Command-Line Parsing and Initialization

On this page:

- Parsing Command Line Options
- The Ice.ProgramName Property

Parsing Command Line Options

When you initialize the loe run time by calling Ice::initialize (C++/Ruby), Ice.Util.initialize (Java/C#), Ice.initialize (Python), or Ice_initialize (PHP), you can pass an argument vector to the initialization call.

For C++, Ice::initialize accepts a C++ reference to argc:

```
namespace Ice {
    CommunicatorPtr initialize(int& argc, char* argv[]);
}
```

Ice::initialize parses the argument vector and initializes its property settings accordingly. In addition, it removes any arguments from argv that are property settings. For example, assume we invoke a server as:

```
$ ./server --myoption --Ice.Config=config -x a --Ice.Trace.Network=3 -y opt file
```

Initially, argc has the value 9, and argv has ten elements: the first nine elements contain the program name and the arguments, and the final element, argv[argc], contains a null pointer (as required by the ISO C++ standard). When Ice::initialize returns, argc has the value 7 and argv contains the following elements:

```
./server
--myoption
-x
a
-y
opt
file
0  # Terminating null pointer
```

This means that you should initialize the Ice run time before you parse the command line for your application-specific arguments. That way, the Ice-related options are stripped from the argument vector for you so you do not need to explicitly skip them. If you use the Ice::Application helper class, the run member function is passed an argument vector with the Ice-related options already stripped. The same is true for the runWithSession member function called by the Glacier2::Application helper class.

For Java, ${\tt Ice.Util.initialize}$ is overloaded. The signatures are:

Java

```
package Ice;
public final class Util {
    public static Communicator
    initialize();
    public static Communicator
    initialize(String[] args);
    public static Communicator
    initialize(StringSeqHolder args);
    public static Communicator
    initialize(InitializationData id);
    public static Communicator
    initialize(String[] args, InitializationData id);
    public static Communicator
    initialize(StringSeqHolder args, InitializationData id);
    // ...
}
```

The versions that accept an argument vector of type <code>String[]</code> do not strip lce-related options for you, so, if you use one of these methods, your code must ignore options that start with one of the preserved prefixes. The versions that accept a <code>StringSeqHolder</code> behave like the C++ version and strip the lce-related options from the passed argument vector.

In C#, the argument vector is passed by reference to the initialize method, allowing it to strip the Ice-related options:

namespace Ice { public sealed class Util { public static Communicator initialize(); public static Communicator initialize(ref string[] args); public static Communicator initialize(InitializationData id); public static Communicator initialize(ref string[] args, InitializationData id); // ... } }

The Python, Ruby, and PHP implementations of initialize have the same semantics as C++ and .NET; they expect the argument vector to be passed as a list from which all Ice-related options are removed.

If you use the Ice.Application helper class, the run method is passed the cleaned-up argument vector. The Ice.Application class is described separately for each language mapping.

The Ice. ProgramName Property

For C++, Python, and Ruby, initialize sets the Ice. ProgramName property to the name of the current program (argv[0]). In C#, initialize sets Ice. ProgramName to the value of System. AppDomain. CurrentDomain. FriendlyName. Your application code can read this property and use it for activities such as logging diagnostic or trace messages.

Even though Ice.ProgramName is initialized for you, you can still override its value from a configuration file or by setting the property on the command line.

For Java, the program name is not supplied as part of the argument vector — if you want to use the Ice.ProgramName property in your application, you must set it before initializing a communicator.

See Also

- Using Configuration Files
- Reading PropertiesCommunicator Initialization
- Glacier2 Helper Classes