102309b0-8

Nicholas Allen

COLLABORATORS

	TITLE : 102309b0-8	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Nicholas Allen	February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

102309b0-8

1.1 MUIPlusPlus

MUIPlusPlus - A C++ interface for MUI

Developer Documentation

v1.0 by Nicholas Allen

1.2 What is MUIPlusPlus?

What is MUIPlusPlus?

MUIPlusPlus is a system that allows the use of MUI through C++ classes. This document will assume you are familiar with programming the MUI system in C and that you understand the C++ language. If this is not the case then you should read the developer documentation that is a standard part of the MUI developer archive and any good book on C++.

Note: This product is freeware so please feel free to copy to anyone else who may find it useful. The author makes no guarantess, however, to the usefulness of the product. It has been tested with GCC but not with any other compilers.

Why use MUIPlusPlus?

Although MUI is object oriented it is normally programmed from a non object oriented language such as C. This makes programming MUI more difficult and time consuming as the compiler has no idea of how the objects can behave and what classes they inherit from. By using MUIPlusPlus in conjunction with a C++ compiler the compiler can find errors in your code before you start running it. For example, you could write the following in C:

```
set (myListView, MUIA_List_Entries, 4);
```

which is clearly meaningless as the number of entries in a list is determined by what has been inserted into the list. Using MUIPlusPlus this would not be possible and the compiler would tell you that you cannot set the number of entries in a list object.

Another advantage of using MUIPlusPlus is that the compiler can check the type and number of arguments you pass to methods and attributes and ensure that they are correct and if they are not then it will convert them automatically if possible. For example, you could do the following in C:

```
BOOL isSelected;
get (myCheckmark, MUIA_Selected, &isSelected);
```

```
    if (isSelected)
    {
        .
        .
        .
    }
```

This could actually cause your program to crash as the value retrieved using get is always a LONG (32 bit) value and a BOOL is only 16 bits long and thus memory will get overwritten. In MUIPlusPlus the syntax is easier and safer:

```
    if (myCheckmark.Selected())
    {
        .
        .
        .
    }
```

Another advantage to MUIPlusPlus is that the syntax is much shorter than in C. For example, to insert an item in a list in C you would have to write:

```
    DoMethod (myList, MUIM_List_InsertSingle, "Hello world!",
              MUIV_List_Insert_Bottom);
```

and in MUIPlusPlus you could write:

```
    myList.InsertSingle("Hello world!", MUIV_List_Insert_Bottom);

    or even:

    myList.InsertBottom("Hello world!");
```

(The InsertBottom method is actually an extension provided by MUIPlusPlus to shorten the syntax of common operations).

1.3 How does it work?

How does it work?

MUIPlusPlus works by defining class definintions that describe all the methods and attributes of the standard MUI classes. When you call one of the methods in a particular class it simply invokes the DoMethod BOOPSI function for that method.

All MUI classes inherit from a base class called CMUI_Object. This class contains all the common features of MUI objects (for example, converting to and from BOOPSI object pointers and supplying a simpler interface for the object to BOOPSI function calls).

1.4 Creating objects

Creating objects

Creating objects is rather easy in MUIPlusPlus and is as easy as declaring an integer variable in C++. All you do is declare a variable of the class you want with "CMUI_" at the start of the class name. For example, if you want to declare a MUI slider object you just write:

```
CMUI_Slider mySlider;    // Declares a MUI slider object called mySlider
```

This only declares an object it does not initialize it. This allows you to declare all your objects before the muimaster.library has been opened. If you want to declare and initialize an object at the same time you put the initialization tag list after the object name. For example, to declare a MUI slider object that has a minimum of 1 and maximum of 10 you would write:

```
CMUI_Slider mySlider (MUIA_Numeric_Min, 1,
                      MUIA_Numeric_Max, 10,
                      TAG_DONE);
```

