TWiMUI

COLLABORATORS			
	TITLE : TWiMUI		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 29, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

TWiMUI

1.1 TWiMUI.guide

```
TWiMUI
******

   TWiMUI is a C++ Class library for MUI.


  Preface              This is the preface
  Requirements         What you need
  Classes              Classes description
  Acknowledgements     Whom I have to thank
  Author               Who I am and how to get in touch with me
  MUI                  MUI
  Registration         Registration form
  Limitations          Limitations in unregistered version
  Disclaimer           No warranties at all
  Index                Index
```

1.2 TWiMUI.guide/Preface

```
Preface
*******

   This class library defines C++ classes for MUI (See MUI.). For any
official MUI Class one C++ class exists with its own constructors and,
of course, all attributes and methods which are defined in the
corresponding class.

   Classes hierarchy derives from MUI classes hierarchy.

   There are also further classes defined here that do not exist in
MUI. These classes integrate original MUI classes.
```

1.3 TWiMUI.guide/Requirements

Requirements
****

For using this class library you have to be experienced in original MUI programming.

Furthermore, you must own a C++ compiler which knows about templates and exceptions and, of course, you need MUI 3.6.

All data models will be useble from version 1.1 of TWiMUI.

1.4 TWiMUI.guide/Classes

Classes
****

Hereunder you can find the description for each class. This chapter articulates in four parts. First come support classes which are needed for a comfortable programming of a class library. Then come the MUI base classes. From none of these classes an instance can nor should be created because the effective MUI classes derive from the latter ones. The effective MUI classes will be then described in the third part. In the fourth part we will present also a couple of inline functions.

Support Classes	various support classes
Base MUI Classes	base classes for MUI classes
MUI Classes	these are the effective classes
Inline Functions	some inline functions definitions

1.5 TWiMUI.guide/Support Classes

Support Classes
===============

Here all support classes are described, which are, so to say, left overs from development of present class library.

TWiArrayList	Class for Array elements of any kind
TWiArrayCursor	Class to traverse an Array class by a cursor
TWiBuff	Class to manage a buffer of data
TWiFormat	Class to comfortably use RawDoFmt
TWiMemX	Exception class for wrong memory usage
TWiShare	Class to protect dynamic elements
TWiStr	Class for string management
TWiStrArray	Class for array of strings
TWiStrCursor	Class to traverse a String-Array class by a cursor

```
  TWiTag               Class for TagItem structures management
  TWiTagCursor         Class to traverse a TagItem class by a cursor
```

1.6 TWiMUI.guide/TWiArrayList

```
TWiArrayList
------------

     template <class T> class TWiArrayList
         {
         friend class TWiArrayCursor<T>;
         public:
             TWiArrayList(const ULONG);
             ULONG count();
             ULONG length() const;
             T &operator= (const T &);
             T &operator[] (ULONG);
             T &addTail();
             T &insert(const ULONG);
             void remove(const ULONG);
             void remTail();
             void clear();
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Misc.h

   This class provides arrays for any kind of elements.

TWiArrayList(ULONG)
     The constructor of this class initializes the array and gets
     initial number of elements it should contain. If memory allocation
     for the array fails the *TWiMemX* (See TWiMemX.) exception will be
     thrown. Default is 16.

count()
     This method returns the number of allocated elements.

length()
     This method returns the number of truly available elements.

operator APTR
     This method returns the array initial address.

operator=(const T &)
     This method assigns to the instance the elements of the other
     given instance. In case of an array of pointers you must be very
     careful.

operator[](ULONG)
     This method returns a reference to the single element indicated by
```

```
     the given index.

addTail()
     This method returns a reference to the element next to the last
     one. This method is used to add an element after the last
     available element. If no space is available for another element we
     will try to enlarge the array. Should this fail the *TWiMemX* (See
     TWiMemX.) will be thrown. The element to add must be assigned to
     the method.

insert(ULONG)
     This method returns a reference to the element with the given
     index. All elements lying behind the given element are right
     shifted at once. This method is used to add an element inside the
     array. If no space is available for another element we will try to
     enlarge the array. Should this fail the *TWiMemX* (See TWiMemX.) will be  ↩
        thrown. The
     element to add must be assigned to the method.

remove(ULONG)
     This method removes the element with the given index. All elements
     lying behind are left shifted at once.

remTail()
     This method removes the last element.

clear()
     This method removes all the elements.

   *Example:*
         .
         .
         .
         TWiArrayList<ULONG> arr(5);
         ULONG i;
         i = 256;
         arr.addTail() = i;
         .
         .
         .
```

1.7 TWiMUI.guide/TWiArrayCursor

```
TWiArrayCursor
--------------

     template <class T> class TWiArrayCursor
         {
         public:
             TWiArrayCursor(TWiArrayList<T>);
             void first();
             void last();
             void next();
             void prev();
```

```
            T &item();
            BOOL isDone();
        };

Derived class:
     none

Include file:
     classes/TWiMUI/Misc.h

   This class provides a cursor to traverse *TWiArrayList* (See
TWiArrayList.) instances more comfortably.

TWiArrayCursor(TWiArrayList<T>)
     The constructor of this class initializes the cursor for the array
     for which a reference is given.

first()
     This method sets the cursor to the first element.

last()
     This method sets the cursor to the last element.

next()
     This method sets the cursor to the next element.

prev()
     This method sets the cursor to the previous element.

item()
     This method returns a reference to the actual element, on which
     the cursor points.

isDone()
     This method returns FALSE when the cursor is on a valid element
     and TRUE when the cursor is out of bounds.

   *Example:*
        .
        .
        TWiArrayList<ULONG> arr(5);
        TWiArrayCursor<ULONG> arc(arr);
        ULONG i;
        for (arc.first()  ;  !arc.isDone()  ;  arc.next())
            {
            i = arc.item();
            .
            .
            }
        .
        .
```

1.8 TWiMUI.guide/TWiBuff

```
TWiBuff
-------

     class TWiBuff
         {
         public:
             TWiBuff(const ULONG);
             TWiBuff(const APTR, const ULONG);
             TWiBuff(const TWiBuff &);
             ~TWiBuff();
             TWiBuff &operator= (const TWiBuff &);
             APTR buffer() const;
             ULONG size() const;
             void doubleBuff();
             void setBuffSize(ULONG);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Misc.h

   This class provides an easy to manage data buffer.

TWiBuff(const ULONG)
     This constructor defines a buffer of a given size. Should this
     allocation fail, then the *TWiMemX* (See TWiMemX.) exception will
     be thrown. The default is 256.

TWiBuff(const APTR, const ULONG)
     This constructor takes the given APTR and makes this the initial
     address for a buffer of the given size. If a NULL address is
     passed then we try and allocate a buffer of the given size. Should
     this fail the *TWiMemX* (See TWiMemX.) exception is thrown.

TWiBuff(const TWiBuff &)
     This constructor takes elements from the given class and adapts a
     buffer from itself to the size of the given class and copies the
     given class buffer on its own. Should this fail to allocate memory
     then *TWiMemX* (See TWiMemX.) exception is thrown.

operator=(const TWiBuff &)
     This operator takes the elements from the given class and adapts a
     buffer from itself to the size of the given class and copies the
     given class buffer on its own. Should this fail to allocate memory
     then *TWiMemX* (See TWiMemX.) exception is thrown.

buffer()
     This method returns the initial address of the buffer.

size()
     This method returns the size of the buffer.

doubleBuff()
     This method doubles the size of the buffer. Should the allocation
```

```
     fail the *TWiMemX* (See TWiMemX.) exception is thrown.

setBuffSize(ULONG)
     This method sets the buffer size to the given value. If new size
     is smaller than the older, the buffer is cut away. Should
     allocation of new buffer fail the *TWiMemX* (See TWiMemX.)
     exception is thrown.

   *Example:*
         .
         .
         .
         STRPTR aaa = "This is only a test";
         TWiBuff buf(strlen(aaa)+1);
         memcpy(buf.buffer(),aaa,buf.size());
         .
         .
         .
```

1.9 TWiMUI.guide/TWiFormat

```
TWiFormat
---------

     class TWiFormat
         {
         public:
             TWiFormat(const STRPTR);
             TWiFormat(const STRPTR, const ULONG);
             TWiFormat(const ULONG);
             TWiFormat(const TWiFormat &);
             ~TWiFormat();
             TWiFormat &operator=(const TWiFormat &);
             void setFormat(const STRPTR);
             void setBuffer(const ULONG);
             STRPTR format(const ULONG, ...);
             STRPTR format(const APTR);
             STRPTR getFormat() const;
             STRPTR getBuff() const;
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Misc.h

   This class provides methods to comfortably call the RawDoFmt()
Amiga-Exec function.

TWiFormat(const STRPTR)
     This constructor sets the printf-style format, with which we will
     then use to format the parameters given to the format() method.
     The size of the destination buffer is set to 0 and has to be then
```

```
     set with the setBuffer() method.

TWiFormat(const STRPTR, const ULONG)
     This constructor sets the printf-style format, with which we will
     then to format the parameters given to the format() method. The
     size of the destination buffer is set to the value of the second
     parameter. Should the buffer allocation fail, the *TWiMemX* (See
     TWiMemX.) exception is thrown.

TWiFormat(const ULONG)
     This constructor sets the size of the destination buffer to the
     given value. Should the buffer allocation fail, the *TWiMemX* (See
     TWiMemX.) exception is thrown. The printf-style format is not set
     with this constructor and has to be then set with the setFormat()
     method.

operator=(const TWiFormat &)
     This operator takes the format and the buffer from the given
     class, sets its own fields' size accordingly and copies in its own
     fields the elements of the given class. Should the element
     allocation fail, the *TWiMemX* (See TWiMemX.) exception is thrown.

setFormat(const STRPTR)
     This method sets the printf-style format.

setBuffer()
     This method sets the destination buffer to the given size. Should
     the allocation of the buffer fail, the *TWiMemX* (See TWiMemX.)
     exception is thrown.

format(ULONG, ...)
     This method calls the Amiga-Exec RawDoFmt() function. Format
     arguments are given together as a parameter list. This method
     returns the address of the destination buffer.

format(APTR)
     This method calls the Amiga-Exec function RawDoFmt(). Format
     arguments are given together as an array address. This method
     return the address of the destination buffer.

getFormat()
     This method returns the address of the printf-style formats.

getBuffer()
     This method returns the address of the destination buffer.

   *Example:*
          .
          .
          .
          TWiFormat form("The number %ld is formatted",50);
          cout << form.format(28) << endl;
          .
          .
          .
```

