



PL/I to C++ conversion service

Version 1.1

17 September 2002

Stromasys

P.O.Box 156

1228 Plan-les-Ouates

Switzerland

Tel: +41 22 794 1070

Fax: +41 22 794 1073

www.stromasys.com

info@stromasys.com

This document is provided for information and is not a legally binding offer. Stromasys reserves the right to change service specifications without prior notice or retire the specific service offered. Trademarks are the property of their respective owners.

Service availability

The PL/I to C++ programming language conversion service is available as part of a platform migration project or as a separate service. In the latter case a typical project is split into a fixed-price project evaluation phase and an execution phase.

The fixed price evaluation delivers a detailed project plan within four weeks of its commencement and includes a cost estimate for the conversion. If the project is ordered within two months of completion of the project plan, this estimate can be considered as a fixed-price quotation.

The PL/I to C++ programming language conversion service provided by Stromasys is a risk free way to transform PL/I applications into modern C++ code, while maintaining the original application functionality. Such conversion can generate substantial cost savings; older tools, development platforms and maintenance skills are no longer required. Modern platforms and programming languages make it easier to attract good software engineers, maintain code and offer a wider choice of powerful development platforms. Programming language conversion can significantly ease the transition to a new system environment.

Stromasys provides a comprehensive service designed to limit the involvement of your own staff to the final acceptance testing after delivery of the translated applications. We can tailor the conversion process to company specific coding standards, and replace middleware and system calls. Several levels of refinement are available, ranging from a rapid conversion for unmodified further use to development of customer specific methods and classes. In all cases, the original functionality is maintained by default.

Service components

The PL/I to C++ programming language conversion service is based on the following main components:

- A set of source code analysis tools to gather statistical information and to detect constructions and extensions that cannot be converted automatically. This information is used to tune the PLI2CPP converter and to extend/develop the C++ runtime libraries if necessary. This analysis also covers the system functions of the host platform operating system, an essential part to be considered in the migration process.
- The PLI2CPP converter engine to handle automated volume conversion of the PL/I source code. The PLI2CPP converter is parameter controlled and manually adjusted based with the information obtained in the first step.
- The PL/I support C++ runtime library.
- Specific extensions to efficiently handle VAX PL/I.
- Specific emulation libraries for Operating System dependent code segments. Stromasys developed technology templates for migrations between different host platforms (e.g. from VAX VMS/OpenVMS to industry-standard UNIX or Windows 2000 systems).

The conversion service includes

- A feasibility analysis to determine manual and automatic components.
- Pre-conversion code preparation, cleaning up, filtering and processing with 'standardization' tools.
- The analysis of OS-specific extensions and packages used in the application.
- Bulk conversion of PL/I code to C++ using the PLI2CPP converter.
- Generation of OS-specific migration components for the new platform.
- Post-conversion clean-up (e.g. inline comments).
- Manual conversion of the outstanding conversion issues as reported by the initial analysis and PLI2CPP.

Code conversion principles

From a design point of view, PL/I and C++ have radically different philosophies in implementing an application design. PL/I places an emphasis on routines and structural design, while C++ places its emphasis on classes and object orientation.

There are also some specific data types and some control statements that cannot be directly converted to C++ statements without introducing emulation. Most of the control statements and data types naturally convert to C++ with no loss in readability, but some compromises for PL/I-specific constructions are made in the converted code.

Conversion vs. emulation

A predefined PL/I language environment consists of a set of standard functions provided by every PL/I implementation.

Some of the procedures implemented in these packages can be converted to native C++ functions, while others need special emulation provided by the PL/I support C++ runtime library.

VAX PL/I

The VAX PL/I language provides extensions for procedure calling, condition handling, compilation control and miscellaneous purposes, as well as support integration of the VMS System Services and VMS Run-Time Libraries.

Some of the procedures and objects implemented in these packages can be converted automatically to native C++ runtime classes and functions with translator's embedded conversion tables containing PL/I vs. C++ system calls formats, while others require special classes and objects from the VMS Emulation Library (VEL) of Stromasys.

- Converted source code style improvements and redesign (if required).

Code conversion principles

The following specific constructions and situations are handled separately during PL/I to C++ conversion:

▪ **Functions and subroutines**

PL/I functions and subroutines are converted to C++ functions. Nested subroutines are moved out of outer subroutines to the global scope (possibly with prefixes in names to resolve name conflicts). Variables from outer functions that are also used in nested functions are not moved automatically to global scope. They are moved manually with local redesign.

▪ **Declaration of types**

Converted either automatically if understood by translator (in case there is direct matching with the C++ variable's types), otherwise converted manually (all such cases are explicitly marked by the translator).

▪ **Data types conversion**

Fixed-point binary data without precision are converted to an integer type of the appropriate length. Fixed-point binary data with precision have a fractional part and is converted to double. Each operation with such converted data types require restoring of the internal representation of such value using multipliers. Floating-point variables are converted to double. Single char is converted to string with single char and NULL byte.

▪ **Pictured Data conversion**

Pictured variables have fixed-point decimal attributes, but values of the variable are stored internally as character strings. The character string contains decimal digits representing the numeric value of the variable, plus special-editing symbols described in the picture.

To represent pictured variables in C++, the following method is used:

1. All pictured variables are stored as integer or double.
2. For each pictured variable character array is added.
3. At places where character representation of pictured variable is required, actual conversion is supplied and the character array is used instead.

▪ **Character array (string) conversion**

1. Strings are converted to zero-ended strings, which are artificially filled by spaces. This is done automatically at the conversion stage when string length is explicitly declared, or in case when the string is transferred as parameter and the string length is unknown, by introducing of the additional parameter (string length).
2. The built-in function TRIM is ignored.
3. In places where a whole line (with spaces) is needed, function expand() is used.
4. Conversion routines for integer and floating-point data are supplied (character_d() and character_f()). They append values to the end of first argument without spaces.
5. A two-dimension array of chars is converted to a array of pointers to char filled dynamically with malloc().
6. If the substr() operator appears at the left side of assignment, calls for substr_l() appears with all the original parameters plus the right side of the assignment.

▪ **Conditions handling for IO**

Decentralised solution is used, when the translator automatically binds the condition check with the operator which might cause this exception.

▪ **Built-in's conversion**

All built-in functions/operators that may be emulated directly by C++ run-time library calls are substituted by equivalent functions. Others are emulated by the supplied PLI2C run-time library.