If you want to declare an object but initialize it some time later then you could do this:

```
CMUI_Slider mySlider;

.
.
.

mySlider = CMUI_Slider (MUIA_Numeric_Min, 1,
                        MUIA_Numeric_Max, 10,
                        TAG_DONE);
```

1.5 Disposing objects

Disposing objects

When you have finished using an object you need to dispose of it. Normally you will only have to dispose the Application object as this will automatically dispose all of its children. However, if you add and remove objects dynamically to the application then they will need to be disposed when they are removed. To dispose of an object you just call its Dispose method. For example:

```
myApplication.Dispose();
```

will dispose of the application object and its children. The same syntax is used to dispose of any kind of object.

If you dispose of an object that is connected to an application then when the application is disposed it will get disposed twice and this will cause your program to crash. If MUIPlusPlus is in

debug

mode then an error message will
be generated and the object won't be disposed.

1.6 Getting and setting attributes

Getting and setting attributes

To get an attribute from an object you call the appropriate get member function. The get member function has the same name as the last part of the tag name in MUI. Thus to check if a window is open or not (i.e. get its MUIA_Window_Open attribute) you would write:

```
if (myWindow.Open())
{
    .
    .
    .
}
```

To set an attribute you call the appropraite set member function. The set member function has the same name as the last part of the tag name in MUI but is peceeded by "Set". The value to set the attribute is passed as a parameter. Thus to open a window object (i.e. set its MUIA_Window_Open attribute to TRUE) you would write:

```
myWindow.SetOpen(TRUE);
```

1.7 Calling methods

Calling methods

Calling methods with a fixed number of arguments

To call a method just call the member function of the object with the same name as the last part of the method tag with the parameters to the method. For example, to call the Jump method for a List object to jump to line 10 you would write:

```
myList.Jump(10);    // Scroll the 10th line into view
```

Calling methods with a variable number of arguments

Because of the way BOOPSI has been implemented calling methods that have a variable number of arguments cannot be called in exactly the same way as those with a fixed number of arguments. In MUIPlusPlus you must pass an object called sva (meaning Start Variable Args) as the first argument to the method and then any other arguments must follow. For example, the Notify method of an object takes a variable number of arguments. To setup a notification for an application object to return MUIV_Application_ReturnID_Quit when a window object is closed:

```
    window.Notify(sva, MUIA_Window_CloseRequest, TRUE,
                  app, 2, MUIM_Application_ReturnID,
                  MUIV_Application_ReturnID_Quit);
```

The only difference, therefore, is that you must pass sva as the first parameter. If you forget to do this then the compiler will object (you will not cause any problems in your program by forgetting to do this- it just won't compile).

1.8 Special features

```
                Special features
```

MUIPlusPlus has a few extra methods for some classes that simplify using them.

MUI_List class

```
                operator~[]
                             Getting an entry from a list using array syntax

                InsertBottom
                            Insert an item at the bottom of the list

                InsertTop
                               Insert an item at the top of the list

                InsertActive
                            Insert an item before the active entry in a list

                InsertSorted
                            Insert an item sorted in the list
```

1.9 MUIPP_DEBUG

MUIPP_DEBUG

MUIPlusPlus has a debug mode that helps identify problems in your program when you run it. To enable this debugging you need to define a macro called MUIPP_DEBUG before inclusion of the MUIPlusplus header file. It is recommended that you enable this whilst developing your applications and when you find they are working correctly then disable it (by not defining it).

The debug mode will check that you have supplied all the required tags when creating objects. It will also warn you when an object fails to create successfully or if you try to dispose of an object that is still connected to an application.

e.g. if you write:

```
    CMUI_Listview myListview (TAG_DONE);
```

then the following message will be printed on stderr when your program is run:

MUIPP warning: when creating CMUI_Listview objects the MUIA_Listview_List attribute should be supplied.

Note: Checking required tags only works for the Application, Window and Listview classes at present.