1.10 TWiMUI.guide/TWiMemX

```
TWiMemX
-------

     class TWiMemX
         {
         public:
             TWiMemX(ULONG, ULONG);
             ULONG size() const;
             ULONG flags() const;
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Misc.h

   This class is meant to throw exceptions after a memory allocation is
failed.

TWiMemX(ULONG, ULONG)
     This constructor receives as first parameter the size of the
     memory to allocate. As second parameter flags for memory
     allocation are given. This second parameter is not mandatory.
     Default is then MEMF_ANY.

size()
     This method returns the size of the allocated memory. For this, to
     return a meaningful value, you must give the value to the
     constructor.

flags()
     This method returns the flags of the allocated memory. For this,
     to return a meaningful value, you must give the value to the
     constructor.

   *Example:*
         .
         .
         void Class::Method()
             {
             .
             .
             APTR mem = AllocMem(1024,MEMF_CLEAR);
             if (!mem)
                 throw TWiMemX(1024,MEMF_CLEAR);
             .
             .
             }

         void main()
             {
             try
                 {
```

```
                Class xx();
                xx.Method();
                }
            catch(TWiMemX(memx))
                {
                cout << "Exception TWiMemX! Size : " << memx.size() << endl;
                cout << "                   Flags: " << memx.flags() << endl;
                }
            }
```

1.11 TWiMUI.guide/TWiShare

```
TWiShare
--------

     class TWiShare
         {
         public:
             TWiShare();
             TWiShare(const TWiShare &);
             ~TWiShare();
             TWiShare &operator=(const TWiShare &);
             BOOL only() const;
         };

Derived class:
     none

Include file:
     classes/TWiMUI/Misc.h This class is used for a more simple
managing of dynamic objects. When a dynamic object appears in a class
(e.g. an element allocated by "new") it must be correctly destroyed in
the destructor. But if the dynamic object has been given to another
class by the copy constructor or by the operator= pointer, the
destruction must take place in the last instance destructor. This can
made safe by using this class as base class and testing the only()
method in the destructor.

TWiShare()
     This constructor initializes this class. Should initialization
     fail, the *TWiMemX* (See TWiMemX.) is thrown.

TWiShare(const TWiShare &)
     This constructor has to take care that a copy of this instance is
     made.

only()
     This method returns TRUE when this instance is the last one using
     this object. Otherwise, FALSE.

   *Example:*
         .
         .
         class Class : protected TWiShare
```

```
        {
        private:
            STRPTR ptr;
        public:
            Class(const STRPTR);
            Class(const Class &);
            ~Class();
            Class &operator=(const Class &);
        };

    Class::Class(const STRPTR p)
        :   TWiShare(),
            ptr(new UBYTE(strlen(p)+1))
        {
        if (ptr != NULL)
            memcpy(ptr,p,strlen(p)+1);
        else
            throw TWiMemX(strlen(p)+1);
        }

    Class::Class(const Class &c)
        :   TWiShare(c),
            ptr(c.ptr)
        {
        }

    Class::~Class()
        {
        if (only())
            delete []ptr;
        else
            ;
        }

    Class &Class::operator=(const Class &c)
        {
        if (this != &c)
            {
            ptr = c.ptr;
            TWiShare::operator=(c);
            }
        else
            ;
        return(*this);
        }
```

1.12 TWiMUI.guide/TWiStr

```
TWiStr
------

     class TWiStr
         {
         public:
```

```
            TWiStr(const STRPTR);
            TWiStr(const ULONG, const STRPTR);
            TWiStr(const TWiStr &);
            TWiStr(const UBYTE);
            virtual ~TWiStr();
            operator STRPTR() const;
            TWiStr &operator= (const TWiStr &);
            TWiStr &operator= (const STRPTR);
            TWiStr &operator+= (const TWiStr &);
            TWiStr &operator+= (const STRPTR);
            UBYTE &operator[] (ULONG);
            ULONG length() const;
            ULONG buffsize() const;
            TWiStr left(const ULONG) const;
            TWiStr right(const ULONG) const;
            TWiStr mid(const ULONG, const ULONG) const;
            void doubleBuff();
            void shrinkBuff();
            void setBuffSize(const ULONG);
        };

Derived classes:
     none

Include file:
     classes/TWiMUI/Misc.h

   This class provides a comfortable string handling.

TWiStr(const STRPTR)
     This constructor allocates a memory area of the length of the
     specified string and copies the given string onto it. Should the
     memoty allocation fail, the *TWiMemX* (See TWiMemX.) exception is
     thrown. Default is an empty string with a length of 64 bytes.

TWiStr(const ULONG, const STRPTR)
     This constructor allocates a memory area of the length specified
     as first parameter and copies onto it the content of the second
     parameter. If the length of the given string is greater then the
     size specified by the first parameter, the remaing part of the
     string is cut away. Should the memory allocation fail, the
     *TWiMemX* (See TWiMemX.) exception is thrown.

TWiStr(const TWiStr &)
     This constructor makes a copy of the given instance. Should the
     memory allocation fail, the *TWiMemX* (See TWiMemX.) exception is
     thrown.

TWiStr(const UBYTE)
     This constructor makes a null-terminated string from the given
     number. Should the memoru allocation fail, the *TWiMemX* (See
     TWiMemX.) exception is thrown.

operator STRPTR()
     This operator returns the initial address of the string.

operator=(const TWiStr &)
```

This operator takes the string in the given class, creates its own fields accordingly, and copies onto it the string in the given class. Should the allocation fail, the *TWiMemX* (See TWiMemX.) exception is thrown.

operator=(const STRPTR)
This operator takes the given string, creates its own fields accordingly and copies onto it the given string. Should the allocation fail, the *TWiMemX* (See TWiMemX.) exception is thrown.

operator+=(const TWiStr &)
This operator takes the string from the given class and concatenates it to its own string. In the case its field is too small, it will be enlarged. Should enlargement fail the *TWiMemX* (See TWiMemX.) exception is thrown.

operator+=(const STRPTR)
This operator takes the given string and concatenates it to its own. In the case its field is too small, it will be enlarged. Should enlargement fail, the *TWiMemX* (See TWiMemX.) exception is thrown.

length()
This method returns the length of the string.

buffsize()
This method returns the size of the buffer.

left(ULONG)
This method returns an instance of TWiStr class which will contain the leftmost part of the string. The number of characters to return is given as the only parameter. Should the buffer allocation fail, the *TWiMemX* (See TWiMemX.) exception is thrown.

right(ULONG)
This method returns an instance of TWiStr class which will contain the rightmost part of the string. The number of characters to return is given as the only parameter. Should the buffer allocation fail, the *TWiMemX* (See TWiMemX.) exception is thrown.

mid(ULONG, ULONG)
This method returns an instance of TWiStr class which will contain the central part of the string. The first parameter specifies the length of the string to return and the second one specifies the start position. Should the buffer allocation fail, the *TWiMemX* (See TWiMemX.) exception is thrown.

doubleBuff()
This method doubles the buffer. Should this fail, the *TWiMemX* (See TWiMemX.) exception is thrown.

shrinkBuff()
This method sets the buffer size according to the string plus the null byte. Should this fail, the *TWiMemX* (See TWiMemX.) exception is thrown.

setBuffSize(ULONG)

```
     This method sets the buffer size on the desired value. In the case
     that the new size is smaller than the older, the string is cut
     away. Should this fail, the *TWiMemX* (See TWiMemX.) exception is
     thrown.

   *Example:*
         .
         .
         TWiStr str1("This is a test");
         TWiStr str2 = str1.mid(3,4);
         .
         .
```

1.13 TWiMUI.guide/TWiStrArray

```
TWiStrArray
-----------

     class TWiStrArray
         {
         friend class TWiStrCursor;
         public:
             TWiStrArray(STRPTR, ...);
             TWiStrArray(STRPTR *);
             ULONG length() const;
             STRPTR *strings();
             STRPTR &operator[] (const ULONG);
             void addTail(const STRPTR);
             void insert(const STRPTR, const ULONG);
             void remTail();
             void remove(const ULONG);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Misc.h

   This class provides an array of string pointers.

TWiStrArray(STRPTR, ...)
     This constructor initializes the array with the given pointers.
     The list must be null-terminated. Should this fail, the *TWiMemX*
     (See TWiMemX.) exception is thrown.

TWiStrArray(STRPTR *)
     This constructor initializes the array with the given pointers.
     The list must be null-terminated. Should this fail, the *TWiMemX*
     (See TWiMemX.) exception is thrown.

length()
     This method returns the number of available elements.
```

```
strings()
     This method returns the array initial address.

operator[](ULONG)
     This method returns a reference to the element with the specified
     index.

addTail(const STRPTR)
     This method adds, as the last element of the array, the given
     pointer. Should array enlargement fail, the *TWiMemX* (See
     TWiMemX.) exception is thrown.

insert(const ULONG, const STRPTR)
     This method adds the given pointer in the array at the position
     specified by the first parameter. Should array enlargement fail,
     the *TWiMemX* (See TWiMemX.) exception is thrown.

remTail()
     This method removes the last element from the array.

remove(const ULONG)
     This method removes the specified element from the array.

   *Example:*
         .
         .
         .
         TWiStrArray sa("1. String","2. String","3. String",NULL);
         sa.addTail("4. String");
         .
         .
         .
```

