Writing an Ice Application with MATLAB

Previous Next

This page shows how to create an Ice client application with MATLAB.

On this page:

Compiling a Slice Definition for MATLAB

The first step in creating our MATLAB application is to compile our Slice definition to generate MATLAB proxies. You can compile the definition as follows:

```
$ slice2matlab Printer.ice
```

The `slice2matlab` compiler produces a single source file, `PrinterPrx.rb`, from this definition. The exact contents of the source file do not concern us for now — it contains the generated code that corresponds to the `Printer` interface we defined in `Printer.ice`.

Back to Top ^

Writing a Client in MATLAB

The client code, in `Client.m`, is shown below in full:

MATLAB

```
function Client()
    if ~libisloaded('ice')
        loadlibrary('ice', @iceproto)
    end
    communicator = Ice.initialize();
    cleanup = onCleanup(@() communicator.destroy());
    base = communicator.stringToProxy('SimplePrinter:default -h localhost -p 10000');
    printer = Demo.PrinterPrx.checkedCast(base);
    if isempty(printer)
        throw(MException('Client:RuntimeError', 'Invalid proxy'));
    end

    printer.printString('Hello World!');
end
```

The function begins by ensuring that the Ice library is loaded. The remainder of the function goes through the following steps:

1. We initialize the Ice run time by calling `Ice.initialize`. The call to `initialize` returns an `Ice.Communicator` reference, which is the main object in the Ice run time.
2. We register a "clean up" function that will be called when the `cleanup` variable is reclaimed and ensures that the communicator is destroyed before the program terminates. Doing this is essential in order to correctly finalize the Ice run time: the program *must* call `destroy` on any communicator it has created; otherwise, undefined behavior results.
3. The next step is to obtain a proxy for the remote printer. We create a proxy by calling `stringToProxy` on the communicator, with the string `'SimplePrinter:default -p 10000'`. Note that the string contains the object identity and the port number that were used by the server. (Obviously, hard-coding object identities and port numbers into our applications is a bad idea, but it will do for now; we will see more architecturally sound ways of doing this when we discuss IceGrid.)
4. The proxy returned by `stringToProxy` is of type `Ice.ObjectPrx`, which is at the root of the inheritance tree for interfaces and classes. But to actually talk to our printer, we need a proxy for a `Demo::Printer` interface, not an `Object` interface. To do this, we need to do a down-cast by calling `Demo.PrinterPrx.checkedCast`. A checked cast sends a message to the server, effectively asking "is this a proxy for a `Demo::Printer` interface?" If so, the call returns a proxy of type `Demo.PrinterPrx`; otherwise, if the proxy denotes an interface of some other type, the call returns an empty array.
5. We test that the down-cast succeeded and, if not, throw an exception that terminates the client.
6. We now have a live proxy in our address space and can call the `printString` method, passing it the time-honored `'Hello World!'` string. The server prints that string on its terminal.

Back to Top ^

Running the Client in MATLAB

The server must be started before the client. Since Ice for MATLAB does not support server-side behavior, we need to use a server from another language mapping. In this case, we will use the C++ server:

```
$ server
```

At this point, we won't see anything because the server simply waits for a client to connect to it. We run the client in a MATLAB console:

```
>> Client()
```

The client runs and exits without producing any output; however, in the server window, we see the `"Hello World!"` message that is produced by the printer. To get rid of the server, we interrupt it on the command line.

If anything goes wrong, the client will print an error message. For example, if we run the client without having first started the server, we get something like the following:

```
Error using Demo.PrinterPrx.checkedCast (line 59)
::Ice::ConnectionRefusedException

Error in Client (line 8)
    printer = Demo.PrinterPrx.checkedCast(base);
```

Note that, to successfully run the client, MATLAB must be able to locate Ice for MATLAB. See the Ice for MATLAB installation instructions for more information.

Back to Top ^

See Also

- Client-Side Slice-to-MATLAB Mapping
- IceGrid

Previous Next