Initialization in C-Sharp

Previous Next

Every Ice-based application needs to initialize the Ice run time, and this initialization returns an `Ice.Communicator` object.

A `Communicator` is a local C# object that represents an instance of the Ice run time. Most Ice-based applications create and use a single `Communicator` object, although it is possible and occasionally desirable to have multiple `Communicator` objects in the same application.

You initialize the Ice run time by calling `Ice.Util.initialize`, for example:

C#

```
public static void Main(string[] args)
{
    Ice.Communicator communicator = Ice.Util.initialize(ref args);
    ...
}
```

`Ice.Util.initialize` accepts the argument vector that is passed to `Main` by the operating system. The method scans the argument vector for any command-line options that are relevant to the Ice run time; any such options are removed from the argument vector so, when `Ice.Util.initialize` returns, the only options and arguments remaining are those that concern your application. If anything goes wrong during initialization, `initialize` throws an exception.

Before leaving your `Main` method, you must call `Communicator.destroy`. The `destroy` operation is responsible for finalizing the Ice run time. In particular, in an Ice server, `destroy` waits for any operation implementations that are still executing to complete. In addition, `destroy` ensures that any outstanding threads are joined with and reclaims a number of operating system resources, such as file descriptors and memory. Never allow your `Main` method to terminate without calling `destroy` first.

The general shape of our `Main` method becomes:

C#

```
using System;

public class App
{
    public static int Main(string[] args)
    {
        int status = 0;
        Ice.Communicator communicator = null;

        try
        {
            // correct but suboptimal, see below
            communicator = Ice.Util.initialize(ref args);
            // ...
        }
        catch(Exception ex)
        {
            Console.Error.WriteLine(ex);
            status = 1;
        }

        if(communicator != null)
        {
            // correct but suboptimal, see below
            communicator.destroy();
        }
        return status;
    }
}
```

This code is a little bit clunky, as we need to make sure the communicator gets destroyed in all paths, including when an exception is thrown.

Fortunately, the `Ice.Communicator` interface implements `IDisposable`: this allows us to call `initialize` in a `using` statement, which disposes of (destroys) the communicator automatically, without an explicit call to the `destroy` method.

The preferred way to initialize the Ice run time in C# is therefore:

C#

```
using System;

public class App
{
    public static int Main(string[] args)
    {
        try
        {
            using(Ice.Communicator communicator = Ice.Util.initialize(ref args))
            {
                // ...
            } // communicator is destroyed automatically here
        }
        catch(Exception ex)
        {
            Console.Error.WriteLine(ex);
            return 1;
        }
        return 0;
    }
}
```

Back to Top ^

See Also

- Communicators
- Communicator Initialization
- Application Helper Class

Previous Next