

## Re: APQ

**Source:** <http://coding.derkeiler.com/Archive/Ada/comp.lang.ada/2004-12/0354.html>

---

**From:** Dmitry A. Kazakov ([mailbox\\_at\\_dmitry-kazakov.de](mailto:mailbox_at_dmitry-kazakov.de))

**Date:** 12/17/04

Date: Fri, 17 Dec 2004 14:54:56 +0100

On Thu, 16 Dec 2004 10:31:49 +1100, Brian May wrote:

- > 1. Decide on or write a "smart pointer" library. ie. a library that
- > manages pointers based on a reference count. Obviously the license
- > must be compatible with the license of APQ.

I have one in <http://www.dmitry-kazakov.de/ada/components.htm>

- > 3. Create new Database\_Type. This type should not be
- > tagged. Applications will use this instead of the
- > Root\_Connection\_Type. It has the following data:
- >
- > \* Smart pointer to Root\_Connection\_Type'Class. This is dynamically
- > allocated when the connection is created.

It should \*be\* a smart pointer. Generally with components it goes as follows:

```
package APQ.Handles is
  type Database_Type is private;
  -- Here all visible methods
  function Connect (...) return Data_Base_Type;
  ...
private
  -- Wrapper type for Root_Connection_Type'Class, to implement
  -- mix-in. Necessary, only because Root_Connection_Type is not
  -- a descendant of Object.Entity. Ptr points to the connection
  -- object allocated by Connect. Data_Base_Object itself is also
  -- allocated by Connect. It is destroyed as soon as the
  -- last handle to it disappears.
  --
  type Root_Connection_Ptr is access Root_Connection_Type'Class;
  type Data_Base_Object is new Object.Entity with record
    Ptr : Root_Connection_Ptr;
  end record;
  type Data_Base_Ptr is access Data_Base_Object'Class;
  --
  -- Finalize deletes Ptr.
```

```
---
procedure Finalize (Object : Data_Base_Object);
---
--- Handles, for internal use, they do not have any
--- interesting methods.
---
package Connection_Handles is
  new Object.Handle (Data_Base_Object, Data_Base_Ptr);
---
--- Database_Type is a handle
---
type Database_Type is new Connection_Handles.Handle;
end APQ.Handles;
```

> *It has the following methods:*

>

> *\* Connect method takes a string, which is the connection URL. Note:*

> *Two ways of implementing:*

>

> *1. connect is hard-coded to support every possible database, and*

> *parses the URL itself.*

>

> *2. each database is registered in a global list, connect*

> *determines the most appropriate database based on prefix in URL*

> *and calls appropriate connect method with URL.*

>

> *The second method is more complicated but more flexible. In any*

> *case, changing this should not require changing the user API.*

>

> *\* All methods associated with Root\_Connection\_Type. This does \*not\**

> *include options to set options specific to the database, like user*

> *name or password. These call the appropriate routine in the*

> *Root\_Connection\_Type'Class. Status information should perhaps be*

> *cached, so it remains constant even after queries (might be more*

> *appropriate way to handle errors).*

>

> *\* New\_Query takes a \*Database\_Type\* parameter, not a*

> *Root\_Connection\_Type parameter. This function constraints a query*

> *and saves a smart pointer to the Root\_Connection\_Type contained*

> *within the Database\_Type variable.*

You could leave it as is and add a handle to a query, which will be created from a handle to data base. It is a standard pattern: user interface has only handles:

Handle to connection ----> Connection\_Object --> Connection\_Type

Handle to query ----> Query\_Object -----> Query\_Type

Handle to connection

Query\_Object has a handle to its connection as a component. This ensures that the Connection\_Object will never be finalized before it. The original Connection\_Type and Query\_Type may be left intact. They will be allocated by handle constructors and destroyed automatically.

- > *\* Debatable: Function that returns smart pointer to*
- > *Root\_Connection\_Type'Class. This might be required to access*
- > *database specific functions. Ideally it shouldn't be required.*
- >
- > *4. Modify Root\_Query\_Type class:*
- >
- > *New\_Query is the only function that takes a \*Database\_Type\**
- > *parameter.*
- >
- > *The functions that use to require Root\_Connection\_Type now use the*
- > *smart pointer that was saved instead.*
- >
- > *Why not make the procedures abstract instead of having them raise*
- > *an Is\_Abstract? This way the checks can be done at compile time.*
- >
- > *Execute already returns exception if not connected.*
- >
- > *Add methods for obtaining and clearing status information. This*
- > *should be cached so if you execute two separate queries (in two*
- > *separate variables) on the same database, you will get two status*
- > *messages saved.*
- >
- > *I hope this helps explain what I said earlier... I believe this solves*
- > *a number of concerns with the existing system in one go.*
- >
- > *The major difference is that I have created a new Database\_Type class,*
- > *which allows the Connection\_Type to be shared in a safe manner between*
- > *the user Database\_Type and the user Query\_Type classes.*
- >
- > *This would involve making incompatible changes to Root\_Query\_Type. It*
- > *might be possible to avoid this, not sure if it would be worth the*
- > *effort.*

No, if you would make a handle for Root\_Query\_Type.

--

Regards,

Dmitry A. Kazakov

<http://www.dmitry-kazakov.de>