1.10 Header file inclusion

Header file inclusion

To include the MUIPlusplus class definitions you need to include the file <libraries/MUI.hpp>. This includes the class definitions for all standard MUI classes. If you wish to include header files for a custom class then the file is called <mui/classname_mcc.hpp> (e.g. <mui/HTMLtext_mcc.hpp>).

When including the header file you can define a number of macros before hand which will affect the behaviour of the header file. These macros are listed below:

MUIPP_NOINLINES
Define if you don't want inlined member functions

MUIPP_DEBUG
Define to turn on debugging mode

MUIPP_TEMPLATES
Define to include template classes

1.11 link with a link library

Link library

MUIPlusPlus comes with a link library (compiled for the GCC compiler). The source code for the link library is also included so you can compile a version for your compiler if you do not use GCC. Using link libraries can speed up compilation. To use a link library instead of inlining the methods define

MUIPP_NOINLINES
before inclusion of the <libraries/mui.hpp> header file.

1.12 Converting to and from BOOPSI objects

Converting to and from BOOPSI objects

Sometimes you may need to convert to and from BOOPSI objects to the C++

objects.

Converting to BOOPSI objects

You would need to convert from a C++ object to a BOOPSI object, for example, when you are passing a C++ object as a tag value in an initialization list for another object. To convert to a BOOPSI object you just need to typecast it by putting (Object *) before the object's name:

```
    CMUI_Window myWin (
                          .
                          .
                          .
                       MUIA_Window_Contents, (Object *)CMUI_Button ("_Ok"),
                          .
                          .
                          .
                       );
```

If you are actually passing it as a tag value it is better to typecast to a tag value by putting (Tag) instead of (Object *). Because of this typecasting you can actually treat the C++ objects just like BOOPSI object as well:

```
    CMUI_Window myWin (......);

    DoMethod (myWin, MUIM_Window_ToFront);  // Call the ToFront method
```

although you should never need to do this.

Converting from BOOPSI objects

Converting from BOOPSI objects is also a useful thing. For example, if you setup a notification event to call a hook function with a pointer to the application object as the calling object you may want to convert this to a C++ application object:

```
void MyHookFunction (REG(a2) Object *app) {
    CMUI_Application application = app;

    // Now use application C++ class instead of DoMethod and get and set

    .
    .
    .

}
```

1.13 Dynamically creating objects

Dynamically creating objects

Dynamically adding objects

You can dynamically add objects by calling the AddMember function. For

example, if you create an application object like this:

```
    CMUI_Application myApp (MUIA_Application_Author, "Nicholas Allen",
                            MUIA_Application_Base, "TEST",
                            .
                            .
                            .
                            );
```

and then later on you wish to add a window object to the application it could be done like this:

```
    CMUI_Window myWindow (....);

    myApp.AddMember(myWindow);      // Connect the window to the application
```

Note: This function takes a BOOPSI object pointer (so you can add objects created in the usual MUI_NewObject way as well). Because of the automatic coersion to BOOPSI objects you don't have to worry if it is a C++ object or a BOOPSI object that you pass to this function).

Dynamically removing objects

Objects can be dynamically removed by calling the RemMember function. To remove the window from the application in the above example you would write:

```
    myApp.RemMember(myWindow);      // Remove window from application
```

Note: Removing objects does not dispose of them.

1.14 InsertBottom

InsertBottom

This method can be used to insert an item at the bottom of a list. It is equivalent to calling InsertSingle with MUIV_List_Insert_Bottom. For example:

```
    MUI_List myList;

    myList.InsertBottom("Hello world!");
```

1.15 InsertTop

InsertTop

This method can be used to insert an item at the top of a list. It is equivalent to calling InsertSingle with MUIV_List_Insert_Top. For example:

```
    MUI_List myList;

    myList.InsertTop("Hello world!");
```

