers). The speed advantage of C is always over-estimated, especially in the beginning. This leads to programmers being pressured to hurry their work, which leads to bad C.

The coding standards, naming and commenting conventions for the C code should be set carefully. Otherwise the idea of "self-explanatory, no need to comment" C code can turn into a bigger nightmare then the old assembler code. It is possible to create spaghetti with any language, no matter how high level it may be.

When a TPF-C segment needs to access an existing TPF database record a C header file must be created to reflect the TPF data macro in C environment. This can be a very time consuming experience and should be done very carefully otherwise it can result in database corruption.

The TPFDF API of ISO-C is one of the weak points of TPF-C. Multiple C functions should be coded for each TPFDF command. Some installations overcome this problem by creating their own library functions to act as a middleware between the application program and the TPFDF API.

How ISO-C functions are packaged can affect the overall performance of a program. There are several ways to package functions:

- As **inline functions** of the compiler. These functions are directly in place where they are called. The fastest performance for a function comes from using the INLINE option of the compiler. Using inline functions eliminates entirely call linkage. The disadvantage of using inline functions is the maintenance. The source code for the functions is included in header files for all the programs that use them. If a function needs to be updated all applications that call it must be recompiled, relinked, and reloaded.

- As **internal functions** added by the linkage editor. These functions are contained in the same load module as the calling function. They are accessed without using a stub and are faster than those accessed through a stub. If a function needs to be

updated, only the function itself must be recompiled, but all applications that use the function still must be relinked, reloaded, and tested again

- As **library functions**. These functions are referenced by stubs added to the load module during linkage editing. The use of library functions involves a slightly longer path length than the use of link-edited functions. This is because the calling program must use a library call stub to access library functions. The advantage of library functions is that they are easy to maintain and reuse in many applications. If the function needs to be updated only the function needs to be recompiled, and only the library load module needs to be relinked and loaded
- As **external functions** in the form of a DLM. These are function calls to separately compiled programs. DLMs are the primary structure for ISO-C functions. Call and return linkages between DLMs are managed by TPF system services and are comparable to ENTRC and BACKC linkage. The performance decision involves the number of DLMs that need to be created for a given application. The fewer DLM calls needed, the faster a given application performs. The disadvantage is the loss of visibility of the function of a DLM to the system. If DLM A is included in DLM B to improve DLM B's performance, the service that DLM A provides is lost to other DLMs (without replicating DLM A). If a DLM ends up containing too much code, it can become a nuisance to update. Several programmers trying to work on the same DLM leads to a lot of double maintenance.

The use of recursive C functions can be damaging to the performance of the system. They should be used with caution.

Even if an installation decides to do all new development in C, some careful thinking should be done when creating very highly used functions. Sometimes it can be a good performance option to code a library function in assembler, which can easily be called from a C program.

Live system coverage and dumps in C-DLMs can initially be a challenge to the old-timer coverage programmer. All coverage programmers should be able to read ISO-C stack frames in a dump and should be aware that the live and development versions of the C programs differ because of the optimisation and "hooks" for the debugger. (IBM has announced in fall 2002 TUG that the new version of the debugger will not have the need to compile with TEST (HOOK) option.) In general, it is much more difficult to solve C dumps than assembler dumps. On rare occasions optimised code can act differently from the development code. Some careful pre-implementation test should be done with the optimised code before the Live load.

Considerable amount of initial investment is required by systems (to support development in C) and by application programmers (training, copying macros to C, preparing C versions of common functions, etc.). For installations that do major developments the payback time is couple of years. For installations, which do not do any major developments, it may not be worth the effort.

3 Use of TPF-C in the development cycle

The following lists the 'typical' steps used in the development cycle:

1. Requirements specification
2. Functional Design (Logical Database and Application Design)
3. Technical Design (Physical Database and Application Design)
4. Application Build and Test
5. Implementation
6. Maintenance
7. Future enhancements

The benefits of TPF-C are most obvious in the Application Build and Test phase. However all the steps except for Requirement Specification can benefit from the use of TPF-C.

The TPF-C advantages and features applicable for steps 2 through 7 can be summarized as follows:

2-Functional Design: Generally the functional designs try to explain the required functionality in long wordy detail, and sometimes the final code produced does things slightly different than what was intended in the functional design. The functional design should be independent of in what language the actual code is going to be built. However, adapting the ISO-C philosophy of single entry and exit points for each piece of functionality and defining what goes in and out of each piece of functionality would result in clearer functional designs. The functional designs created this way can later be turned into technical designs and code, which actually does what was intended in the design.

3-Technical Design: Technical design is the crucial phase of development as far as ISO-C is concerned. The all-important decisions about how to "package" the application and if or what piece of code needs to be written in assembler should be made at this stage.
Not to re-invent the wheel and make good practice of C programming careful investigation should be done to find out if there is an existing function doing the required piece of work. This way there will be only one standard way of doing a piece of work.
Again applying the rule of single entry and exit points for each function and defining what goes in and out of a function would result in very clear structured charts.

4-Application Build and Test: The structured design built in the previous phases can be turned into C code very rapidly. The main structure of the program is exactly as it is in the structured chart. Some installations use code generators that turn the structured chart into skeleton C code and the programmer only needs to fill in the details. Colour coded Visual Age Editor points out the syntax errors as the programmer types the code. The Visual Age Source Level Debugger allows the programmer to use run options like step-over, step-return, run-to-cursor and set breakpoints to stop at specific instructions or

source lines of a program. When the debugger stops at an instruction or a source line, the programmer can display and/or alter data via several monitors like a storage monitor or a local variable monitor.

5-Implementation: At this stage the code should be doing what was said in the functional design and implementation can go ahead without the sometimes-annoying iteration process between the functional analyst and the application programmer. IBM Visual Age Debugger Performance Analyser tool can be used to identify performance bottlenecks in an application.

6-Maintenance: The readability of the good C code and the clear definition of each piece of functionality in functional/technical designs are the benefits in the maintenance phase. The Visual Age Performance Analyser can be used at this stage as well.

7-Future Enhancements: The benefits of the phase 6 are also applicable to this phase. Another considerable advantage can be in doing enhancements to an old package written in ISO-C with less worry about the register and/or work area usage in the old application.

4 List of acronyms

ALCS	Airline Control System ALCS is also referred to as TPF/MVS. Unlike TPF, ALCS runs as an application under MVS. As such it is able to utilize all the generic services that are provided to all jobs and applications running in this same environment.
DLL	Dynamic Link Library A collection of one or more functions or variables gathered in a load module and executable or accessible from a separate DLL application load module.
DLM	Dynamic Load Module A collection of compiled and linked source programs that are ready to be loaded into main storage and run
HL-ASM	High Level Assembler
ISO-C	C language implementation that conforms to ANSI and ISO standard C as defined in ANSI/ISO 9899-1990.
POSIX	Portable Operating System Interface for Computer Environments. An interface standard governed by the IEEE and based on UNIX. POSIX is not a product. Rather, it is an evolving family of standards describing a wide spectrum of operating system components ranging from C language and shell interfaces to system administration.
TPF	Transaction Processing Facility Try to obtain a copy of the 'Layman's Guide to TPF'. This will give you all the insight that you need.
XML	Extensible Markup Language