Communicator

The main entry point to the Ice run time is represented by the local Slice interface `Ice::Communicator`. An instance of `Ice::Communicator` is associated with a number of run-time resources:

- Client-side thread pool
  The client-side thread pool is used to process replies to asynchronous method invocations (AMI), to avoid deadlocks in callbacks, and to process incoming requests on bidirectional connections.
- Server-side thread pool
  Threads in this pool accept incoming connections and handle requests from clients.
- Configuration properties
  Various aspects of the Ice run time can be configured via properties. Each communicator has its own set of such configuration properties.
- Object factories
  In order to instantiate classes that are derived from a known base type, the communicator maintains a set of object factories that can instantiate the class on behalf of the Ice run time. Object factories are discussed in each client-side language mapping.
- Logger object
  A logger object implements the `Ice::Logger` interface and determines how log messages that are produced by the Ice run time are handled.
- Default router
  A router implements the `Ice::Router` interface. Routers are used by Glacier2 to implement the firewall functionality of Ice.
- Default locator
  A locator is an object that resolves an object identity to a proxy. A locator object is implemented by a location service.
- Plug-in manager
  Plug-ins are objects that add features to a communicator. For example, IceSSL is implemented as a plug-in. Each communicator has a plug-in manager that implements the `Ice::PluginManager` interface and provides access to the set of plug-ins for a communicator.
- Object adapters
  Object adapters dispatch incoming requests and take care of passing each request to the correct servant.

Object adapters and objects that use different communicators are completely independent from each other. Specifically:

- Each communicator uses its own thread pool. This means that if, for example, one communicator runs out of threads for incoming requests, only objects using that communicator are affected. Objects using other communicators have their own thread pool and are therefore unaffected.
- Collocated invocations across different communicators are not optimized, whereas collocated invocations using the same communicator bypass much of the overhead of call dispatch.

Typically, servers use only a single communicator but, occasionally, multiple communicators can be useful. For example, IceBox, uses a separate communicator for each Ice service it loads to ensure that different services cannot interfere with each other. Multiple communicators are also useful to avoid thread starvation: if one service runs out of threads, this leaves the remaining services unaffected.

See Also

- Communicator Shutdown and Destruction
- Obtaining Proxies
- Converting Proxies to Strings
- Creating an Object Adapter
- Object Identity
- Value Factories
- Properties and Configuration
- Implicit Request Contexts
- Plug-in Facility
- Logger Facility
- Locators
- Routers
- Logger Facility
- Administrative Facility
- Instrumentation Facility