1.16 InsertActive

```
InsertBottom
```

This method can be used to insert an item before the active entry in a list. It is equivalent to calling InsertSingle with MUIV_List_Insert_Active. For example:

```
    MUI_List myList;

    myList.InsertActive("Hello world!");
```

1.17 InsertSorted

```
InsertSorted
```

This method can be used to insert an item in a list such that it is sorted. It is equivalent to calling InsertSingle with MUIV_List_Insert_Sorted. For example:

```
    MUI_List myList;

    myList.InsertSorted("Hello world!");
```

1.18 installation

```
Installation
```

Installation is fairly easy. To install the header files copy the supplied Include directory into a path used by your compiler for header file inclusion. Alternatively, just copy the whole archive to a directory and add the Include directory to your compiler's include path list (see your compiler documentation for more details). In GCC this can be done by the following command:

```
        setenv CPLUS_INCLUDE_PATH Work:MUIPlusPlus/Include
```

assuming this archive had been unarchived to the Work: partition.

There is also a link library supplied that can be optional linked with if you do not wish to use inline member functions. This has been compiled already for the GCC compiler but the source code is included in case you wish to compile it for another compiler. To install the link library for GCC just copy libmuipp.a into GNU:lib/.

1.19 Features

```
Features
```

```
 * Supports all attributes and methods of MUI 3.8
 * Support for NList, NListview, and HTMLtext custom classes
 * Template classes supplied for List, Listview, NList, NListview
 * Methods and attributes can be inlined for efficiency or linked with a
   link library for faster compilation.
 * Ability to convert to and from BOOPSI objects
 * Extra support for List classes including AddHead, AddTail, InsertTop,
   InsertBottom, Length.
 * Numeric classes have coersision to ints and longs
 * Includes ARexx macros for generating main header filer and header files
   for custom classes from autodocs.
 * Is completely free!
```

1.20 MUIPP_NOINLINES

```
                MUIPP_NOINLINES
```

If you define MUIPP_NOINLINES before inclusion of this file then all calls to class methods will be done directly through a link library. This can speed up compilation time quite considerably but may make your executable slightly larger and slower. If this is not defined then methods will be inlined and you will not need to link with a link library at all. Thus if you write:

```
    #define MUIPP_NOINLINES         // Don't make calls inline
    #include <libraries/MUI.hpp>


    int main (void)
    {
        .
        .
        .

        myList.InsertSingle("An inserted item", MUIV_List_Insert_Bottom);

        .
        .
        .
    }
```

Then you will need to link with a link library for the call to InsertSingle. However, if you did not define MUIPP_NOINLINES then this call would be substituted by a call to

```
        DoMethod (myList, MUIM_List_InsertSingle, "An inserted item",
                  MUIV_List_Insert_Bottom);
```

automatically by the compiler. This will make the code slightly faster and eliminates the need to

```
                link~with~a~link~library
```

. However, it slows down compilation as well. During development time I define MUIPP_NOINLINES and when my code is finished I don't define this to optimize it (the best of both worlds).

1.21 MUIPP_TEMPLATES

MUIPP_TEMPLATES

If your compiler supports templates then you may wish to use the template versions of the List, Listview, NList and NListview classes. To allow these classes to be used you must define MUIPP_TEMPLATES before inclusion of the <libraries/mui.hpp> header file. Please see the supplied TListview.cpp and TNListview.cpp examples for more details.

