Communicator Initialization

During the creation of a communicator, the Ice run time initializes a number of features that affect the communicator's operation. Once set, these features remain in effect for the life time of the communicator, that is, you cannot change these features after you have created a communicator. Therefore, if you want to customize these features, you must do so when you create the communicator.

The following features can be customized at communicator creation time:

- the property set
- the logger object
- the instrumentation observer
- the narrow and wide string converters (C++ only)
- the thread notification hook
- the dispatcher
- the compact ID resolver (used by the streaming interfaces when extracting Ice objects)
- the class loader (Java only)

To establish these features, you initialize a structure or class of type `InitializationData` with the relevant settings. For C++ the structure is defined as follows:

C++

```
namespace Ice {
    struct InitializationData {
        PropertiesPtr properties;
        LoggerPtr logger;
        Instrumentation::CommunicatorObserverPtr observer;
        StringConverterPtr stringConverter;
        WstringConverterPtr wstringConverter;
        ThreadNotificationPtr threadHook;
        DispatcherPtr dispatcher;
        CompactIdResolverPtr compactIdResolver;
    };
}
```

For languages other than C++, `InitializationData` is a class with all data members public. (The data members supported by this class vary with each language mapping.)

For C++, `Ice::initialize` is overloaded as follows:

C++

```
namespace Ice {
    CommunicatorPtr initialize(int&, char*[],
                const InitializationData& = InitializationData(),
                Int = ICE_INT_VERSION);
    CommunicatorPtr initialize(StringSeq&,
                const InitializationData& = InitializationData(),
                Int = ICE_INT_VERSION);
    CommunicatorPtr initialize(
                const InitializationData& = InitializationData()
                Int = ICE_INT_VERSION);
}
```

The versions of `initialize` that accept an argument vector look for Ice-specific command-line options and remove them from the argument vector, as described in the C++ language mapping. The version without an `argc`/`argv` pair is useful if you want to prevent property settings for a program from being changed by command-line arguments — you can use the `Properties` interface for this purpose.

To set a feature, you set the corresponding field in the `InitializationData` structure and pass the structure to `initialize`. For example, to establish a custom logger of type `MyLogger`, you can use:

```
Ice::InitializationData id;
id.logger = new MyLoggerI;
Ice::CommunicatorPtr ic = Ice::initialize(argc, argv, id);
```

For Java, C#, and Objective-C, `Ice.Util.initialize` is overloaded similarly (as is `Ice.initialize` for Python, `Ice::initialize` for Ruby, and `Ice_initialize` for PHP), so you can pass an `InitializationData` instance either with or without an argument vector.

For Objective-C, the method name is `createCommunicator`.

Note that you must supply an argument vector if you want `initialize` to look for a configuration file in the `ICE_CONFIG` environment variable.

See Also

- The Server-Side main Function in C++
- The Server-Side main Function in Objective-C
- Command-Line Parsing and Initialization
- The Properties Interface