# Slice Compilation

On this page:


- Compilation
- Single Development Environment for Client and Server
- Different Development Environments for Client and Server

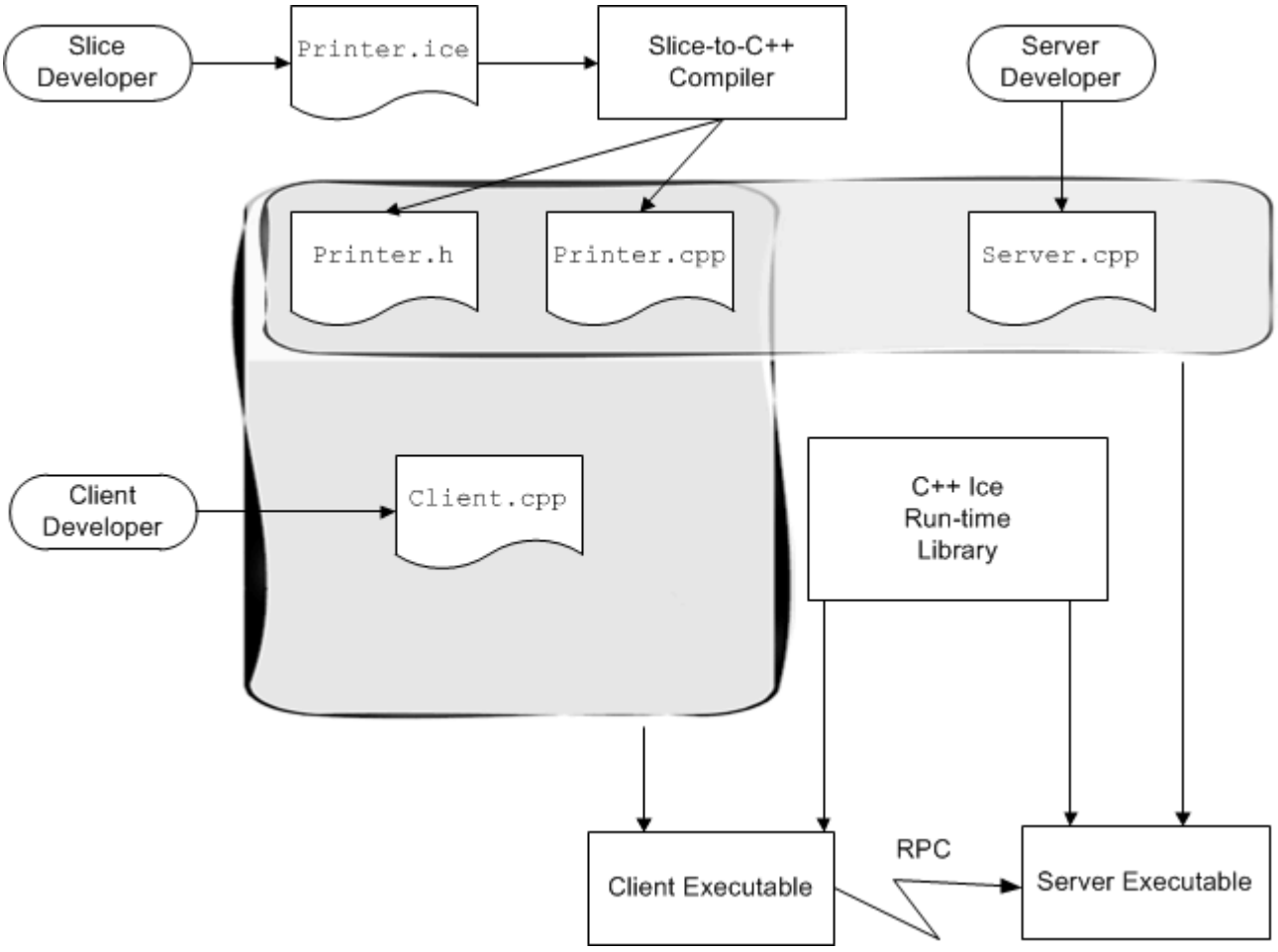

## Compilation

A Slice compiler produces source files that must be combined with application code to produce client and server executables.

The outcome of the development process is a client executable and a server executable. These executables can be deployed anywhere, whether the target environments use the same or different operating systems and whether the executables are implemented using the same or different languages. The only constraint is that the host machines must provide the necessary run-time environment, such as any required dynamic libraries, and that connectivity can be established between them.

## Single Development Environment for Client and Server

The figure below shows the situation when both client and server are developed in C++. The Slice compiler generates two files from a Slice definition in a source file `Printer.ice`: a header file (`Printer.h`) and a source file (`Printer.cpp`)




.
*Development process if client and server share the same development environment.*

- The `Printer.h` header file contains definitions that correspond to the types used in the Slice definition. It is included in the source code of both client and server to ensure that client and server agree about the types and interfaces used by the application.

- The `Printer.cpp` source file provides an API to the client for sending messages to remote objects. The client source code (`Client.cpp`, written by the client developer) contains the client-side application logic. The generated source code and the client code are compiled and linked into the client executable.

The `Printer.cpp` source file also contains source code that provides an up-call interface from the Ice run time into the server code written by the developer and provides the connection between the networking layer of Ice and the application code. The server implementation file (`Server.cpp`, written by the server developer) contains the server-side application logic (the object implementations, properly termed *servants*). The generated source code and the implementation source code are compiled and linked into the server executable.