1.14 TWiMUI.guide/TWiStrCursor

```
TWiStrCursor
------------

   This class is used just like the *TWiArrayCursor* (See
TWiArrayCursor.) class. Please refer to that description.
```

1.15 TWiMUI.guide/TWiTag

```
TWiTag
------

     class TWiTag
         {
         public:
             TWiTag();
```

```
            TWiTag(const Tag, ...);
            TWiTag(const struct TagItem *);
            TWiTag(const TWiTag &);
            TWiTag &operator = (const TWiTag &);
            struct TagItem *tags() const;
            void append(const TWiTag &);
            void append(const Tag, ...);
            void append(const struct TagItem *);
            void set(const TWiTag &tags);
            void set(const Tag tag1Type, ...);
            void set(const struct TagItem *tags);
            struct TagItem *find(const Tag tagType) const;
            ULONG getData(const Tag tagType, const ULONG defaultData) const;
            ULONG filter(const LONG logic, const Tag tagTypes[]);
            ULONG filter(const LONG logic, const Tag tag1, ... );
        };

Derived classes:
     none

Include file:
     classes/TWiMUI/Misc.h

   This class manages a TagItem structures array.

TWiTag()
     This constructor returns an empty array.

TWiTag(const Tag, ...)
     This constructor returns an array containing a copy of the given
     TagItem structures. The last element of the list must be TAG_DONE.
     Should array allocation fail, the *TWiMemX* (See TWiMemX.)
     exception is thrown.

TWiTag(const struct TagItem *)
     This constructor returns an array of TagItem structures and copies
     into it the given TagItem list. Should array allocation fail, the
     *TWiMemX* (See TWiMemX.) exception is thrown.

TWiTag(const TWiTag &)
     This constructor provides a copy of the given instance. Should the
     array allocation fail, the *TWiMemX* (See TWiMemX.) exception is
     thrown.

operator=(const TWiTag &)
     This constructor frees its own array and makes a copy of the given
     instance. Should array allocation fail, the *TWiMemX* (See
     TWiMemX.) exception is thrown.

tags()
     This method returns the array address.

append(const struct TagItem *)
     This method appends the given TagItem list to its own array.
     Should array enlargement fail, the *TWiMemX* (See TWiMemX.)
     exception is thrown.
```

```
append(const Tag, ...)
     This method appends the given parameter list to its own array. The
     last element in the list must be TAG_DONE. Should array
     enlargement fail, the *TWiMemX* (See TWiMemX.) exception ois
     thrown.

append(const TWiTag &)
     This method appends the given instance to its own array. Should
     array enlargement fail, the *TWiMemX* (See TWiMemX.) exception is
     thrown

set(const struct TagItem *)
     This method mixes the given Tag list into its own Tag list. If in
     the given list a Tag value exists, which already exists in this
     object's list, the value is overwritten. Tags not available in
     this object's list, are appended. Should array appending fail, the
     *TWiMemX* (See TWiMemX.) exception is thrown.

set(const Tag, ...)
     This method mixes the given Tag list into its own Tag list. If in
     the given list a Tag value exists, which already exists in this
     object's list, the value is overwitten. Tags not available in this
     object's list, are appended. The last element of the list must be
     TAG_DONE. Should array appending fails, the *TWiMemX* (See
     TWiMemX.) exception is thrown.

set(const TWiTag &)
     This method mixes the given instance into its own Tag list. If in
     the given instance a Tag value exists, which already exists in
     this object's list, the value is overwitten. Tags not available in
     this object's list, are appended. The last element of the list
     must be TAG_DONE. Should array appending fail, the *TWiMemX* (See
     TWiMemX.) exception is thrown.

find(const Tag)
     This method calls the FindTagItem() Amiga Utility function with
     the given parameters and returns its result.

getData(const Tag, const ULONG)
     This method calls the GetTagData() Amiga Utility function with the
     given parameters and returns its result.

filter(const LONG, const Tag)
     This method calls the FilterTagItems() Amiga Utility function with
     the given parameters and returns its result.

   *Example:*
          .
          .
          .
          TWiTag tag(MUIA_Window_Activate, TRUE, TAG_DONE);
          tag.append(MUIA_WINDOW_Open, TRUE, TAG_DONE);
          .
          .
          .
```

1.16 TWiMUI.guide/TWiTagCursor

```
TWiTagCursor
------------

     class TWiTagCursor
         {
         public:
             TWiTagCursor(const TWiTag &);
             TWiTagCursor(const struct TagItem *);
             BOOL isDone();
             void first();
             void next();
             struct TagItem *item() const;
             Tag itemTag() const;
             ULONG itemData() const;
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Misc.h

   This class provides a cursor to comfortably traverse instances of
*TWiTag* (See TWiTag.) class or to traverse usual Tag lists.

TWiTagCursor(const TWiTag &)
     This constructor initializes a cursor for the given array
     reference.

TWiTagCursor(const struct TagItem *)
     This constructor initializes the cursor for the given Tag list
     pointer.

isDone()
     This vmethod returns FALSE when the cursor is on a valid element
     and TRUE when it is out of bounds.

first()
     This method sets the cursor on the first element.

next()
     This method sets the cursor on the next element.

item()
     This method returns the address of the current element, which the
     cursor is pointing on.

itemTag()
     This method returns the ti_Tag value of the current element, which
     the cursor is pointing on.

itemData()
     This method returns the ti_Data value of the current element,
     which the cursor is pointing on.
```

```
*Example:*
      .
      .
      .
      TWiTag tag(MUIA_Window_Activate, TRUE, TAG_DONE);
      tag.append(MUIA_WINDOW_Open, TRUE, TAG_DONE);
      TWiTagcursor tc(tag);
      for (tc.first()  ;  !tc.isDone()  ;  tc.next())
          {
          if (tc.itemTag() = MUIA_Window_Open)
          .
          .
          .
          }
      .
      .
      .
```

1.17 TWiMUI.guide/Base MUI Classes

```
Base MUI Classes
================

   Here we describe all base MUI classes. These classes are usually not
needed in your programs. They are described here just because you may
need some methods from these.


  MUIApplicationBrokerHook   Help for BrokerHook in an Application
  MUIApplicationRexxHook     Help for RexxHook in an Application
  MUIDirlistFilterHook       Help for FilterHook in a Dirlist
  MUIGroupLayoutHook         Help for LayoutHook in a Group
  MUIListCompareHook         Help for CompareHook in a List
  MUIListConstructHook       Help for ConstructHook in a List
  MUIListDestructHook        Help for DestructHook in a List
  MUIListDisplayHook         Help for DisplayHook in a List
  MUIListMultiTestHook       Help for MultiTestHook in a List
  MUIPopaslStartHook         Help for StartHook in a Popasl
  MUIPopaslStopHook          Help for StopHook in a Popasl
  MUIPopobjectObjStrHook     Help for ObjStrHook in a Popobject
  MUIPopobjectStrObjHook     Help for StrObjHook in a Popobject
  MUIPopobjectWindowHook     Help for WindowHook in a Popobject
  MUIPopstringCloseHook      Help for CloseHook in a Popstring
  MUIPopstringOpenHook       Help for OpenHook in a Popstring
  MUIStringEditHook          Help for EditHook in a String

  MUILabelHelp               Help for Label with Control-Character

  MUIErrorX                  Exception Class
```

1.18 TWiMUI.guide/MUIApplicationBrokerHook

```
MUIApplicationBrokerHook
------------------------

     class MUIApplicationBrokerHook
         {
         private:
             struct Hook brokerhook;
             static void BrokerHookEntry(register __a0 struct Hook *,
                                         register __a2 Object *,
                                         register __a1 CxMsg *);
             virtual void BrokerHookFunc(struct Hook *,
                                         Object *,
                                         CxMsg *);
         protected:
             MUIApplicationBrokerHook();
             MUIApplicationBrokerHook(const MUIApplicationBrokerHook &);
             ~MUIApplicationBrokerHook();
             MUIApplicationBrokerHook &operator=
                 (const MUIApplicationBrokerHook &);
         public:
             struct Hook *broker();
         };

Derived classes:
     MUIApplication (See MUIApplication.)

Include file:
     classes/TWiMUI/Application.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a BrokerHook in an
Application. For this the tag *MUIA_Application_BrokerHook* should be
set to the *broker()* method return value and the *BrokerHookFunc()*
method is invoked too.

   *Example:*
     class App : public MUIApplication
         {
         private:
             virtual void BrokerHookFunc(struct Hook *,
                                         Object *,
                                         CxMsg *);
         public:
             App(MUIA_Application_Title, "???",
                 MUIA_Application_Version, "$VER: ????",
                 MUIA_Application_BrokerHook, broker(),
                 TAG_DONE);
                 { };
             ~App() { };
         };

     void App::BrokerHookFunc(struct Hook *h,
                              Object *o,
                              CxMsg *m)
```

```
        {
        your code
        };
```

