

Mrs. Lisa Rajchel/Secretary JTC 1
Mr. Keith Brannon/ITTF

OUR REF. ECMA-372: Response to results of Review
YOUR REF.

DATE: 29th of January 2006

Dear Lisa and Keith,

This letter is to provide you with the answer to your emailed request of 25 January 2006.

Best regards.

Jan van den Beld
Ecma International, Secretary General

Ecma is grateful for the extensive input received from SC22/WG21 ("WG21") during the development of the C++/CLI standard. This was possible through the combined efforts of Ecma TC39/TG5 ("TG5"), Ecma International, WG21, and ISO/IEC JTC1 ("JTC1"):

- Since 2003, TG5 and WG21 had a broad bidirectional liaison and actively worked to eliminate conflicts between C++/CLI and both ISO C++ and future C++0x evolution:
 - The membership of TG5 was a subset of the technical experts participating in WG21.
 - All overlapping proposals (including features that might overlap on areas where ISO C++ was considering extensions or changes in C++0x) were brought to WG21 for their input and direction. TG5 sought and followed WG21 direction on not only how to be compatible with ISO C++ but also how to stay out of the way of ISO C++0x evolution (e.g., with the "for each" feature, TG5 asked WG21's direction about what syntax that WG21 wanted to reserve the right to consider for themselves, and TG5 deliberately used a syntax WG21 were not interested in using so that C++/CLI would not conflict with C++0x evolution).
- Since 2003, Ecma made special provisions and granted all SC22 and WG21 members unprecedented liaison access to fully participate in TG5 without cost or fee (except only not being able to cast a TG5 vote), including:
 - WG21 members could attend all TG5 meetings, and TG5 scheduled its meetings to locate with or near every WG21 meeting during TG5's active work. Several WG21 members from UK, US, and France took advantage of this and attended one or more TG5 meetings, and made very helpful technical and procedural contributions, all of which were considered by TG5 and most of which were adopted (in a few cases there were technical reasons not to adopt the requested changes, but they were still considered, often multiple times). TG5 spent significant time in multiple meetings with WG21 members on their proposals and concerns on both technical and nontechnical issues, and the C++/CLI standard has clearly benefited from this and is of higher quality because of it.

- WG21 members had access to all TG5 technical discussion on the email reflectors. Ecma has created a special email reflector for this purpose, and much useful input came through that reflector.
- WG21 members had access to all TG5 documents. All TG5 documents were provided in liaison reports in WG21 mailings, including working drafts of the C++/CLI standard and meeting minutes.

All of these provisions were actively used, and Ecma is grateful for the input – standard C++/CLI incorporates much input from WG21 members, most notably from UK participants, and it is a better standard for having incorporated their input.

The following are responses to specific parts of the comments from Germany and the UK:

On a technical level, there are some rather different approaches between C++ and C++/CLI which can easily cause considerable confusion when both languages are considered to be "C++".

Actually, C++/CLI is not a "language," but a "binding." The purpose of C++/CLI is to permit a programmer who knows C++ (ISO/IEC 14882) to write programs that utilize the capabilities of the CLI (ISO/IEC 23271). The CLI is a "Common Language Infrastructure", serving as a platform to which dozens of different programming languages can be bound. For example, CLI bindings are available in languages such as Cobol, C#, Java, Eiffel, Mercury, Perl, and many others. The CLI is a "large" technology; it embodies many features such as classes, inheritance, visibility, constructors, conversion functions, virtual method lookup, etc (as touched upon in the Comment). In each instance where the CLI features follow a different semantics than a specific programming language, a "confusion" can arise in that instance. But the implication of the Comment is that the semantics of the programming language should "trump" the semantics of the underlying platform. Such an approach is not helpful to the individual programmer, nor is it helpful to the eventual consumer of the CLI capabilities written in the specific programming language. When the CLI classes have the same semantics, regardless of the programming language that produced them, then the programmer making use of those classes must know only two sets of rules: the "native" rules of the particular programming language, and the "common" CLI rules for CLI classes. But if CLI classes produced by C++/CLI were to embody significantly different rules from other CLI classes, then a programmer (using e.g. Cobol) would be required to know the "native" rules, the "usual" CLI rules, and the "special CLI from C++" rules.

