Summary

COLLABORATORS			
	TITLE : Summary		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		December 6, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Summary

1.1 A++ Class Summary

```
        A++ Class summary  -  A view through the library
     ------------------------------------------------------
              ($Date: 1994/05/23 14:27:48 $)


  A++ APPObject class
  A++ Doubly Linked List classes
  A++ LivingObject class
  A++ SignalResponder class
  A++ Timer class
  A++ TimedMsgPort class

  A++ AttrList class
  A++ IntuiObject class
  A++ GraphicObject class
  A++ Gadget classes
  A++ Window classes
  A++ DrawArea classes

  A++ Inheritance Tree
 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  How A++ applications are build up

  Open questions and problems (maybe you find a solution?)

 ------------------------------------------------------------
  -> Back to the root menu..
```

1.2 a++inheritancetree

```
                                                                              E

                               APPObject
                                   |
         ListC<---MapArray<---IntuiObject--->ITransponder--->AttrList
```

```
                                  |
                                  |
                     RectObject   |
                          |   \   |
                          |  GraphicObject---->GBorder
                          \    /   \     \
                           \  /     \   ScreenC
                            \/       \
                            /\      WindowCV--->IntuiMessageC
                           /  \        |
                          /    \       |
                         /   DrawArea--|-->FontC
                        /        \     |
                       /          \    |
                   GadgetCV       GWindow
                  /   |    \
                 /    |     \
                /     |      \
               /      |       \
       StdGadget BoopsiGadget GT_Gadget


 A--->B : "A uses B"-relation

   A
   |   : "B is derived from A"-relation
   B
```

1.3 a++application

```
Every A++ application that uses IntuiObjects, that is every application with
an Intuition® graphical user interface, has the same pattern:

void APPmain() // NOTE: int main(argc,argv) is defined in 'APPmain.cxx'
{
   // create objects that represent the application (any objects you like)
   // (of course you can create and destroy objects anywhere in your
   //  application.)
   ...
   // enter the main event loop
   while (running)   // control the loop yourself
   {
      SignalResponder::WaitSignal();
      // each received signal is processed within WaitSignal().
      // WaitSignal() returnes after each signal.
      // Usually you will not do anything here in this loop.
      // Action takes place in event callbacks on objects.
      // "Think object-oriented!"
   }
}

The C standard main function is defined in "APPmain.cxx" which is within
the A++ link library ("aplusplus.lib").

By simply adding a SignalResponder for the CTRL-C_BREAK signal, that breaks
```

```
the loop, the program can be terminated at any point by sending a BREAK
signal to it, either with pressing CTRL-C in the standard input window (CLI)
or by use of the 'break <process-no.>' command. Look at the demo programs.
```

1.4 appobject

```
                 APPObject class
               ---------------------
The APPObject class is one root class for the A++ classes, actually it's the
root class for most A++ classes.
It provides an object status report and introduces a runtime type inquiry
mechanism. All derived classes have to support this runtime type inquiry, too.

The APPObject plays an important role during the constructor execution of
any object:

-  constructors are invoked in order from base class towards derived
   classes. So, APPObject::APPObject is called first and sets object status
   to APPOBJECT_INVALID which is no error status!
-  Each derived class' constructor now has to check for the validity of the
   object in construction.
   - if Ok() returns TRUE do the constructor work and on successfull
     initialisation set the object to a valid state with 'setID(id_number)',
     where 'id_number' is >0. If your constructor failed to allocate some
     resources etc., use 'setError(error_number)' to set a class specific
     error code.
   - if Ok() returns FALSE only initialise for safe destruction (do not
     allocate any resources) and keep off the 'setID' method so that
     the error code can be read from the class using code.


The class user who creates an object should test it's validity with 'Ok()'.
The macro 'APPOK()' checks a given object pointer being NULL before applying
'Ok()' to it.

On object deletion, the APPObject base class sets the object status to
APPOBJECT_INVALID, thus causing obj->Ok() to return FALSE.


myProcedure( )
{
   DerivedObject *obj = new DerivedObject( );

   if (obj)          // check for memory allocation failure
      if (obj->Ok())    // check validity of the created object
   // both if stmnts can be replaced by 'if (APPOK(obj)'
   {
         ....  // work on the object
   }
   else
   {
      cerr << "Initialisation error occured: " << obj->error() << endl;
      delete obj;    // free the memory allocated for the object data
   }
   }
}
```

Class implementors are recommended to check the base class validity within
their constructor:

```
class MyClass : private InheritedClass, virtual public APPObject
{
   public:
      MyClass( )
      {
         if (Ok())   // has an error already occured ?
         {
            // at this point the object has the class ID of the last class in ↩
               the
            // inheritance list.
            if (initialise( )==FALSE)    // class initialisation
            {
               #define MYCLASS_SOMETHING_FAILED (MY_CLASS+1)
               _ierror(MYCLASS_SOMETHING_FAILED);
               // set the error variable to a value and
               // print the error string "MYCLASS_SOMETHING_FAILED" to stderr
            }
            else setID(MY_CLASS);
         }
         else  // initialise only for SAFE destruction, no resource allocation
         {
         }
      }
}
```

And define a personal class ID like this

```
   #define MY_CLASS xxx
```

Do not forget to set up the necessary support for the Type_info class,
the A++ Runtime-Type-Inquiry mechanism.