CtrlCHandler Helper Class

Previous Next

The C++ class `CtrlCHandler` provides a portable abstraction to handle CTRL-C and similar signals sent to a C++ process. On Windows, `CtrlCHandler` is a wrapper for `SetConsoleCtrlHandler`; on POSIX platforms, it handles `SIGHUP`, `SIGTERM` and `SIGINT` with a dedicated thread that waits for these signals using `sigwait`. Signals are handled by a callback function that you implement and register. The callback is a simple function that takes an `int` (the signal number) and returns `void`; it must not throw any exception:

C++11

```
namespace Ice
{
    class CtrlCHandler
    {
    public:
        explicit CtrlCHandler(std::function<void(int)> = nullptr);
        ~CtrlCHandler();

        std::function<void(int)> setCallback(std::function<void(int>));
        std::function<void(int)> getCallback() const;
    };
}
```

C++98

```
namespace Ice
{
    typedef void (*CtrlCHandlerCallback)(int);

    class CtrlCHandler
    {
    public:
        explicit CtrlCHandler(CtrlCHandlerCallback = 0);
        ~CtrlCHandler();

        CtrlCHandlerCallback setCallback(CtrlCHandlerCallback);
        CtrlCHandlerCallback getCallback() const;
    };
}
```

The member functions of `CtrlCHandler` behave as follows:

- `CtrlCHandler`
  Constructs an instance with optionally a callback function. Only one instance of `CtrlCHandler` can exist in a process at a given moment in time. On POSIX platforms, the constructor masks `SIGHUP`, `SIGTERM` and `SIGINT`, then starts a thread that waits for these signals using `sigwait`. For signal masking to work properly, it is imperative that the `CtrlCHandler` instance be created before starting any thread, and in particular before initializing an Ice communicator.
- `~CtrlCHandler`
  Destroys the instance, after which the default signal processing behavior is restored on Windows (`TerminateProcess`). On POSIX platforms, the "sigwait" thread is terminated and joined, but the signal mask remains unchanged, so subsequent signals are ignored.
- `setCallback`
  Sets a new callback function, and returns the old callback function.
- `getCallback`
  Gets the current callback function.

It is legal specify `nullptr` for the callback function, in which case signals are caught and ignored until a non-null callback function is set.

A typical use for `CtrlCHandler` is to shutdown a communicator in an Ice server. For example, you could use it as follows:

C++11

```
int
main(int argc, char* argv[])
{
    Ice::CtrlCHandler ctrlCHandler;
    try
    {
        Ice::CommunicatorHolder ich(argc, argv);
        ctrlCHandler.setCallback([communicator = ich.communicator()](int signal) // C++14 syntax
                                 {
                                     cerr << "caught signal " << signal << ", shutting down communicator" <<
endl;
                                     try
                                     {
                                         communicator->shutdown();
                                     }
                                     catch(const Ice::CommunicatorDestroyedException&)
                                     {
                                         // ignored
                                     }
                                 });
        auto adapter =
            ich->createObjectAdapterWithEndpoints("Hello", "default -h localhost -p 10000");
        adapter->add(make_shared<HelloI>(), Ice::stringToIdentity("hello"));
        adapter->activate();
        ich->waitForShutdown();
    }
    catch(const std::exception& ex)
    {
        cerr << ex.what() << endl;
        return 1;
    }
    return 0;
}
```

C++98

```
Ice::CommunicatorPtr communicator;
void shutdownCommunicator(int);

int
main(int argc, char* argv[])
{
    Ice::CtrlCHandler ctrlCHandler;
    try
    {
        Ice::CommunicatorHolder ich(argc, argv);
        communicator = ich.communicator();
        ctrlCHandler.setCallback(shutdownCommunicator);

        Ice::ObjectAdapterPtr adapter =
            ich->createObjectAdapterWithEndpoints("Hello", "default -h localhost -p 10000");
        adapter->add(new HelloI, Ice::stringToIdentity("hello"));
        adapter->activate();
        ich->waitForShutdown();
    }
    catch(const std::exception& ex)
    {
        cerr << ex.what() << endl;
        return 1;
    }
    return 0;
}

void
shutdownCommunicator(int signal)
{
    cerr << "caught signal " << signal << ", shutting down communicator" << endl;
    try
    {
        communicator->shutdown();
    }
    catch(const Ice::CommunicatorDestroyedException&)
    {
        // ignored
    }
}
```

Back to Top ^

See Also

- Application Helper Class

Previous Next