

# Glacier2 Application Class

 Previous

 Next

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 "heartbeat" requests
- Automatically restarts a session if a failure occurs
- Optionally creates an object adapter for callbacks
- Destroys the session when the application completes

Glacier2 Application is defined as follows :

**C++11**

```
namespace Glacier2
{
    class Application : public Ice::Application
    {
        public:

            using Ice::Application::Application;

            virtual std::shared_ptr<Glacier2::SessionPrx> createSession() = 0;
            virtual int runWithSession(int argc, char* argv[]) = 0;
            virtual void sessionDestroyed();

            static void restart();

            static std::shared_ptr<Glacier2::RouterPrx> router();
            static std::shared_ptr<Glacier2::SessionPrx> session();

            static std::string categoryForClient();
            static Ice::Identity createCallbackIdentity(const std::string&);

            static std::shared_ptr<Ice::ObjectPrx> addWithUUID(const std::shared_ptr<Ice::Object>& servant);
            static std::shared_ptr<Ice::ObjectAdapter> objectAdapter();

            ...
    };
}
```

**C++98**

```

namespace Glacier2
{
    class Application : public Ice::Application
    {
        public:

            Application(Ice::SignalPolicy signalPolicy = Ice::HandleSignals);

            virtual Glacier2::SessionPrx createSession() = 0;
            virtual int runWithSession(int argc, char* argv[]) = 0;
            virtual void sessionDestroyed();

            static void restart();

            static Glacier2::RouterPrx router();
            static Glacier2::SessionPrx session();

            static std::string categoryForClient();
            static Ice::Identity createCallbackIdentity(const std::string&);
            static Ice::ObjectPrx addWithUUID(const Ice::ObjectPtr& servant);
            static Ice::ObjectAdapterPtr objectAdapter();

            ...
    };
}

```

## C#

```

namespace Glacier2
{
    public abstract class Application : Ice.Application
    {
        public Application(Ice.SignalPolicy signalPolicy = Ice.SignalPolicy.HandleSignals) { ... }

        public abstract Glacier2.SessionPrx createSession();
        public abstract int runWithSession(string[] args);
        public virtual void sessionDestroyed() {}

        public static void restart() { ... }

        public static Glacier2.RouterPrx router() { ... }
        public static SessionPrx session() { ... }

        public static string categoryForClient() { ... }
        public static Ice.Identity createCallbackIdentity(string name) { ... }
        public static Ice.ObjectPrx addWithUUID(Ice.Object servant) { ... }
        public static Ice.ObjectAdapter objectAdapter() { ... }

        ...
    }
}

```

## Java

```

package com.zeroc.Glacier2;

public abstract class Application extends com.zeroc.Ice.Application
{
    public static class RestartSessionException extends Exception
    {
    }
    public Application() { ... }
    public Application(com.zeroc.Ice.SignalPolicy signalPolicy) { ... }

    public abstract com.zeroc.Glacier2.SessionPrx createSession();
    public abstract int runWithSession(String[] args) throws RestartSessionException;
    public void sessionDestroyed() {}

    public static void restart() throws RestartSessionException { ... }

    public static com.zeroc.Glacier2.RouterPrx router() { ... }
    public static com.zeroc.Glacier2.SessionPrx session() { ... }

    public static String categoryForClient() throws SessionNotExistException { ... }
    public static com.zeroc.Ice.Identity createCallbackIdentity(String name) throws SessionNotExistException {
    }
    public static com.zeroc.Ice.ObjectPrx addWithUUID(com.zeroc.Ice.Object servant) throws
SessionNotExistException { ... }
    public static com.zeroc.Ice.ObjectAdapter objectAdapter() throws SessionNotExistException { ... }

    ...
}

```

#### Java Compat

```

package Glacier2;

public abstract class Application extends Ice.Application
{
    public static class RestartSessionException extends Exception
    {
    }
    public Application() { ... }
    public Application(Ice.SignalPolicy signalPolicy) { ... }

    public abstract Glacier2.SessionPrx createSession();
    public abstract int runWithSession(String[] args) throws RestartSessionException;
    public void sessionDestroyed() {}

    public static void restart() throws RestartSessionException { ... }

    public static Glacier2.RouterPrx router() { ... }
    public static Glacier2.SessionPrx session() { ... }

    public static String categoryForClient() throws SessionNotExistException { ... }
    public static Ice.Identity createCallbackIdentity(String name) throws SessionNotExistException { ... }
    public static Ice.ObjectPrx addWithUUID(Ice.Object servant) throws SessionNotExistException { ... }
    public static Ice.ObjectAdapter objectAdapter() throws SessionNotExistException { ... }

    ...
}

```

#### Python

```

# in Glacier2 module
class Application(Ice.Application):

    def __init__(self, signalPolicy=0):

        def createSession(self, args):
        def runWithSession(self, args):
        def sessionDestroyed(self):

        def restart(self):
        def router(self):
        def session(self):

        def categoryForClient(self):
        def createCallbackIdentity(self, name):
        def addWithUUID(self, servant):
        def objectAdapter(self):

```

The intent of this class is that you specialize `Glacier2 Application` and implement the pure virtual functions or abstract methods `createSession` and `runWithSession` in your derived class.

This Application class provides the following functions or methods:

- **Constructor**  
Like the Ice Application constructor, you can construct a Glacier2 application that handles signals (the default) or does not.
- **createSession**  
This pure virtual function or abstract 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` throws an `Ice LocalException`.
- **runWithSession**  
This pure virtual function or abstract 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 arguments passed to `runWithSession` contain the arguments passed to `main` on `Application` 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 `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` throws any of the following Ice exceptions:



Unknown macro: 'ul'

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

- **sessionDestroyed**  
A subclass can optionally override this function or method to take action when connectivity with the Glacier2 router is lost.
- **router**  
Returns the proxy for the Glacier2 router.
- **session**  
Returns the proxy for the current session.
- **restart**  
Causes `Application` to destroy the current session, create a new session (by calling `createSession`), and start a new main loop (in `runWithSession`). This function or method does not return but rather throws a `RestartSessionException`, later caught by `Application`.
- **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. This function or method raises `SessionNotExistException` if no session is currently active.
- **createCallbackIdentity**  
Creates a new Ice identity for a callback object with the given identity name.
- **addWithUUID**  
Adds a servant to the callback object adapter's Active Servant Map using a UUID for the identity name.
- **objectAdapter**  
Returns the object adapter used for callbacks, creating the object adapter if necessary.

## Application and Threads

Just like Ice Application, Glacier2 Application should be considered single-threaded until you call `main`. Then, in `createSession` and `runWithSession`, you can call concurrently Application's functions or methods from multiple threads. If you create threads, you must join them before or when `runWithSession` returns.

The thread you use to call `main` is used to initialize the Communicator and then to call `createSession` and later `runWithSession`. On Linux and macOS with C++, this thread must be the only thread of the process at the time you call `main` for signal handling to operate correctly.

[Back to Top ^](#)

### See Also

- [Application Helper Class](#)
- [Callbacks through Glacier2](#)
- [Glacier2 Session Management](#)