Both client and server also link with an Ice library that provides the necessary run-time support.

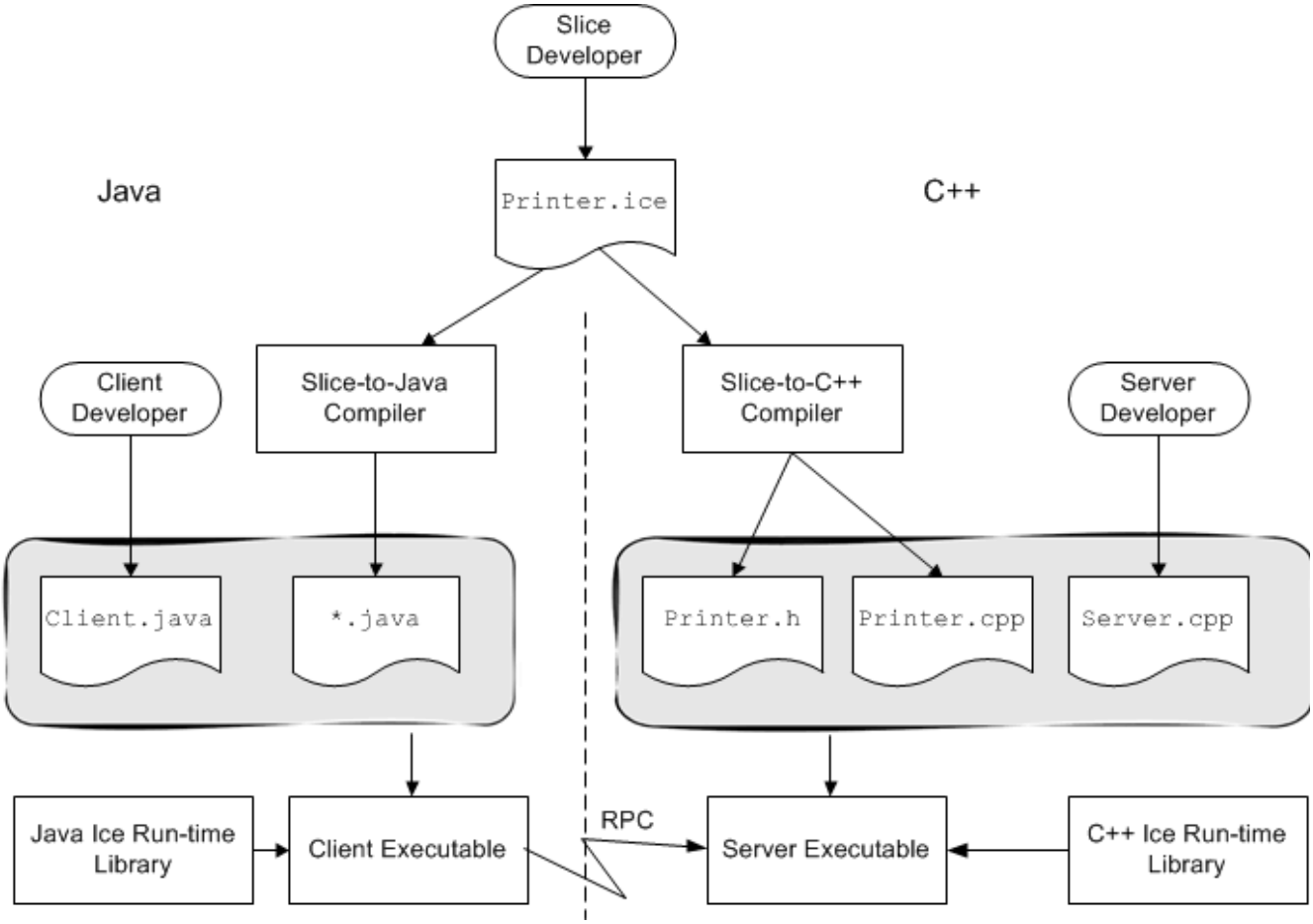
You are not limited to a single implementation of a client or server. For example, you can build multiple servers, each of which implements the same interfaces but uses different implementations (for example, with different performance characteristics). Multiple such server implementations can coexist in the same system. This arrangement provides one fundamental scalability mechanism in Ice: if you find that a server process starts to bog down as the number of objects increases, you can run an additional server for the same interfaces on a different machine. Such *federated* servers provide a single logical service that is distributed over a number of processes on different machines. Each server in the federation implements the same interfaces but hosts different object instances. (Of course, federated servers must somehow ensure consistency of any databases they share across the federation.)

Ice also provides support for *replicated* servers. Replication permits multiple servers to each implement the same set of object instances. This improves performance and scalability (because client load can be shared over a number of servers) as well as redundancy (because each object is implemented in more than one server).

# Different Development Environments for Client and Server

Client and server cannot share any source or binary components if they are developed in different languages. For example, a client written in Java cannot include a C++ header file.

This figure shows the situation when a client written in Java and the corresponding server is written in C++. In this case, the client and server developers are completely independent, and each uses his or her own development environment and language mapping. The only link between client and server developers is the Slice definition each one uses.




*Development process for different development environments.*

For Java, the slice compiler creates a number of files whose names depend on the names of various Slice constructs. (These files are collectively referred to as `*.java` in the above figure.)

## See Also

- Using the Slice Compilers