RTTI in current_exception

by T P K Healy

A C++ program can link with a shared library which has incomplete or inaccurate documentation for the exceptions it throws, or the program may link at runtime using '`LoadLibrary`' or '`dlopen`' with a library which it knows very little about. Code in the main program may handle an exception thrown from within the shared library, as follows:

```
#include <exception>  // exception, exception_ptr
#include <typeinfo>   // typeid, type_info
#include <new>        // bad_alloc

extern void SomeLibraryFunction(void) noexcept(false) {}

int main(void)
{
    try { SomeLibraryFunction(); }
    catch (std::bad_alloc const & ) { /* Do something */ }
    catch (std::exception const &e)
    {
        std::type_info const &ti = typeid(e);
        // As 'std::exception' has a virtual destructor, it is a polymorphic
        // type, and so 'typeid' will give us the RTTI of the derived class
    }
    catch (...)
    {
        std::exception_ptr const p = std::current_exception();
        // We have no idea the type of the object which was thrown ? ? ?
    }
}
```

I propose that the RTTI of the object which was thrown should be obtainable inside the body of the '`catch (...)`', in one of these three ways:

```
catch (...)
{
    /* Solution No. 1 */ std::type_info const &ti = std::current_exception_typeid();
    std::exception_ptr const p = std::current_exception();
    /* Solution No. 2 */ std::type_info const &ti = p->ti;
    /* Solution No. 3 */ std::type_info const &ti = std::exception_ptr_typeid(p);
}
```

This proposed feature would not be needed if we could only throw objects derived from '`std::exception`', however since we can throw any type (for example `int`, `std::complex<long>`), it is fitting that we should be able to get the RTTI inside '`catch (...)`', otherwise the C++26 standard should deprecate the throwing of an object which is not derived from '`std::exception`'.