1.19 TWiMUI.guide/MUIApplicationRexxHook

```
MUIApplicationRexxHook
----------------------

     class MUIApplicationRexxHook
         {
         private:
             struct Hook rexxhook;
             static ULONG RexxHookEntry(register __a0 struct Hook *,
                                        register __a2 Object *,
                                        register __a1 struct RexxMsg *);
             virtual ULONG RexxHookFunc(struct Hook *,
                                        Object *,
                                        struct RexxMsg *);
         protected:
             MUIApplicationRexxHook();
             MUIApplicationRexxHook(const MUIApplicationRexxHook &);
             ~MUIApplicationRexxHook();
             MUIApplicationRexxHook &operator= (const MUIApplicationRexxHook &);
         public:
             struct Hook *rexx();
         };

Derived classes:
     MUIApplication (See MUIApplication.)

Include file:
     classes/TWiMUI/Application.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a RexxHook in an
Application. For this the tag *MUIA_Application_RexxHook* should be set
to the *rexx()* method return value and the *RexxHookFunc()* method is
invoked too.

   *Example:*
     class App : public MUIApplication
         {
         private:
             virtual ULONG RexxHookFunc(struct Hook *,
                                        Object *,
                                        struct RexxMsg *);
         public:
             App(MUIA_Application_Title, "???",
                 MUIA_Application_Version, "$VER: ????",
                 MUIA_Application_RexxHook, rexx(),
                 TAG_DONE);
                 { };
             ~App() { };
```

```
        };

    ULONG App::RexxHookFunc(struct Hook *h,
                            Object *o,
                            struct RexxMsg *m)
        {
        your code
        };
```

1.20 TWiMUI.guide/MUIDirlistFilterHook

```
MUIDirlistFilterHook
--------------------

    class MUIDirlistFilterHook
        {
        private:
            struct Hook filterhook;
            static ULONG FilterHookEntry(register __a0 struct Hook *,
                                         register __a2 Object *,
                                         register __a1 struct ExAllData *);
            virtual ULONG FilterHookFunc(struct Hook *,
                                         Object *,
                                         struct ExAllData *);
        protected:
            MUIDirlistFilterHook();
            MUIDirlistFilterHook(const MUIDirlistFilterHook &);
            ~MUIDirlistFilterHook();
            MUIDirlistFilterHook &operator= (const MUIDirlistFilterHook &);
        public:
            struct Hook *filter();
        };

Derived classes:
    MUIDirlist (See MUIDirlist.)

Include file:
    classes/TWiMUI/Dirlist.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a FilterHook in a
Dirlist. For this the tag *MUIA_Dirlist_FilterHook* should be set to
the *filter()* method return value and the *FilterHookFunc()* method is
invoked too.

   *Example:*
     class Dir : public MUIDirlist
         {
         private:
             virtual ULONG FilterHookFunc(struct Hook *,
                                          Object *,
                                          struct ExAllData *);
         public:
             Dir(MUIA_Dirlist_Directory, "RAM:",
```

```
                MUIA_Dirlist_FilterHook, filter(),
                TAG_DONE);
                { };
            ~Dir() { };
        };

    ULONG Dir::FilterHookFunc(struct Hook *h,
                              Object *o,
                              struct ExAllData *d)
        {
        your code
        };
```

1.21 TWiMUI.guide/MUIGroupLayoutHook

```
MUIGroupLayoutHook
------------------

    class MUIGroupLayoutHook
        {
        private:
            struct Hook layouthook;
            static ULONG LayoutHookEntry(register __a0 struct Hook *,
                                         register __a2 Object *,
                                         register __a1 struct MUI_LayoutMsg *);
            virtual ULONG LayoutHookFunc(struct Hook *,
                                         Object *,
                                         struct MUI_LayoutMsg *);
        protected:
            MUIGroupLayoutHook();
            MUIGroupLayoutHook(const MUIGroupLayoutHook &);
            ~MUIGroupLayoutHook();
            MUIGroupLayoutHook &operator= (const MUIGroupLayoutHook &);
        public:
            struct Hook *layout();
        };

Derived classes:
    MUIGroup (See MUIGroup.)

Include file:
    classes/TWiMUI/Group.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a LayoutHook in a
Group. For this the tag *MUIA_Group_LayoutHook* should be set to the
*layout()* method return value and the *LayoutHookFunc()* method is
invoked too.

   *Example:*
    class Grp : public MUIGroup
        {
        private:
            virtual ULONG LayoutHookFunc(struct Hook *,
```

```
                                        Object *,
                                        struct MUI_LayoutMsg *);
        public:
            Grp(MUIA_Group_Horiz, TRUE,
                MUIA_Group_LayoutHook, layout(),
                TAG_DONE);
                { };
            ~Grp() { };
        };

    ULONG Grp::LayoutHookFunc(struct Hook *h,
                              Object *o,
                              struct MUI_LayoutMsg *m)
        {
        your code
        };
```

1.22 TWiMUI.guide/MUIListCompareHook

```
MUIListCompareHook
------------------

    class MUIListCompareHook
        {
        private:
            struct Hook comparehook;
            static LONG CompareHookEntry(register __a0 struct Hook *,
                                         register __a2 APTR,
                                         register __a1 APTR);
            virtual LONG CompareHookFunc(struct Hook *,
                                         APTR,
                                         APTR);
        protected:
            MUIListCompareHook();
            MUIListCompareHook(const MUIListCompareHook &);
            ~MUIListCompareHook();
            MUIListCompareHook &operator= (const MUIListCompareHook &);
        public:
            struct Hook *compare();
        };

Derived classes
    MUIList (See MUIList.)

Include file:
    classes/TWiMUI/List.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a CompareHook in a
List. For this the tag *MUIA_List_CompareHook* should be set to the
*compare()* method return value and the *CompareHookFunc()* method is
invoked too.

   *Example:*
```

```
     class Lst : public MUIList
         {
         private:
             virtual LONG CompareHookFunc(struct Hook *,
                                          APTR,
                                          APTR);
         public:
             Lst(ReadListFrame,
                 MUIA_List_CompareHook, compare(),
                 TAG_DONE);
                 { };
             ~Lst() { };
         };

     LONG Lst::CompareHookFunc(struct Hook *h,
                               APTR e1,
                               APTR e2)
         {
         your code.
         };
```

1.23 TWiMUI.guide/MUIListConstructHook

```
MUIListConstructHook
--------------------

     class MUIListConstructHook
         {
         private:
             struct Hook constructhook;
             static APTR ConstructHookEntry(register __a0 struct Hook *,
                                            register __a2 APTR,
                                            register __a1 APTR);
             virtual APTR ConstructHookFunc(struct Hook *,
                                            APTR,
                                            APTR);
         protected:
             MUIListConstructHook();
             MUIListConstructHook(const MUIListConstructHook &);
             ~MUIListConstructHook();
             MUIListConstructHook &operator= (const MUIListConstructHook &);
         public:
             struct Hook *construct();
         };

Derived classes:
     MUIList (See MUIList.)

Include file:
     classes/TWiMUI/List.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a ConstructHook in a
List. For this the tag *MUIA_List_ConstructHook* should be set to the
```

```
*construct()* method return value and the *ConstructHookFunc()* method
is invoked too.

   *Example:*
     class Lst : public MUIList
         {
         private:
             virtual APTR ConstructHookFunc(struct Hook *,
                                            APTR,
                                            APTR);
         public:
             Lst(ReadListFrame,
                 MUIA_List_ConstructHook, construct(),
                 TAG_DONE);
                 { };
             ~Lst() { };
         };

     APTR Lst::ConstructHookFunc(struct Hook *h,
                                 APTR p,
                                 APTR e)
         {
         your code
         };
```

1.24 TWiMUI.guide/MUIListDestructHook

```
MUIListDestructHook
-------------------

     class MUIListDestructHook
         {
         private:
             struct Hook destructhook;
             static void DestructHookEntry(register __a0 struct Hook *,
                                           register __a2 APTR,
                                           register __a1 APTR);
             virtual void DestructHookFunc(struct Hook *,
                                           APTR,
                                           APTR);
         protected:
             MUIListDestructHook();
             MUIListDestructHook(const MUIListDestructHook &);
             ~MUIListDestructHook();
             MUIListDetructHook &operator= (const MUIListDestructHook &);
         public:
             struct Hook *destruct();
         };

Derived classes:
     MUIList (See MUIList.)

Include file:
     classes/TWiMUI/List.h
```

```
   This class does not have any official constructor, as no instances
should be created from it. It is used to define a DestructHook in a
List. For this the tag *MUIA_List_DestructHook* should be set to the
*destruct()* method return value and the *DestructHookFunc()* method is
invoked too.

   *Example:*
     class Lst : public MUIList
         {
         private:
             virtual void DestructHookFunc(struct Hook *,
                                           APTR,
                                           APTR);
         public:
             Lst(ReadListFrame,
                 MUIA_List_DestructHook, destruct(),
                 TAG_DONE);
                 { };
             ~Lst() { };
         };

     void Lst::DestructHookFunc(struct Hook *h,
                                APTR p,
                                APTR e)
         {
         your code
         };
```

1.25 TWiMUI.guide/MUIListDisplayHook

```
MUIListDisplayHook
------------------

     class MUIListDisplayHook
         {
         private:
             struct Hook displayhook;
             static void DisplayHookEntry(register __a0 struct Hook *,
                                          register __a2 STRPTR *,
                                          register __a1 APTR);
             virtual void DisplayHookFunc(struct Hook *,
                                          STRPTR *,
                                          APTR);
         protected:
             MUIListDisplayHook();
             MUIListDisplayHook(const MUIListDisplayHook &);
             ~MUIListDisplayHook();
             MUIListDisplayHook &operator= (const MUIListDisplayHook &);
         public:
             struct Hook *display();
         };

Derived classes:
```

```
     MUIList (See MUIList.)

Include file:
     classes/TWiMUI/List.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a DisplayHook in a
List. For this the tag *MUIA_List_DisplayHook* should be set to the
*display()* method return value and the *DisplayHookFunc()* method is
invoked too.

   *Example:*
     class Lst : public MUIList
         {
         private:
             virtual void DisplayHookFunc(struct Hook *,
                                          STRPTR *,
                                          APTR);
         public:
             Lst(ReadListFrame,
                 MUIA_List_DisplayHook, display(),
                 TAG_DONE);
                 { };
             ~Lst() { };
         };

     void Lst::DisplayHookFunc(struct Hook *h,
                               STRPTR *a,
                               APTR e)
         {
         your code.
         };
```