Some bindings are achieved as a layered "bolt-on", often relying upon APIs and/or class libraries. But other bindings are achieved through an "integrated" implementation using extensions to the language, for example the SQLJ binding for SQL in Java. The earliest bindings for CLI in C++ (the so-called "managed C++") were of the "bolt-on" variety, and the language users considered the result to be "ugly" and cumbersome to use. It took about three years to develop Ecma C++/CLI, an "integrated" binding, and the C++ programmers who have used the resulting C++/CLI have commented positively upon the improvement.

There is no reason to expect that programmers will mistake C++/CLI features for ISO C++ features any more than they occasionally mistake extensions in specific compilers (e.g., Borland, Gnu, and Microsoft) for ISO C++ features. Rather, having these extensions as a named standard instead of unnamed vendor extensions serves to help distinguish them more clearly from ISO C++.

For its part, the C++/CLI standard is clear that C++/CLI is not C++; quoting from clause 1 of Standard ECMA-372: "C++/CLI is an extension of the C++ programming language as described in ISO/IEC 14882:2003, *Programming languages — C++*. In addition to the facilities provided by C++, C++/CLI provides additional keywords, classes, exceptions, namespaces, and library facilities, as well as garbage collection."

Quoting further from clause 3 (normative references):

“ISO/IEC 14882:2003, *Programming languages — C++*. [Note: Revision of the C++ Standard is currently underway, and changes proposed in that revision will affect future versions of this C++/CLI standard. For an example, see §9.1.1. *end note*]”

And from clause 4:

“Terms not defined in this Standard are to be interpreted according to the C++ Standard, ISO/IEC 14882:2003.”

C++/CLI exists to provide the support necessary to use C++ as a first-class language in a CLI environment, and the whole point of C++/CLI is to ensure that ISO C++ is not marginalized or rendered inapplicable on a major platform (ISO CLI), as it clearly was and would have continued to have been without C++/CLI.

Additional wording of this sort can be added, if the NBs have suggestions.

or add unnecessary overhead when trying to write C++ code usable with C++ and C++/CLI.

Examples of “unnecessary overhead” have not been seen yet. Perhaps Germany could clarify this comment.

Below are a few examples although, if there were sufficient time for a thorough analysis of the C++/CLI document, more could probably be found.

Ecma has granted unprecedented liaison latitude, and given all WG21 members full access to TG5 meetings and documents since 2003. All TG5 documents, including working drafts and meeting minutes, have been proactively put in WG21 mailings, and WG21 members have been providing feedback to TG5 since 2003 based on their analysis of C++/CLI drafts and change proposal papers.

For writing templates, it is essential to use one notation for all applicable types. For templates which are not involved in life-time management of objects, e.g. algorithmic functions, one template should be sufficient but the standard is silent about how to cope with the different pointer and reference notations for managed and unmanaged classes, implying that the user is forced to write unnecessarily non-portable template code.

This portion of the Comment states a desirable design guideline, not an inherent rule of C++ semantics. Because TG5 agreed, in general, with its desirability; there are design features of C++/CLI which were adopted in order that templates could be written to accept managed (CLI) classes as well as un-managed (C++) classes. For example, it is for exactly this reason that standard C++/CLI incorporates a change (proposed by Bjarne Stroustrup, and with thanks to him) that handles (^) are dereferenced with unary * (not ^) so that the programmer can write a template that operates on any of pointers, handles, or smart pointers. For example:

```
template<class Ptr>
void CallFooIndirectly( Ptr p ) {
    p->foo();
}

class CFoo { public: void Foo() { } };
ref class RFoo { public: void Foo() { } };
```

```
int main() {  
    CFoo cfoo;  
    RFoo rfoo;  
    tr1::shared_ptr<CFoo> sp_cfoo( new CFoo );  
  
    CallFooIndirectly( &cfoo ); // ok, Ptr is CFoo*  
    CallFooIndirectly( %rfoo ); // ok, Ptr is RFoo^  
    CallFooIndirectly( sp_cfoo ); // ok, Ptr is tr1::shared_ptr<CFoo>  
}
```

This kind of general template would not have been possible if one were required to dereference a handle with unary ^.

However, in some cases when there is a semantically significant difference in the underlying behavior, it really provides no benefit to the template writer to pretend that a certain syntax will work for both kinds of classes.