1.22 Passing objects to functions

Passing objects to functions

Because the C++ classes for MUI are just wrapper classes, making a copy of a C++ MUI object only copies the BOOPSI object pointer that the class encapsulates. This means the when an object is copied onto the stack for passing into a function it is, in effect, passed by reference instead of by value. The function will operate on the same object that the calling function is operating on. The following (silly) example makes this a bit clearer:

```
    void SetStringContentsToHello (CMUI_String string)
    {
        string.SetContents("Hello");
    }

    void main (void)
    {
        CMUI_String myString;
        CMUI_Application app
        (
            .
            .
            .
            SubWindow, CMUI_Window
            (
                .
                .
                .
                WindowContents, CMUI_VGroup
                (
                    Child, myString = CMUI_String (...),
                    TAG_DONE
                ),
                TAG_DONE
            ),
            TAG_DONE
        );

        SetStringContentsToHello (myString);

        // This will print "Hello" because the above call has changed
        // myString even thoough it was not passed by reference
```

```
        printf ("Contents = %s\n", myString.Contents());

        app.Dispose();
    }
```

1.23 Introduction to header generation

```
                Header file generation

Supplied in this archive are tree ARexx macros that can be used for generating
C++ header files for the standard MUI classes as well as for MUI custom
classes.

In order to generate a header file a
                .muidefs
                 file needs to exist for each
class. This file describes all the attributes and methods for the class and
has the same name as the class followed by the .muidefs extension. These files
can be written by hand although a much easier and quicker solution is to use
the
                Doc2Def.rexx
                 macro to convert an autodoc file to the
                .muidefs
                 equivalent.
Another file, called the
                inheritance~file
                , then needs to be written which
describes the order in which the classes will be written to the header file
and also which classes each one inherits from. This file must be called
"Inheritance" and placed in the directory with all the other .muidefs file.

To generate a header you must then CD into the directory with these files in
and run either the
                CreateHeader.rexx
                 or
                CreateMCCHeader.rexx
                 macros depending
on whether you wish to generate the mui.hpp file or a header for a MCC (MUI
custom class).
```

1.24 The Doc2Def.rexx macro

```
                Doc2Def.rexx macro

Note: This macro requires the rexx reqtools support available on aminet
(util/rexx/RexxReqTools.lha).

This macro can be used for converting a MUI autodoc to a
                .muidefs
                 file. Copy
it to your Rexx: directory and to run it type:
```

```
    rx Doc2Def

Upon launching this macro will display a reqtools file requester allowing you
to select the autodocs that need to be converted. The autodoc must have one of
the following naming conventions:

    MUI_<classname>.doc

  or:

    MCC_<classname>.doc

Any other file names will be ignored. The files will then be converted to

                .muidefs
                 files and named <classname>.muidefs and put in the directory the
macro was originally launched from.
```

1.25 The CreateHeader.rexx macro

```
                CreateHeader.rexx macro

This macro is used to generate the mui.hpp and mui.cpp files for the standard
MUI classes. Copy this macro into you Rexx: directory. To run the macro type:

    rx CreateHeader <outputdir>

where outputdir is optional and specifies where the files will be stored. The
macro should be run from the directory containg the
                inheritance~file
                 and

                .muidefs
                 files. When this macro wites a class definition it will look for
three other types of files for each class:

    <classname>.public      File containing any additional code for
                            the class to be put in its public section.
                            Click
                here
                 for example file.

    <classname>.required    File containing list of tag names that
                            determine which tags must be supplied when
                            creating objects of this class. A warning
                            will be generated at run time if one of
                            these tags is not supplied and debug mode
                            is enabled.
                            Click
                here
                 for example file.

    <classname>.makeobj     File containing the MUIO_<name> and
                            parameters descriptions for calls to
                            MUI_MakeObject. This will allow the class
```

to be constructed using MUI_MakeObject
parameters as well as tags.
Click
here
for example file.

and generate the class accordingly. Please see the Source/MainHeader directory of this archive to see how I have done this.

1.26 The CreateMCCHeader.rexx macro

CreateMCCHeader.rexx macro

This macro can be used to create a header file for a MUI custom class. Copy it to your Rexx: directory. To run the macro type:

```
rx CreateMCCHeader classname
```

The classname should be supplied although this is only used to determine the name of the header file (which is classname_mcc.hpp). Other than this simple difference this macro is used in exactly the same way as the
CreateHeader.rexx
macro and therefore needs an
inheritance~file
and
.muidefs
files. Please see
the example NList, NListview, or HTMLtext directories to see how this has been done for these custom classes.

