Glacier2 Helper Classes

Ice includes a number of helper classes to help you build robust Glacier2 clients.

The `Glacier2::Application` Class

You may already be familiar with the `Ice::Application` class, which encapsulates some basic Ice functionality such as communicator initialization, communicator destruction, and proper handling of signals and exceptions. The `Glacier2::Application` extends `Ice::Application` to add functionality that is commonly needed by Glacier2 clients:

- Keeps a session alive by periodically sending "ping" requests from a background thread
- Automatically restarts a session if a failure occurs
- Optionally creates an object adapter for callbacks
- Destroys the session when the application completes

The C++ definition of `Glacier2::Application` is shown below. (The Java and C# versions offer identical functionality so we do not show them here.)

C++

```
namespace Glacier2 {

class Application : public Ice::Application {
public:
    Application();
    Application(Ice::SignalPolicy policy);

    virtual int runWithSession(int argc, char* argv[]) = 0;

    virtual Glacier2::SessionPrx createSession() = 0;

    virtual void sessionDestroyed();

    static Glacier2::RouterPrx router();

    static Glacier2::SessionPrx session();

    void restart();

    std::string categoryForClient();

    Ice::Identity createCallbackIdentity(const std::string& name);

    Ice::ObjectPrx addWithUUID(const Ice::ObjectPtr& servant);

    Ice::ObjectAdapterPtr objectAdapter();
};

}
```

The following methods are supported:

- `Application()`
  Instantiating the class using its default constructor has the same semantics as calling `Application(Ice::HandleSignals)`.

- `Application(Ice::SignalPolicy policy)`
  This constructor allows you to indicate whether the class should handle signals. The `SignalPolicy` enumeration contains two enumerators: `HandleSignals` and `NoSignalHandling`. If you specify `HandleSignals`, the class automatically shuts down or destroys

its communicator upon receipt of certain signals. Refer to the `Ice::Application` documentation in the relevant language mapping chapter for more information on signal handling.

- `int runWithSession(int argc, char* argv[])`
  This method must be overridden by a subclass and represents the "main loop" of the application. It is called after the communicator has been initialized and the Glacier2 session has been established. The argument vector passed to this method contains the arguments passed to `Application::main` with all Ice-related options removed. The implementation of `runWithSession` must return zero to indicate success and non-zero to indicate failure; the value returned by `runWithSession` becomes the return value of `Application::main`.

  `runWithSession` can call `restart` to restart the session. This destroys the current session, creates a new session (by calling `createSession`), and calls `runWithSession` again. The `Application` base class also restarts the session if `runWithSession` raises (or allows to be raised) any of the following exceptions:

  - `Ice::ConnectionLostException`
  - `Ice::ConnectionRefusedException`
  - `Ice::RequestFailedException`
  - `Ice::TimeoutException`
  - `Ice::UnknownLocalException`

  All other exceptions cause the current session to be destroyed without restarting.

- `Glacier2::SessionPrx createSession()`
  This method must be overridden by a subclass to create the application's Glacier2 session. A successful call to `createSession` is followed by a call to `runWithSession`. The application terminates if `createSession` raises (or allows to be raised) an `Ice::LocalException`.

- `void sessionDestroyed()`
  A subclass can optionally override this method to take action when connectivity with the Glacier2 router is lost.

- `Glacier2::RouterPrx router()`
  Returns the proxy for the Glacier2 router.

- `Glacier2::SessionPrx session()`
  Returns the proxy for the current session.

- `void restart()`
  Causes `Application` to destroy the current session, create a new session (by calling `createSession`), and start a new main loop (in `runWithSession`). This method does not return but rather raises a `RestartSessionException` that is trapped by `Application`.

- `std::string categoryForClient()`
  Returns the category to be used in the identities of all of the client's callback objects. Clients must use this category for the router to forward callback requests to the intended client. The method raises `SessionNotExistException` if no session is currently active.

- `Ice::Identity createCallbackIdentity(const std::string& name)`
  Creates a new Ice identity for a callback object with the given identity name.

- `Ice::ObjectPrx addWithUUID(const Ice::ObjectPtr& servant)`
  Adds a servant to the callback object adapter's Active Servant Map using a UUID for the identity name.

- `Ice::ObjectAdapterPtr objectAdapter()`
  Returns the object adapter used for callbacks, creating the object adapter if necessary.