1.26 TWiMUI.guide/MUIListMultiTestHook

```
MUIListMultiTestHook
--------------------

     class MUIListMultiTestHook
         {
         private:
             struct Hook multitesthook;
             static BOOL MultiTestHookEntry(register __a0 struct Hook *,
                                            register __a1 APTR);
             virtual BOOL MultitestHookFunc(struct Hook *,
                                            APTR);
         protected:
             MUIListMultiTestHook();
             MUIListMultiTestHook(const MUIListMultiTestHook &);
             ~MUIListMultiTestHook();
             MUIListMultiTestHook &operator= (const MUIListMultiTestHook &);
         public:
             struct Hook *multitest();
         };
```

```
Derived classes:
     MUIList (See MUIList.)

Include file:
     classes/TWiMUI/List.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a MultiTestHook in a
List. For this the tag *MUIA_List_MultiTestHook* should be set to the
*multitest()* method return value and the *MultiTestHookFunc()* method
is invoked too.

   *Example:*
     class Lst : public MUIList
         {
         private:
             virtual BOOL MultiTestHookFunc(struct Hook *,
                                            APTR);
         public:
             Lst(ReadListFrame,
                 MUIA_List_MultiTestHook, multitest(),
                 TAG_DONE);
                 { };
             ~Lst() { };
         };

     BOOL Lst::MultiTestHookFunc(struct Hook *h,
                                 APTR e)
         {
         your code.
         };
```

1.27 TWiMUI.guide/MUIPopaslStartHook

```
MUIPopaslStartHook
------------------

     class MUIPopaslStartHook
         {
         private:
             struct Hook starthook;
             static BOOL StartHookEntry(register __a0 struct Hook *,
                                        register __a2 Object *,
                                        register __a1 struct TagItem *);
             virtual BOOL StartHookFunc(struct Hook *,
                                        Object *,
                                        struct TagItem *);
         protected:
             MUIPopaslStartHook();
             MUIPopaslStartHook(const MUIPopaslStartHook &);
             ~MUIPopaslStartHook();
             MUIPopaslStartHook &operator= (const MUIPopaslStartHook &);
         public:
```

```
            struct Hook *start();
        };

Derived classes
     MUIPopasl (See MUIPopasl.)

Include file:
     classes/TWiMUI/Popasl.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a StartHook in a
Popasl. For this the tag *MUIA_Popasl_StartHook* should be set to the
*start()* method return value and the *StartHookFunc()* method is
invoked too.

   *Example:*
     class Pop : public MUIPopasl
         {
         private:
             virtual BOOL StartHookFunc(struct Hook *,
                                        Object *,
                                        struct TagItem *);
         public:
             Pop(MUIA_Popasl_Type, ASL_FileRequest,
                 MUIA_Popasl_StartHook, start(),
                 TAG_DONE);
                 { };
             ~Pop() { };
         };

     BOOL Pop::StartHookFunc(struct Hook *h,
                             Object *o,
                             struct TagItem *t);
         {
         your code.
         };
```

1.28 TWiMUI.guide/MUIPopaslStopHook

```
MUIPopaslStopHook
-----------------

     class MUIPopaslStopHook
         {
         private:
             struct Hook stophookFile;
             struct Hook stophookFont;
             struct Hook stophookScreenMode;
             static void StopHookEntryFile(register __a0 struct Hook *,
                                           register __a2 Object *,
                                           register __a1 struct FileRequester *);
             static void StopHookEntryFont(register __a0 struct Hook *,
                                           register __a2 Object *,
                                           register __a1 struct FontRequester *);
```

```
            static void StopHookEntryScreenMode(register __a0 struct Hook *,
                                register __a2 Object *,
                                register __a1 struct ScreenModeRequester *);
            virtual void StopHookFuncFile(struct Hook *,
                                    Object *,
                                    struct FileRequester *);
            virtual void StopHookFuncFont(struct Hook *,
                                    Object *,
                                    struct FontRequester *);
            virtual void StopHookFuncScreenMode(struct Hook *,
                                        Object *,
                                        struct ScreenModeRequester *);
        protected:
            MUIPopaslStopHook();
            MUIPopaslStopHook(const MUIPopaslStopHook &);
            ~MUIPopaslStopHook();
            MUIPopaslStopHook &operator= (const MUIPopaslStopHook &);
        public:
            struct Hook *stopFile();
            struct Hook *stopFont();
            struct Hook *stopScreenMode();
        };

Derived classes:
     MUIPopasl (See MUIPopasl.)

Include file:
     classes/TWiMUI/Popasl.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a StopHook in a Popasl.
For this the tag *MUIA_Popasl_StopHook* should be set to the return
value of one of the methods *stopFile()*, *stopFont()* or
*stopScreenMode()* and the *StopHookFunc()* method is invoked too.

   *Example:*
     class Pop : public MUIPopasl
         {
         private:
             virtual void StopHookFunc(struct Hook *,
                                 Object *,
                                 struct FileRequester *);
         public:
             Pop(MUIA_Popasl_Type, ASL_FileRequest,
                 MUIA_Popasl_StopHook, stopFile(),
                 TAG_DONE);
                 { };
             ~Pop() { };
         };

     void Pop::StopHookFunc(struct Hook *h,
                            Object *o,
                            struct FileRequester *f);
         {
         your code
         };
```

1.29 TWiMUI.guide/MUIPopobjectObjStrHook

```
MUIPopobjectObjStrHook
----------------------

     class MUIPopobjectObjStrHook
         {
         private:
             struct Hook objstrhook;
             static void ObjStrHookEntry(register __a0 struct Hook *,
                                         register __a2 Object *,
                                         register __a1 Object *);
             virtual void ObjStrHookFunc(struct Hook *,
                                         Object *,
                                         Object *);
         protected:
             MUIPopobjectObjStrHook();
             MUIPopobjectObjStrHook(const MUIPopobjectObjStrHook &);
             ~MUIPopobjectObjStrHook();
             MUIPopobjectObjStrHook &operator= (const MUIPopobjectObjStrHook &);
         public:
             struct Hook *objstr();
         };

Derived classes:
     MUIPopobject (See MUIPopobject.)

Include file:
     classes/TWiMUI/Popobject.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a ObjStrHook in a
Popobject. For this the tag *MUIA_Popobject_ObjStrHook* should be set
to the *objstr()* method return value and the *ObjStrHookFunc()* method
is invoked too.

   *Example:*
     class Pop : public MUIPopobject
         {
         private:
             virtual void ObjStrHookFunc(struct Hook *,
                                         Object *,
                                         Object *);
         public:
             Pop(MUIA_Popobject_Object, obj,
                 MUIA_Popobject_ObjStrHook, objstr(),
                 TAG_DONE);
                 { };
             ~Pop() { };
         };

     void Pop::ObjStrHookFunc(struct Hook *h,
                              Object *o,
```

```
                              Object *str);
        {
        your code
        };
```

1.30 TWiMUI.guide/MUIPopobjectStrObjHook

```
MUIPopobjectStrObjHook
----------------------

     class MUIPopobjectStrObjHook
         {
         private:
             struct Hook strobjhook;
             static BOOL StrObjHookEntry(register __a0 struct Hook *,
                                         register __a2 Object *,
                                         register __a1 Object *);
             virtual BOOL StrObjHookFunc(struct Hook *,
                                         Object *,
                                         Object *);
         protected:
             MUIPopobjectStrObjHook();
             MUIPopobjectStrObjHook(const MUIPopobjectStrObjHook &);
             ~MUIPopobjectStrObjHook();
             MUIPopobjectStrObjHook &operator= (const MUIPopobjectStrObjHook &);
         public:
             struct Hook *strobj();
         };

Derived classes:
     MUIPopobject (See MUIPopobject.)

Include file:
     classes/TWiMUI/Popobject.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a StrObjHook in a
Popobject. For this the tag *MUIA_Popobject_StrObjHook* should be set
to the *strobj()* method return value and the *StrObjHookFunc()* method
is invoked too.

   *Example:*
     class Pop : public MUIPopobject
         {
         private:
             virtual BOOL StrObjHookFunc(struct Hook *,
                                         Object *,
                                         Object *);
         public:
             Pop(MUIA_Popobject_Object, obj,
                 MUIA_Popobject_StrObjHook, strobj(),
                 TAG_DONE);
                 { };
             ~Pop() { };
```

```
        };

    BOOL Pop::StrObjHookFunc(struct Hook *h,
                             Object *o,
                             Object *str);
        {
        your code
        };
```

1.31 TWiMUI.guide/MUIPopobjectWindowHook

```
MUIPopobjectWindowHook
----------------------

    class MUIPopobjectWindowHook
        {
        private:
            struct Hook windowhook;
            static void WindowHookEntry(register __a0 struct Hook *,
                                        register __a2 Object *,
                                        register __a1 Object *);
            virtual void WindowHookFunc(struct Hook *,
                                        Object *,
                                        Object *);
        protected:
            MUIPopobjectWindowHook();
            MUIPopobjectWindowHook(const MUIPopobjectWindowHook &);
            ~MUIPopobjectWindowHook();
            MUIPopobjectWindowHook &operator= (const MUIPopobjectWindowHook &);
        public:
            struct Hook *window();
        };

Derived classes:
    MUIPopobject (See MUIPopobject.)

Include file:
    classes/TWiMUI/Popobject.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a WindowHook in a
Popobject. For this the tag *MUIA_Popobject_WindowHook* should be set
to the *window()* method return value and the *WindowHookFunc()* method
is invoked too.

   *Example:*
     class Pop : public MUIPopobject
         {
         private:
             virtual void WindowHookFunc(struct Hook *,
                                         Object *,
                                         Object *);
         public:
             Pop(MUIA_Popobject_Object, obj,
```

```
                MUIA_Popobject_WindowHook, window(),
                TAG_DONE);
                { };
            ~Pop() { };
        };

    void Pop::WindowHookFunc(struct Hook *h,
                             Object *o,
                             Object *win);
        {
        your code
        };
```