1.27 CMUI_Object

CMUI_Object

This is the class all MUI objects inherit from in MUIPlusPlus. It contains support for converting to and from BOOPSI object pointers. It also suppliers a rather useful and more convienient interface to BOOPSI function calls, allowing the easy setting and getting of attributes and calling methods. For example, if we have a CMUI_Window object and we wish to get its MUIA_Window_Open attribute we could write:

```
BOOL isOpen = (BOOL)myWindow.GetAttr(MUIA_Window_Open);
```

Note: This is only an example, in reality it would be neater to use the following syntax:

```
BOOL isOpen = myWindow.Open();
```

The CMUI_Object class can be used for calling methods as well:

```
myWindow.DoMethod(MUIM_Window_ToFront);
```

1.28 operator []

operator []

You can treat a CMUI_List, CMUI_Listview, CMUI_NList, CMUI_NListview objects just like arrays in C. For example:

```
APTR entry = myList[10];    // Get the 11th entry form a list
```

Remember that the index is 0 based like in C arrays. If debug~mode is turned on the a warning will be generated if the index is out of range.

1.29 inheritance file

Inheritance file:

This file is crucial when building the header files using the

CreateHeader.rexx and CreateMCCHeader.rexx macros. It lists all classes that need to be defined and also which classes they inherit from. Each class must be on a seperate line and be followed (on the same line) by the names of the classes it inherits from. If more than one class is specified for inheritance then the first one will be used for C++ inheritance and the other classes will be aggregatated into the class. There are a few useful keywords that can be used in the file:

include <filename>	This will cause a verbatim include of the named file into the generated header.
comment <filename>	This will cause the named file to be includes into the header file but as a comment.
end <filename>	This will cause the named file to be included at the very end of the header file.

The inheritance file must be named "Inheritance" and stored in the same directory as the .muidefs files. To view the Inheritance file for the standard MUI classes click here.

1.30 .muidefs

.muidefs files:

```
These files describe all the attributes and methods for a particular class.
The files should be named <classname>.muidefs (e.g. Window.muidefs). The file
can be written by hand but it is probably easier and more convienient to use
the
                Doc2Def.rexx
                 macro.

Click
                here
                 to view an example muidefs file.
```

1.31 here

```
This is the contents of the List.public file:

    // By overloading the [] operator you can treat lists like arrays

    APTR operator [] (LONG pos)
    {
        APTR entry;
        DoMethod (MUIM_List_GetEntry, pos, &entry);
        return entry;
    }

    // This method is a convienient alternative to the Entries attribute

    LONG Length (void) const
    {
        return (LONG)GetAttr (MUIA_List_Entries);
    }

    // This method can be used to retrieve the number of selected entries
    // in a list

    ULONG NumSelected (void)
    {
        ULONG numSelected;
        DoMethod (MUIM_List_Select, MUIV_List_Select_All,
MUIV_List_Select_Ask, &numSelected);
        return numSelected;
    }

    // These methods can be used as shortcuts for inserting objects into
lists

    void AddHead (APTR entry)
    {
        DoMethod (MUIM_List_InsertSingle, entry, MUIV_List_Insert_Top);
    }

    void AddTail (APTR entry)
    {
        DoMethod (MUIM_List_InsertSingle, entry, MUIV_List_Insert_Bottom);
    }
```

```
    void InsertTop (APTR entry)
    {
        DoMethod (MUIM_List_InsertSingle, entry, MUIV_List_Insert_Top);
    }

    void InsertBottom (APTR entry)
    {
        DoMethod (MUIM_List_InsertSingle, entry, MUIV_List_Insert_Bottom);
    }

    void InsertSorted (APTR entry)
    {
        DoMethod (MUIM_List_InsertSingle, entry, MUIV_List_Insert_Sorted);
    }

    void InsertActive (APTR entry)
    {
        DoMethod (MUIM_List_InsertSingle, entry, MUIV_List_Insert_Active);
    }
```

Note: The file should not be indented

1.32 here

This is the contents of the Application.required file:

```
    MUIA_Application_Author
    MUIA_Application_Base
    MUIA_Application_Copyright
    MUIA_Application_Description
    MUIA_Application_Title
    MUIA_Application_Version
```

Note: The file should not be indented.