Also, current C++/CLI adds considerable confusion by changing some of C++ rules when classes happen to be managed classes:

- Inheritance is no longer private by default but public.
- One argument constructors are no longer considered conversion functions in the absence of the "explicit" keyword (the keyword itself is ignored on constructors of managed classes).
- A different set of conversion functions is added.
- The virtual function look-up during construction and destruction differs between managed and unmanaged classes: for managed classes the looked up function is the one of the most derived class while for unmanaged classes it is the one of the class whose constructor or destructor is currently processed.

This has not been confusing in practice, including based on extensive user experience with a beta implementation of C++/CLI, and this feedback was considered several times by TG5 which decided the correct technical decision was to leave it this way.

The basic issue is that there are basic essential differences between C++ types and CLI types that must be surfaced (it's why "ref class" and "value class" were added as two new type categories). CLI types do not behave like C++ types, and there's no way around that; TG5 tried to find alternatives that would enforce C++ semantics, but couldn't (and anyway they wouldn't be observed by CLI types not authored in C++). Finally, it is precisely the lack of direct language expression of these essential behaviors of CLI types that made C++ unusable and irrelevant on the ISO CLI platform, until C++/CLI was created.

So these differences aren't a bug; they're an essential feature. Consider that ISO C++ and ISO CLI do these things differently, and C++/CLI cannot greatly change either ISO C++ or ISO CLI (though C++/CLI has influenced ISO CLI, which was an important goal of doing this work in Ecma where CLI was being revised).

The job of TG5 was to find a binding that would harmonize C++ and CLI and let both standards work together well despite some conflicting requirements. Both semantics had to be fully preserved – correct ISO C++ semantics for ISO C++ types, and correct ISO CLI semantics for ISO CLI types.

At the time this project was launched in 2003, participants described it as an attempt to develop a "binding" of C++ to CLI, and a minimal (if still substantial) set of extensions to support that environment. C++/CLI is intended to be upwardly compatible with Standard C++, and Ecma TG5 have gone to praiseworthy efforts to guarantee that standard-conforming C++ code will compile and run correctly in this environment.

Nevertheless, **we believe C++/CLI has effectively evolved into a language which is almost, but not quite, entirely unlike C++ as we know it. Significant differences can be found in syntax, semantics, idioms, and underlying object model.** It is as if an architect said, "we're going to bind a new loft conversion on to your house, but first please replace your foundations, and make all your doors open the other way."

This comment appears to be self-contradictory, in that it acknowledges the near-perfect compatibility of C++/CLI with C++ in avoiding any change in the meaning of a legal ISO C++ program, but then claims that C++/CLI changes the foundation. A correct analogy would be that there are already two buildings on adjacent lots, each with their own foundation and structure, and C++/CLI builds a bridge that connects the buildings so that people in either building can conveniently use the facilities in the other without going outside (e.g., into another programming language building that is well connected to the CLI building).

Also, we note that all of the major features of the current C++/CLI standard were presented to WG21 in essentially their current form in October 2003 (and consistently since then), so what was described in 2003 is what was actually delivered.

Future development of the standard language will be damaged, as there will be massive market resistance to adding any additional complexity on top, such as the changes planned for C++0x now undergoing development in WG21.

C++/CLI applies only to those using C++ for CLI development. But whether this is a real issue or not, the complexity will not be any lesser or greater, if C++/CLI is a standard approved by ISO/IEC JTC1, or remains only an Ecma standard.

The UK request that Ecma withdraw this document from fast-track voting and if they decide to re-submit it to JTC1 for fast-track processing, do so under a name which does not include "C++".

Ecma suggests that the determination of a name change is not appropriate for the 30-day "conflicts" ballot. In the general case, each national body will have its own view of appropriate names, and the regular ballot-resolution process provides the mechanism for all ballots to be considered, since in general some national bodies might change their vote to "no" if a proposed new name is unacceptable to that national body. However, the 30-day "conflicts" ballot provides no mechanism for receiving and reconciling all national body views. Perhaps the national bodies will eventually prefer some name such as "CLI++" that would presumably satisfy the criteria expressed by the UK, but the 30-day "conflicts" ballot is not the appropriate process to resolve this issue.