1.32 TWiMUI.guide/MUIPopstringCloseHook

```
MUIPopstringCloseHook
---------------------

    class MUIPopstringCloseHook
        {
        private:
            struct MUI_Popstring_CloseHook { Object *str; LONG success; };
            struct Hook closehook;
            static void CloseHookEntry(register __a0 struct Hook *,
                           register __a2 Object *,
                           register __a1 struct MUI_Popstring_CloseHook *);
            virtual void CloseHookFunc(struct Hook *,
                                       Object *,
                                       struct MUI_Popstring_CloseHook *);
        protected:
            MUIPopstringCloseHook();
            MUIPopstringCloseHook(const MUIPopstringCloseHook &);
            ~MUIPopstringCloseHook();
            MUIPopstringCloseHook &operator= (const MUIPopstringCloseHook &);
        public:
            struct Hook *close();
        };

Derived classes:
    MUIPopstring (See MUIPopstring.)

Include file:
    classes/TWiMUI/Popstring.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a CloseHook in a
Popstring. For this the tag *MUIA_Popstring_CloseHook* should be set to
the *close()* method return value and the *CloseHookFunc()* method is
invoked too.

   *Example:*
     class Pop : public MUIPopstring
         {
         private:
```

```
            virtual void CloseHookFunc(struct Hook *,
                                       Object *,
                                       struct MUI_Popstring_CloseHook *);
        public:
            Pop(MUIA_Popstring_String, strobj,
                MUIA_Popstring_Button, butobj,
                MUIA_Popstring_CloseHook, close(),
                TAG_DONE);
                { };
            ~Pop() { };
        };

    void Pop::CloseHookFunc(struct Hook *h,
                            Object *o,
                            struct MUI_Popstring_CloseHook *c);
        {
        your code
        };
```

1.33 TWiMUI.guide/MUIPopstringOpenHook

```
MUIPopstringOpenHook
--------------------

    class MUIPopstringOpenHook
        {
        private:
            struct Hook openhook;
            static BOOL OpenHookEntry(register __a0 struct Hook *,
                                      register __a2 Object *,
                                      register __a1 Object **);
            virtual BOOL OpenHookFunc(struct Hook *,
                                      Object *,
                                      Object **);
        protected:
            MUIPopstringOpenHook();
            MUIPopstringOpenHook(const MUIPopstringOpenHook &);
            ~MUIPopstringOpenHook();
            MUIPopstringOpenHook &operator= (const MUIPopstringOpenHook &);
        public:
            struct Hook *open();
        };

Derived classes
    MUIPopstring (See MUIPopstring.)

Include file:
    classes/TWiMUI/Popstring.h

   This class does not have any official constructor, as no instances
should be created from it. It is used to define a OpenHook in a
Popstring. For this the tag *MUIA_Popstring_OpenHook* should be set to
the *open()* method return value and the *OpenHookFunc()* method is
invoked too.
```

```
   *Example:*
     class Pop : public MUIPopstring
         {
         private:
             virtual BOOL OpenHookFunc(struct Hook *,
                                       Object *,
                                       Object **);
         public:
             Pop(MUIA_Popstring_String, strobj,
                 MUIA_Popstring_Button, butobj,
                 MUIA_Popstring_OpenHook, open(),
                 TAG_DONE);
                 { };
             ~Pop() { };
         };

     BOOL Pop::OpenHookFunc(struct Hook *h,
                            Object *o,
                            Object **str);
         {
         your code
         };
```

1.34 TWiMUI.guide/MUIStringEditHook

```
MUIStringEditHook
-----------------

     class MUIStringEditHook
         {
         private:
             struct Hook edithook;
             static void EditHookEntry(register __a0 struct Hook *,
                                       register __a2 struct SGWork *,
                                       register __a1 Msg);
             virtual void EditHookFunc(struct Hook *,
                                       struct SGWork *,
                                       Msg);
         protected:
             MUIStringEditHook();
             MUIStringEditHook(const MUIStringEditHook &);
             ~MUIStringEditHook();
             MUIStringEditHook &operator= (const MUIStringEditHook &);
         public:
             struct Hook *edit();
         };

Derived classes:
     MUIString (See MUIString.)

Include file:
     classes/TWiMUI/String.h
```

```
   This class does not have any official constructor, as no instances
should be created from it. It is used to define a EditHook in a String.
For this the tag *MUIA_String_EditHook* should be set to the *edit()*
method return value and the *EditHookFunc()* method is invoked too.

   *Example:*
     class Str : public MUIString
         {
         private:
             virtual void EditHookFunc(struct Hook *,
                                       struct SGWork *,
                                       Msg);
         public:
             Str(MUIA_String_Contents, "String-Contents",
                 MUIA_String_MaxLen, 64,
                 MUIA_String_EditHook, edit(),
                 TAG_DONE);
                 { };
             ~Str() { };
         };

     void Str::EditHookFunc(struct Hook *h,
                            struct SGWork *s,
                            Msg m);
         {
         your code
         };
```

1.35 TWiMUI.guide/MUILabelHelp

```
MUILabelHelp
------------

     class MUILabelHelp
         {
         private:
             StringC labstr;
             UBYTE cc;
         protected:
             MUILabelHelp(const STRPTR);
             MUILabelHelp(const MUILabelHelp &);
             virtual ~MUILabelHelp();
             MUILabelHelp &operator=(const MUILabelHelp &);
             StringC &gLab();
             UBYTE gCC();
         };

Derived classes:
     MUILabButton        (See MUILabButton.)
     MUILabCheckmark     (See MUILabCheckmark.)
     MUILabCycle         (See MUILabCycle.)
     MUILabNumericbutton (See MUILabNumericbutton.)
     MUILabRadio         (See MUILabRadio.)
     MUILabSlider        (See MUILabSlider.)
```

```
     MUILabString        (See MUILabString.)
Include file:
     classes/TWiMUI/Notify.h

   This class does not have any official constructor. It is used to
identify an object with a label and, at the same time, to define a
control character for use with keyboard. For this you must assign to
the derived class a string for the label in which you must precede the
control character with an underscore ("_"). This underscore will be
stripped from the string and the control character will be underlined.

   *Example:*
     void main()
         {
         MUILabString Str("_String:","Contents",32);
         .
         .
         };
```

1.36 TWiMUI.guide/MUIErrorX

```
MUIErrorX
---------

     class MUIErrorX
         {
         public:
             MUIErrorX(const TWiStr &, const ULONG);
             MUIErrorX(const MUIErrorX &);
             ~MUIErrorX();
             const TWiStr &name() const;
             ULONG typ() const;
         };

Derived classes
     none

Include file:
     classes/TWiMUI/Notify.h

   This class is thrown when an error occurs during a MUI function. If
this exception is caught, name of defective class can be discovered by
the *name* method and the corresponding number with the *typ* method.
This numbers are preset as follows

     const ULONG MUIV_TWiMUI_MUIErrorX_Notify        = 1;
     const ULONG MUIV_TWiMUI_MUIErrorX_Family        = 2;
     const ULONG MUIV_TWiMUI_MUIErrorX_Menustrip     = 3;
     const ULONG MUIV_TWiMUI_MUIErrorX_Menu          = 4;
     const ULONG MUIV_TWiMUI_MUIErrorX_Menuitem      = 5;
     const ULONG MUIV_TWiMUI_MUIErrorX_Application   = 6;
     const ULONG MUIV_TWiMUI_MUIErrorX_Window        = 7;
     const ULONG MUIV_TWiMUI_MUIErrorX_Aboutmui      = 8;
     const ULONG MUIV_TWiMUI_MUIErrorX_Area          = 9;
```

```
  const ULONG MUIV_TWiMUI_MUIErrorX_Rectangle     = 10;
  const ULONG MUIV_TWiMUI_MUIErrorX_Balance       = 11;
  const ULONG MUIV_TWiMUI_MUIErrorX_Image         = 12;
  const ULONG MUIV_TWiMUI_MUIErrorX_Bitmap        = 13;
  const ULONG MUIV_TWiMUI_MUIErrorX_Bodychunk     = 14;
  const ULONG MUIV_TWiMUI_MUIErrorX_Text          = 15;
  const ULONG MUIV_TWiMUI_MUIErrorX_Gadget        = 16;
  const ULONG MUIV_TWiMUI_MUIErrorX_String        = 17;
  const ULONG MUIV_TWiMUI_MUIErrorX_Boopsi        = 18;
  const ULONG MUIV_TWiMUI_MUIErrorX_Prop          = 19;
  const ULONG MUIV_TWiMUI_MUIErrorX_Gauge         = 20;
  const ULONG MUIV_TWiMUI_MUIErrorX_Scale         = 21;
  const ULONG MUIV_TWiMUI_MUIErrorX_Colorfield    = 22;
  const ULONG MUIV_TWiMUI_MUIErrorX_List          = 23;
  const ULONG MUIV_TWiMUI_MUIErrorX_Floattext     = 24;
  const ULONG MUIV_TWiMUI_MUIErrorX_Volumelist    = 25;
  const ULONG MUIV_TWiMUI_MUIErrorX_Dirlist       = 26;
  const ULONG MUIV_TWiMUI_MUIErrorX_Numeric       = 27;
  const ULONG MUIV_TWiMUI_MUIErrorX_Knob          = 28;
  const ULONG MUIV_TWiMUI_MUIErrorX_Levelmeter    = 29;
  const ULONG MUIV_TWiMUI_MUIErrorX_Numericbutton = 30;
  const ULONG MUIV_TWiMUI_MUIErrorX_Slider        = 31;
  const ULONG MUIV_TWiMUI_MUIErrorX_Pendisplay    = 32;
  const ULONG MUIV_TWiMUI_MUIErrorX_Poppen        = 33;
  const ULONG MUIV_TWiMUI_MUIErrorX_Group         = 34;
  const ULONG MUIV_TWiMUI_MUIErrorX_Register      = 35;
  const ULONG MUIV_TWiMUI_MUIErrorX_Penadjust     = 36;
  const ULONG MUIV_TWiMUI_MUIErrorX_Virtgroup     = 37;
  const ULONG MUIV_TWiMUI_MUIErrorX_Scrollgroup   = 38;
  const ULONG MUIV_TWiMUI_MUIErrorX_Scrollbar     = 39;
  const ULONG MUIV_TWiMUI_MUIErrorX_Listview      = 40;
  const ULONG MUIV_TWiMUI_MUIErrorX_Radio         = 41;
  const ULONG MUIV_TWiMUI_MUIErrorX_Cycle         = 42;
  const ULONG MUIV_TWiMUI_MUIErrorX_Coloradjust   = 43;
  const ULONG MUIV_TWiMUI_MUIErrorX_Palette       = 44;
  const ULONG MUIV_TWiMUI_MUIErrorX_Popstring     = 45;
  const ULONG MUIV_TWiMUI_MUIErrorX_Popobject     = 46;
  const ULONG MUIV_TWiMUI_MUIErrorX_Poplist       = 47;
  const ULONG MUIV_TWiMUI_MUIErrorX_Popasl        = 48;
  const ULONG MUIV_TWiMUI_MUIErrorX_Semaphore     = 49;
  const ULONG MUIV_TWiMUI_MUIErrorX_Dataspace     = 50;
  const ULONG MUIV_TWiMUI_MUIErrorX_CreateClass   = 100;

*Example:*
  void main()
      {
      try
          {
          MUIWindow(......);
          MUIApplication(......);
          }
      catch(MUIErrorX(m))
          {
          cout << "MUI-Error:   " << m.typ() << endl;
          cout << "  Classname: " << m.name() << endl;
          }
      catch(...)
```

```
            {
            cout << "Unknown Exception!" << endl;
            }
        };
```