1.33 here

This is the Slider.makeobj file:

```
    MUIO_Slider
    STRPTR label, LONG min, LONG max
```

Note: The file should not be indented.

1.34 here

The standard inheritance file for MUI (file should not be indented):

```
    ; Inheritance file for use with CreateHeader.rexx
    ; Creates main MUI header and library source files
```

```
; Author: Nicholas Allen

comment HeaderComment
include HeaderStart
end HeaderEnd
abstract Notify Object
abstract Family Notify
Menustrip Family
Menu Family
Menuitem Family
Application Notify
Window Notify
Aboutmui Window
abstract Area Notify
Rectangle Area
Balance Area
Image Area
Bitmap Area
Bodychunk Bitmap
Text Area
abstract Gadget Area
String Gadget
Boopsi Gadget
Prop Gadget
Gauge Area
Scale Area
Colorfield Area
List Area
Floattext List
Volumelist List
Scrmodelist List
Dirlist List
abstract Numeric Area
Knob Numeric
Levelmeter Numeric
Numericbutton Numeric
Slider Numeric
abstract Framedisplay Area
abstract Popframe Framedisplay
abstract Imagedisplay Area
abstract Popimage Imagedisplay
Pendisplay Area
Poppen Pendisplay
Group Area
abstract Mccprefs Group
Register Group
abstract Penadjust Register
abstract Settingsgroup Group
abstract Settings Group
abstract Frameadjust Group
abstract Imageadjust Group
Virtgroup Group
Scrollgroup Group
Scrollbar Group
Listview List Group
Radio Group
Cycle Group
```

```
    Coloradjust Group
    Palette Group
    Popstring Group
    Popobject Popstring
    Poplist Popobject
    Popscreen Popobject
    Popasl Popstring
    Semaphore Object
    Applist Semaphore
    Dataspace Semaphore
    abstract Configdata Dataspace

    ; These are not standard MUI classes but have makeobj files for
    ; creating them.

    Label Text
    Button Text
    Checkmark Image
    HSpace Rectangle
    VSpace Rectangle
    HBar Rectangle
    VBar Rectangle
    BarTitle Rectangle
```