Example

The Ice distribution includes an example in `demo/Glacier2/callback` that shows how to use the `Glacier2::Application` class.

GUI Helper Classes

The "main loop" design imposed by the `Glacier2::Application` class is not suitable for graphical applications, therefore Ice also includes a collection of Java and C# classes that better accommodate the needs of GUI programs:

- `Glacier2.SessionFactoryHelper`
  This class simplifies the task of creating a Glacier2 session. It provides overloaded `connect` methods that support the two authentication styles (user name/password and SSL credentials) accepted by the Glacier2 router, and returns an instance of `Glacier2.SessionHelper` for each new session.

- `Glacier2.SessionHelper`
  This class encapsulates a Glacier2 session and provides much of the same functionality as `Glacier2::Application`. The application must supply an instance of `Glacier2.SessionCallback` when creating the session; `SessionHelper` invokes this callback object when important events occur in the session's lifecycle.
- `Glacier2.SessionCallback`
  An application implements this interface to receive notification about session lifecycle events.

The classes are discussed further in the subsections below.

Example

You can find sample applications that make use of these classes in the `demo/Glacier2/chat` directory of your Ice distribution.

The `SessionFactoryHelper` Class

The `SessionFactoryHelper` class provides convenience methods for configuring the settings that are commonly used to create a Glacier2 session, such as the router's host and port number. Once the application has completed its configuration, it calls one of the `connect` methods to initialize a communicator, establish a Glacier2 session, and receive a `SessionHelper` object with which it can manage the new session. An application should create a new `SessionFactoryHelper` object for each router instance that it uses.

`SessionFactoryHelper` creates an `Ice.InitializationData` object if the application does not pass one to the `SessionFactoryHelper` constructor. `SessionFactoryHelper` also creates a new property set if necessary, and then sets some configuration properties required by Glacier2 clients. The resulting `InitializationData` object is eventually used in `connect` to initialize a new communicator.

The Java definition of `SessionFactoryHelper` is shown below (the C# version is nearly identical and is not shown here):

Java

```
package Glacier2;

public class SessionFactoryHelper {

    public SessionFactoryHelper(SessionCallback callback)
        throws Ice.InitializationException;

    public SessionFactoryHelper(Ice.InitializationData initData, SessionCallback callback)
        throws Ice.InitializationException;

    public SessionFactoryHelper(Ice.Properties properties, SessionCallback callback)
        throws Ice.InitializationException;

    public void setRouterIdentity(Ice.Identity identity);
    public Ice.Identity getRouterIdentity();

    public void setRouterHost(String hostname);
    public String getRouterHost();

    public void setSecure(boolean secure);
    public boolean getSecure();

    public void setTimeout(int timeoutMillisecs);
    public int getTimeout();

    public void setPort(int port);
    public int getPort();

    public Ice.InitializationData getInitializationData();

    public void setConnectContext(java.util.Map<String, String> ctx);

    public SessionHelper connect();
    public SessionHelper connect(String username, String password);
}
```

The following methods are supported:

- `SessionFactoryHelper(SessionCallback callback)`
  This constructor is useful when your application has no other configuration requirements. The constructor allocates an `InitializationData` object and a new property set. The callback argument must not be null.

- `SessionFactoryHelper(Ice.InitializationData initData, SessionCallback callback)`
  Use this constructor when you want to prepare your own instance of `InitializationData`. The `callback` argument must not be null.

- `SessionFactoryHelper(Ice.Properties properties, SessionCallback callback)`
  This constructor is convenient when you wish to supply an initial set of properties. The callback argument must not be null.

- `void setRouterIdentity(Ice.Identity identity)`
  `Ice.Identity getRouterIdentity()`
  Determines the object identity of the Glacier2 router. Note that this setting is only used if `Ice.Default.Router` is undefined.

- `void setRouterHost(String hostname)`
  `String getRouterHost()`
  Determines the host name of the Glacier2 router. Note that this setting is only used if `Ice.Default.Router` is undefined.

- `void setSecure(boolean secure)`
  `boolean getSecure()`
  Determines whether the connection to the Glacier2 router must be secure. Note that this setting is only used if `Ice.Default.Router` is undefined.