1.37 TWiMUI.guide/MUI Classes

```
MUI Classes
===========


  UserDispatch          How to define our own Dispatcher
  MUIAboutmui           The class Aboutmui
  MUIApplication        The class Application
  MUIArea               The class Area
  MUIBalance            The class Balance
  MUIBitmap             The class Bitmap
  MUIBodychunk          The class Bodychunk
  MUIBoopsi             The class Boopsi
  MUIButton             The class Button
  MUILabButton          The class Button with label support
  MUICheckmark          The class Checkmark
  MUILabCheckmark       The class Checkmark whit label support
  MUIColoradjust        The class Coloradjust
  MUIColorfield         The class Colorfield
  MUICycle              The class Cycle
  MUILabCycle           The class Cycle whit label support
  MUIDataspace          The class Dataspace
  MUIDirlist            The class Dirlist
  MUIFamily             The class Family
  MUIFloattext          The class Floattext
  MUIGadget             The class Gadget
  MUIGauge              The class Gauge
  MUIGroup              The class Group
  MUIGroupH             The class Group with horizontal allignment
  MUIGroupV             The class Group with vertical allignment
  MUIGroupCol           The class Group with a number columns
  MUIGroupRow           The class Group with a number of rows
  MUIImage              The class Image
  MUIKnob               The class Knob
  MUILabel              The class Label
  MUILevelmeter         The class Levelmeter
  MUIList               The class List
  MUIListview           The class Listview
  MUIMenu               The class Menu
  MUIMenuitem           The class Menuitem
  MUIMenusep            The class Menusep
  MUIMenustrip          The class Menustrip
  MUINotify             The class Notify
  MUINumeric            The class Numeric
  MUINumericbutton      The class Numericbutton
  MUILabNumericbutton   The class Numericbutton with label support
  MUIPalette            The class Palette
  MUIPendisplay         The class Pendisplay
```

```
  MUIPopasl             The class Popasl
  MUIPopbutton          The class Popbutton
  MUIPoplist            The class Poplist
  MUIPopobject          The class Popobject
  MUIPoppen             The class Poppen
  MUIPopstring          The class Popstring
  MUIProp               The class Prop
  MUIRadio              The class Radio
  MUILabRadio           The class Radio with label support
  MUIRectangle          The class Rectangle
  MUIHBar               The class Rectangle as HBar
  MUIVBar               The class Rectangle as VBar
  MUIRegister           The class Register
  MUIRequest            The class Request
  MUIScale              The class Scale
  MUIScrollbar          The class Scrollbar
  MUIScrollgroup        The class Scrollgroup
  MUISemaphore          The class Semaphore
  MUISlider             The class Slider
  MUILabSlider          The class Slider with label support
  MUIString             The class String
  MUILabString          The class String with label support
  MUIText               The class Text
  MUIVirtgroup          The class Virtgroup
  MUIVolumelist         The class Volumelist
  MUIWindow             The class Window
```

   Here we describe the C++ classes that come from single MUI classes. Furthermore we also describe further classes that are derived from these and that offer more support to the programmer.

   For any MUI-Attribute, which can be set or get, a corresponding method with the same name exists to set and get the attribute. Methods with parameters are used to set attributes and methods with return values to get them.

   Also for MUI methods corresponding methods with the same name exist.

   Should methods or attributes with standard parameters exist like, for example, *MUIV_Application Load_ENV*, or also *...ENVARC*, possibilities in *MUIM_Application_Load*, then you can find for those ready-to-use methods for which you don’t have to give any parameters. (In these cases you don’t need to use neither *LoadENV* nor *LoadENVARC*.)

1.38 TWiMUI.guide/UserDispatch

```
UserDispatch
------------
```

   Because this classes library is based on MUI custom classes, it is necessary to use own dispatchers for MUI classes. For each class, therefore, a virtual function * ULONG UserDispatch(struct IClass *, Object *, Msg)* exists that can be used. In the scope of this function

you can then define your own methods. This must not be forgotten, of course, because all unknown methods are passed to the superclass with DoSuperMethod().

1.39 TWiMUI.guide/MUIAboutmui

```
MUIAboutmui
-----------

     class MUIAboutmui : public MUIWindow
         {
         public:
             MUIAboutmui(const struct TagItem *);
             MUIAboutmui(const Tag, ...);
             MUIAboutmui();
             MUIAboutmui(MUIAboutmui &);
             virtual ~MUIAboutmui();
             MUIAboutmui &operator= (const MUIAboutmui &);
         };

Derived classes
     none

Include file:
     classes/TWiMUI/Aboutmui.h
```

This is the C++ class for the MUI Aboutmui class. Tags are given to the constructors that are needed to create an Aboutmui window. In this, tags can be given as variable parameters or as pointers in a list.

Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI About window. To this method you can give variable tag lists or pointers to a tag list.

```
   *Example:*
     void main()
         {
         MUIAboutmui About(
             MUIA_Window_RefWindow, win,
             MUIA_Aboutmui_Application, app,
             TAG_DONE);
         .
         .
         .
         };
```

1.40 TWiMUI.guide/MUIApplication

MUIApplication

```
class MUIApplication
    :   public MUINotify,
        public MUIApplicationBrokerHook,
        public MUIApplicationRexxHook
    {
    private:
        virtual ULONG Dispatch(struct IClass *, Object *, Msg);
    public:
        MUIApplication(const struct TagItem *);
        MUIApplication(const Tag, ...);
        MUIApplication();
        MUIApplication(MUIApplication &);
        virtual ~MUIApplication();
        MUIApplication &operator= (const MUIApplication &);
        void Loop();
        void Add(MUIWindow &);
        void Rem(MUIWindow &);
    };
```

Derived classes
    none

Include file:
    classes/TWiMUI/Application.h

   This is the C++ class for the MUI Application class. Tags are given to the constructors that are needed to create an Application. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Application. To this method you can give variable tag lists or pointers to a tag list.

   For this class three more methods exist:

Loop
    This method is used to send MUI in the Wait()-loop and get single signals and process them. Because this class library is based on the custom classes, single methods can be easily implemented with own dispatchers (See UserDispatch.), and using MUIA-Application_ReturnID is not needed, excepted MUI_Application_Return_ID_Quit, with which the Loop() method is broken.

Add
    This method is used to dynamically add a Window to an Application.

Rem
    This method is used to remove again a Window from an Application.

