



ADA to C++ Conversion service

Version 1.0

14 February 2001

Stromasys

P.O.Box 156

1228 Plan-les-Ouates

Switzerland

Tel: +41 22 794 1070

Fax: +41 22 794 1073

www.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 ADA 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 ADA to C++ programming language conversion service provided by Stromasys is a risk free way to transform ADA applications into C++ code, while maintaining the original application functionality. Where ADA skills are hard to come by, such conversion can generate substantial cost savings by harmonizing tools and development platforms. Developing and maintaining code in C++ makes it easier to attract good software engineers and offer a wider choice of powerful development platforms. ADA to C++ 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. 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 description

The ADA 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 ADA2CPP converter and to extend/develop the C++ runtime libraries if necessary. This analysis also covers the system functions of host platform operating system, an essential part to be considered in the migration process.
- The ADA2CPP converter engine to handle automated volume conversion of the ADA source code. The ADA2CPP converter is parameter controlled and manually adjusted based on the information obtained in the first step.
- The ADA support C++ runtime library.
- Specific emulation libraries and standard approaches used in operating system migration. Stromasys developed a number of migration 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 ADA code to C++ using ADA2CPP 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 ADA2CPP
- Converted source code style improvements & redesign (if required).

Code conversion principles

From a design point of view, ADA and C++ have radically different philosophies in implementing an application design. ADA, particularly ADA 83, places an emphasis on "packages", while C++ places its emphasis on Classes and Methods.

There are also many specific data type and structure declarations and some control statements, which cannot be directly converted to "plain" C++ declarations or structures. This means that while basic control structures and structures are converted with no loss in readability, some compromises for the ADA-specific constructions must be made in the converted code.

Conversion vs. emulation

A predefined ADA language environment consists of a set of standard packages provided by every ADA implementation.

Some of the procedures and objects implemented in these packages can be converted to native C++ runtime classes and functions, while others need the special classes / objects from the ADA support C++ runtime library.

The following constructions and situations are handled during the ADA to C++ conversion as indicated:

- **Package**
To preserve name-spaces and declaration syntax of the original ADA source code, packages are converted to C++ classes with only static member variables and functions. Each package is translated in two separate C++ files with corresponding class definition and implementation.
- **Data types defined as ranges**
These types are converted to the instances of the special template class defined in the ADA support C++ runtime library (to preserve range checking and exception handling) or to plain C++ integers (to have better performance).
- **Strings**
Strings are converted to string classes from ADA support C++ runtime library.
- **Arrays**
Arrays and array types are converted to the instances of template array classes from ADA support C++ runtime library.
- **Attributes of types/objects**
Some predefined ADA attributes ('ADDRESS', 'LENGTH', 'SIZE' etc) are converted directly to appropriate C++ constructions that don't need run-time support. Other attributes are converted by the automatic conversion tool to calls of member functions for the corresponding classes in converted solutions. For many classes provided by ADA the corresponding functions in the support C++ runtime library are implemented by default. For other classes these functions are implemented manually, if necessary.
- **Named arguments**
Named arguments of functions/subroutines are converted to positioned arguments of the corresponding C++ functions.
- **Discriminants**
Compound data types with discriminants are converted to C++ classes. Discriminants are passed as a parameters to class constructors.
- **Variant records**
Variant records are converted to C++ classes with tagged unions. Constructors and operators are provided to preserve the initial semantics.
- **Generic units**
Generic units (functions, packages etc.) are converted to template functions/classes in resulting C++ code.
- **Access data types**
Access data types are converted to standard C++ pointers.

The major standard ADA packages and the solutions used for each package:

- **STANDARD**
Types and procedures from this package are converted to native C++ objects.
- **CHARACTERS and CHARACTERS.HANDLING**
Types and procedures from this package are converted to native C++ objects.
- **STRINGS**
Types and procedures are converted to C++ string classes from ADA support C++ runtime library.
- **NUMERICS Packages**
Types and procedures from this package are converted to native C++ objects.
- **TEXT_IO**
Types and procedures are converted to C++ classes implemented in ADA support C++ runtime library.

Tasks and synchronization

ADA has built-in support for concurrent processing with tasks and related concepts. These features are used intensively in many applications written in ADA. To simplify the conversion of such applications, a C++ library is used that implements the same architectural approach as used in the ADA language. This library includes:

- **TASK template classes**

These classes implement a generic mechanism for task creation, life cycle control, task dependences and inter-task communication. All ADA tasks are converted to C++ classes derived from these template classes.

- **GUARD template classes**

These classes implement a generic mechanism for concurrent and exclusive data access. All ADA protected units and objects are converted to C++ classes derived from these template classes.

Migration of VMS-specific extensions of DEC ADA

The DEC ADA language environment has additional sets of packages for working with RMS files, X-Window, System Services and VMS Run-Time Libraries. Some of the procedures and objects implemented in these packages can be converted to native C++ runtime classes and functions, while others need special classes / objects from the general VMS Emulation Library (VEL) developed by Stromasys. The main DEC ADA packages and the solutions used for each package.

Standard packages

- I/O packages

The FORM parameter of I/O routines can handle the RMS File Definition Language statements or a reference to the FDL file.

Solution: Manual redesign to flat data files or to third party FMS products (such as C-ISAM) for keyed index data files.

Packages specific to DEC ADA

- SEQUENTIAL_IO, SEQUENTIAL_MIXED_IO, DIRECT_IO, DIRECT_MIXED_IO, INDEXED_IO, INDEXED_MIXED_IO, RELATIVE_IO, RELATIVE_MIXED_IO, AUX_IO_EXCEPTIONS

Provides types and operations for working with sequential, direct, indexed and relative files of uniform-type or mixed-type elements.

Solution: Manual redesign to flat data files or to third party FMS products (such as C-ISAM) for keyed index data files.

- RMS_ASYNCH_OPERATIONS

Provides supporting operations for the package TASKING_SERVICES.

Solution: Manual redesign.

- X, XM, XM_CONVENIENCE, XM_STRING, XM_WIDGET, XT, XT_STRING, X_LIB and X_RESOURCE

Provides interface routines for the X Windows System routines, X Windows Toolkit, and Motif routines.

Solution: Calls to procedures from these packages are converted to native X libraries calls.

- CLI

Provides types and operations for calling VMS Command Language Utility routines.

Solution: Manual redesign or conversion to classes/objects from our VMS Emulation Library.

- CONDITION_HANDLING

Provides types and operations needed to evaluate the condition values returned by system routines.

Solution: Manual redesign.

- STR, DTK, LIB, OTS, PPL, SMG, MTH

Provides types and operations for calling the VMS Run-Time Libraries STR\$, DTK\$, LIB\$, OTS\$, PPL\$, SMG\$, MTH\$ routines.

Solution: Manual redesign or conversion to classes/objects from our VMS Emulation Library.

- STARLET

Provides the types, operations, constants, and so on that you need to call VMS system service and RMS routines.

Solution: Manual redesign or conversion to classes/objects from our VMS Emulation Library.