- `void setTimeout(int timeoutMillisecs)`
  `int getTimeout()`
  Determines the timeout setting (in milliseconds) for the connection to the Glacier2 router. No timeout is used if the argument is less than or equal to zero. Note that this setting is only used if `Ice.Default.Router` is undefined.

- `void setPort(int port)`
  `int getPort()`
  Determines the port on which the Glacier2 router is listening. Note that this setting is only used if `Ice.Default.Router` is undefined.

- `Ice.InitializationData getInitializationData()`
  Returns a reference to the `InitializationData` object that will be used during communicator initialization. If necessary, an application can make modifications to this object prior to calling `connect`.

- `void setConnectContext(java.util.Map<String, String> ctx)`
  Sets the request context to be used when creating a session. This method must be invoked prior to `connect`.

- `SessionHelper connect()`
  Initializes a communicator, creates a Glacier2 session using SSL credentials, and returns a new `SessionHelper` object. The `connected` method is invoked on the session callback if the session was created successfully, otherwise the callback's `connectFailed` method is invoked.

- `SessionHelper connect(String username, String password)`
  Initializes a communicator, creates a Glacier2 session using the given user name and password, and returns a new `SessionHelper` object. The `connected` method is invoked on the session callback if the session was created successfully, otherwise the callback's `connectFailed` method is invoked.

The `SessionHelper` Class

The `SessionHelper` class encapsulates a Glacier2 session and keeps the session alive by periodically "pinging" the router. `SessionHelper` also provides several convenience methods for common session-related actions:

Java

```
package Glacier2;

public class SessionHelper {

    public void destroy();

    public Ice.Communicator communicator();

    public String categoryForClient()
        throws SessionNotExistException;

    public Ice.ObjectPrx addWithUUID(Ice.Object servant)
```

```
        throws SessionNotExistException;

    public Glacier2.SessionPrx session()
        throws SessionNotExistException;

    public boolean isConnected();

    public Ice.ObjectAdapter objectAdapter()
        throws SessionNotExistException;
}
```

The following methods are supported:

- `void destroy()`
  Initiates the destruction of the Glacier2 session, including the communicator created by the `SessionHelper`. This is a non-blocking method that spawns a background thread to perform potentially blocking activities. `SessionHelper` invokes the `disconnected` method on the application's callback object to indicate the completion of the `destroy` request. A well-behaved application must wait for this call to `disconnected` before it terminates.
- `Ice.Communicator communicator()`
  Returns the communicator created by the `SessionHelper`.
- `String categoryForClient()`
  Returns the category that must be used in the identities of all callback objects. Raises `SessionNotExistException` if no session is currently active.
- `Ice.ObjectPrx addWithUUID(Ice.Object servant)`
  Adds a servant to the callback object adapter using a UUID for the identity name. Raises `SessionNotExistException` if no session is currently active.
- `Glacier2.SessionPrx session()`
  Returns a proxy for the Glacier2 session. Raises `SessionNotExistException` if no session is currently active.
- `boolean isConnected()`
  Returns true if the session is currently active, false otherwise.
- `Ice.ObjectAdapter objectAdapter()`
  Returns the callback object adapter, creating it if necessary. Raises `SessionNotExistException` if no session is currently active.

The `SessionCallback` Interface

An application must supply an instance of `SessionCallback` when instantiating a `SessionFactoryHelper` object. The callback methods allow the application to receive notification about events in the lifecycle of the session:

Java

```
package Glacier2;

public interface SessionCallback {

    void createdCommunicator(SessionHelper session);

    void connected(SessionHelper session)
        throws SessionNotExistException;

    void disconnected(SessionHelper session);

    void connectFailed(SessionHelper session, Throwable ex);
}
```

The following callback methods are supported:

- `void createdCommunicator(SessionHelper session)`
  Called after successfully initializing a communicator.

- `void connected(SessionHelper session)`
  Called after successfully establishing the Glacier2 session. The method can raise `SessionNotExistException` to force the new session to be destroyed.
- `void disconnected(SessionHelper session)`
  Called after the Glacier2 session is destroyed.
- `void connectFailed(SessionHelper session, Throwable ex)`
  Called if a failure occurred while attempting to establish a Glacier2 session.

See Also

- The Server-Side main Function in C++
- Callbacks through Glacier2
- Glacier2 Session Management