1.35 Example muidefs file

This is the .muidiefs file for the MUI application class (should not be intented):

```
    attribute MUIA_Application_Active Active N ISG BOOL
    attribute MUIA_Application_Author Author N I.G STRPTR
    attribute MUIA_Application_Base Base N I.G STRPTR
    attribute MUIA_Application_Broker Broker N ..G CxObj *
    attribute MUIA_Application_BrokerHook BrokerHook N ISG struct Hook *
    attribute MUIA_Application_BrokerPort BrokerPort N ..G struct MsgPort *
    attribute MUIA_Application_BrokerPri BrokerPri N I.G LONG
    attribute MUIA_Application_Commands Commands N ISG struct MUI_Command *
    attribute MUIA_Application_Copyright Copyright N I.G STRPTR
    attribute MUIA_Application_Description Description N I.G STRPTR
    attribute MUIA_Application_DiskObject DiskObject N ISG struct DiskObject
*
    attribute MUIA_Application_DoubleStart DoubleStart N ..G BOOL
    attribute MUIA_Application_DropObject DropObject N IS. Object *
    attribute MUIA_Application_ForceQuit ForceQuit N ..G BOOL
    attribute MUIA_Application_HelpFile HelpFile N ISG STRPTR
    attribute MUIA_Application_Iconified Iconified N .SG BOOL
    attribute MUIA_Application_Menu Menu O I.G struct NewMenu *
    attribute MUIA_Application_MenuAction MenuAction N ..G ULONG
    attribute MUIA_Application_MenuHelp MenuHelp N ..G ULONG
    attribute MUIA_Application_Menustrip Menustrip N I.. Object *
    attribute MUIA_Application_RexxHook RexxHook N ISG struct Hook *
    attribute MUIA_Application_RexxMsg RexxMsg N ..G struct RxMsg *
    attribute MUIA_Application_RexxString RexxString N .S. STRPTR
    attribute MUIA_Application_SingleTask SingleTask N I.. BOOL
    attribute MUIA_Application_Sleep Sleep N .S. BOOL
```

```
    attribute MUIA_Application_Title Title N I.G STRPTR
    attribute MUIA_Application_UseCommodities UseCommodities N I.. BOOL
    attribute MUIA_Application_UseRexx UseRexx N I.. BOOL
    attribute MUIA_Application_Version Version N I.G STRPTR
    attribute MUIA_Application_Window Window N I.. Object *
    attribute MUIA_Application_WindowList WindowList N ..G struct List *
    method MUIM_Application_AboutMUI AboutMUI N Object *refwindow
    method MUIM_Application_AddInputHandler AddInputHandler N struct
MUI_InputHandlerNode *ihnode
    method MUIM_Application_CheckRefresh CheckRefresh N
    method MUIM_Application_GetMenuCheck GetMenuCheck O ULONG MenuID
    method MUIM_Application_GetMenuState GetMenuState O ULONG MenuID
    method MUIM_Application_Input Input O LONGBITS *signal
    method MUIM_Application_InputBuffered InputBuffered N
    method MUIM_Application_Load Load N STRPTR name
    method MUIM_Application_NewInput NewInput N LONGBITS *signal
    method MUIM_Application_OpenConfigWindow OpenConfigWindow N ULONG flags
    method MUIM_Application_PushMethod PushMethod N Object *dest, LONG count,
/* ... */
    method MUIM_Application_RemInputHandler RemInputHandler N struct
MUI_InputHandlerNode *ihnode
    method MUIM_Application_ReturnID ReturnID N ULONG retid
    method MUIM_Application_Save Save N STRPTR name
    method MUIM_Application_SetConfigItem SetConfigItem N ULONG item, APTR
data
    method MUIM_Application_SetMenuCheck SetMenuCheck O ULONG MenuID, LONG
stat
    method MUIM_Application_SetMenuState SetMenuState O ULONG MenuID, LONG
stat
    method MUIM_Application_ShowHelp ShowHelp N Object *window, char *name,
char *node, LONG line
```

1.36 Author

MUIPlusPlus was written by Nicholas Allen. If you have any queries on this product then you can write to me (note I am going away to Australia for a year from May 3rd 97 and so will not be available during this time):

My email address is:

    nick@carlton-castel.demon.co.uk.

My postal address is:

    Nicholas Allen
    "Carlton Lodge"
    Rue Presbytere
    Castel
    Guernsey

Hope this is useful to you!

1.37 Bugs and bug reports

```
                Bugs and bug reports

Although this product has been tested with GCC there may be problems with
other compilers or problems with GCC that I have not noticed. If you find a
problem then you can either fix it yourself or send
                me
                 a report of the problem
and I will try to fix it (please note - I am unavailable from May 3rd 97).
```

1.38 Shortcuts

```
Shortcuts

Because of name clashes of some of MUI's shortcuts the standard MUI shortcuts
are not available. Most of them have been redefined in the <libraries/mui.hpp>
file though. The main difference between these shortcuts and the normal ones
is that to create an object instead of writing <classname>Object you write
<classname>Obj (eg instead of WindowObject you write WindowObj).
```