   Example:

```
  void main()
```

```
        {
        MUIApplication App(
            MUIA_Application_Title, "???",
            MUIA_Application_Version, "$VER: ????",
            TAG_DONE);
        .
        .
        .
        };
```

1.41 TWiMUI.guide/MUIArea

```
MUIArea
-------

     class MUIArea : public MUINotify
         {
         protected:
             MUIArea(const STRPTR);
         public:
             MUIArea(const struct TagItem *);
             MUIArea(const Tag, ...);
             MUIArea();
             MUIArea(MUIArea &);
             virtual ~MUIArea();
             MUIArea &operator= (const MUIArea &);
             MUIWindow *WinClass() const;
         };

Derived classes:
     MUIRectangle  (See MUIRectangle.)
     MUIBalance    (See MUIBalance.)
     MUIImage      (See MUIImage.)
     MUIBitmap     (See MUIBitmap.)
     MUIText       (See MUIText.)
     MUIGadget     (See MUIGadget.)
     MUIGauge      (See MUIGauge.)
     MUIScale      (See MUIScale.)
     MUIColorfield (See MUIColorfield.)
     MUIList       (See MUIList.)
     MUINumeric    (See MUINumeric.)
     MUIPendisplay (See MUIPendisplay.)
     MUIGroup      (See MUIGroup.)
Include file:
     classes/TWiMUI/Area.h
```

   This is the C++ class for the MUI Area class. Tags are given to the constructors that are needed to create an Area. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Area. To this method you can give variable tag lists or pointers to a tag list.

```
   In normal cases no instances are created from this class, as this is
a superclass for almost all MUI objects.

   For this class one more method exists:
WinClass
     This method returns a pointer to an instance of the MUIWindow
     class, to which this object is linked.

   *Example:*
     void main()
         {
         MUIArea Area(
             MUIA_ObjectID, 1,
                .
                .
                .
             TAG_DONE);
         .
         .
         .
         };
```

1.42 TWiMUI.guide/MUIBalance

```
MUIBalance
----------

     class MUIBalance : public MUIArea
         {
         public:
             MUIBalance(const struct TagItem *);
             MUIBalance(const Tag, ...);
             MUIBalance();
             MUIBalance(MUIBalance &);
             virtual ~MUIBalance();
             MUIBalance &operator= (const MUIBalance &);
         };

Derived classes
     none

Include file:
     classes/TWiMUI/Balance.h

   This is the C++ class for the MUI Balance class. Tags are given to
the constructors that are needed to create a Balance object. In this,
tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Balance object. To this method you can give variable tag
lists or pointers to a tag list.
```

```
   *Example:*
     void main()
         {
         MUIBalance Bal(
             MUIA_ObjectID, 1,
                 .
                 .
                 .
             TAG_DONE);
         .
         .
         .
         };
```

1.43 TWiMUI.guide/MUIBitmap

```
MUIBitmap
---------

     class MUIBitmap : public MUIArea
         {
         public:
             MUIBitmap(const struct TagItem *);
             MUIBitmap(const Tag, ...);
             MUIBitmap();
             MUIBitmap(MUIBitmap &);
             virtual ~MUIBitmap();
             MUIBitmap &operator= (const MUIBitmap &);
         };

Derived classes
     MUIBodychunk  (See MUIBodychunk.)
Include file:
     classes/TWiMUI/Bitmap.h

   This is the C++ class for the MUI Bitmap class. Tags are given to the
constructors that are needed to create a Bitmap object. In this, tags
can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Bitmap object. To this method you can give variable tag
lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIBitmap BitMap(
             MUIA_Bitmap_UseFriend, TRUE,
                 .
                 .
                 .
             TAG_DONE);
         .
```

```
        .
        .
        };
```

1.44 TWiMUI.guide/MUIBodychunk

```
MUIBodychunk
------------

     class MUIBodychunk : public MUIArea
         {
         public:
             MUIBodychunk(const struct TagItem *);
             MUIBodychunk(const Tag, ...);
             MUIBodychunk();
             MUIBodychunk(MUIBodychunk &);
             virtual ~MUIBodychunk();
             MUIBodychunk &operator= (const MUIBodychunk &);
         };

Derived classes
     none

Include file:
     classes/TWiMUI/Bodychunk.h

   This is the C++ class for the MUI Bodychunk class. Tags are given to
the constructors that are needed to create a Bodychunk object. In this,
tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Bodychunk object. To this method you can give variable
tag lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIBodychunk bc(
             MUIA_Bodychunk_Depth, 8,
                 .
                 .
                 .
             TAG_DONE);
         .
         .
         .
         };
```

1.45 TWiMUI.guide/MUIBoopsi

```
MUIBoopsi
---------

     class MUIBoopsi : public MUIGadget
         {
         public:
             MUIBoopsi(const struct TagItem *);
             MUIBoopsi(const Tag, ...);
             MUIBoopsi();
             MUIBoopsi(MUIBoopsi &);
             virtual ~MUIBoopsi();
             MUIBoopsi &operator= (const MUIBoopsi &);
         };

Derived classes:
     none

Include-File:
     classes/TWiMUI/Boopsi.h

   This is the C++ class for the MUI Boopsi class. Tags are given to the
constructors that are needed to create a Boopsi object. In this, tags
can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Boopsi object. To this method you can give variable tag
lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIBoopsi bc(
             MUIA_Boopsi_ClassID, "colorwheel.gadget",
                .
                .
                .
             TAG_DONE);
         .
         .
         .
         };
```

1.46 TWiMUI.guide/MUIButton

```
MUIButton
---------

     class MUIButton : public MUIText
         {
         public:
             MUIButton(const STRPTR);
             MUIButton(const STRPTR, const UBYTE);
```

```
            MUIButton(MUIButton &);
            virtual ~MUIButton();
            MUIButton &operator= (const MUIButton &);
        };

Derived classes
     MUILabButton  (See MUILabButton.)
Include-File:
     classes/TWiMUI/Button.h
```

   This is the C++ class to create a button from the MUI Text class. A pointer to the button content is given to the constructors. If desired, a second parameter can be given as an UBYTE. This character will be underlined in the button content and will become the control character.

```
   *Example:*
     void main()
         {
         MUIButton but("Button content",'b');
         .
         .
         .
         };
```

1.47 TWiMUI.guide/MUILabButton

```
MUILabButton
------------

     class MUILabButton
         :   public MUILabelHelp,
             public MUIButton
         {
         public:
             MUILabButton(const STRPTR);
             virtual ~MUILabButton();
             MUILabButton &operator= (const MUILabButton &);
         };

Derived classes
     none

Include file:
     classes/TWiMUI/Button.h
```

   This class is for further support in creating Buttons. A string pointer is given in the constructor. This string will become the Button content. Furthermore, this string is parsed for an underscore ("_") and the following character will be underlined in the button. Furthermore, this character becomes the control character.

```
   *Example:*
     void main()
         {
```

```
        MUILabButton but("_Save:");
        .
        .
        .
        };
```

1.48 TWiMUI.guide/MUICheckmark

```
MUICheckmark
------------

     class MUICheckmark : public MUIImage
         {
         public:
             MUICheckmark(const UBYTE);
             MUICheckmark(const STRPTR);
             MUICheckmark(MUICheckmark &);
             virtual ~MUICheckmark();
             MUICheckmark &operator= (const MUICheckmark &);
         };

Derived classes
     MUILabCheckmark  (See MUILabCheckmark.)
Include file:
     classes/TWiMUI/Checkmark.h

   This is the C++ class to create a Checkmark from the MUI Image
class. A character, or a pointer to a character, is given to the
constructor. This character becomes the control character

   *Example:*
     void main()
         {
         MUICheckmark chk('c');
         .
         .
         .
         };
```

1.49 TWiMUI.guide/MUILabCheckmark

```
MUILabCheckmark
---------------

     class MUILabCheckmark : public MUILabelHelp, public MUICheckmark
         {
         public:
             MUILabCheckmark(const STRPTR);
             virtual ~MUILabCheckmark();
             MUILabCheckmark &operator= (const MUILabCheckmark &);
```

```
        };

Derived classes:
     none

Include file:
     classes/TWiMUI/Checkmark.h
```

   This class is for further support in creating Checkmark. A string pointer is given in the constructor. This string will precede the Checkmark object as its label. Furthermore, this string is parsed for an underscore ("_") and the following character will be underlined in the button. Furthermore, this character becomes the control character.

   Example:

```
     void main()
         {
         MUILabCheckmark but("_Save:");
         .
         .
         .
         };
```

1.50 TWiMUI.guide/MUIColoradjust

```
MUIColoradjust
--------------

     class MUIColoradjust : public MUIGroup
         {
         public:
             MUIColoradjust(const struct TagItem *);
             MUIColoradjust(const Tag, ...);
             MUIColoradjust();
             MUIColoradjust(MUIColoradjust &);
             virtual ~MUIColoradjust();
             MUIColoradjust &operator= (const MUIColoradjust &);
         };

Derived classes
     none

Include file:
     classes/TWiMUI/Coloradjust.h
```

   This is the C++ class for the MUI Coloradjust class. Tags are given to the constructors that are needed to create a Coloradjust object. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Coloradjust object. To this method you can give variable tag lists or pointers to a tag list.

```
   *Example:*
     void main()
         {
         MUIColoradjust ca(
             MUIA_Coloradjust_Blue, 0,
                 .
                 .
                 .
             TAG_DONE);
         .
         .
         .
         };
```

1.51 TWiMUI.guide/MUIColorfield

```
MUIColorfield
-------------

     class MUIColorfield : public MUIGroup
         {
         public:
             MUIColorfield(const struct TagItem *);
             MUIColorfield(const Tag, ...);
             MUIColorfield();
             MUIColorfield(MUIColorfield &);
             virtual ~MUIColorfield();
             MUIColorfield &operator= (const MUIColorfield &);
         };

Derived classes
     none

Include file:
     classes/TWiMUI/Colorfield.h

   This is the C++ class for the MUI Colorfield class. Tags are given
to the constructors that are needed to create a Colorfield object. In
this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Colorfield object. To this method you can give variable
tag lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIColorfield cf(
             MUIA_Colorfield_Blue, 0,
                 .
                 .
                 .
             TAG_DONE);
```

```
        .
        .
        .
        };
```

1.52 TWiMUI.guide/MUICycle

```
MUICycle
--------

     class MUICycle : public MUIGroup
         {
         public:
             MUICycle(const struct TagItem *);
             MUICycle(const Tag, ...);
             MUICycle(const STRPTR *);
             MUICycle(const STRPTR *, const UBYTE);
             MUICycle();
             MUICycle(MUICycle &);
             virtual ~MUICycle();
             MUICycle &operator= (const MUICycle &);
         };

Derived classes:
     MUILabCycle  (See MUILabCycle.)
Include file:
     classes/TWiMUI/Cycle.h

   This is the C++ class for the MUI Cycle class. Tags are given to the
constructors that are needed to create a Cycle object. In this, tags
can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Cycle object. To this method you can give variable tag
lists or pointers to a tag list.

   In this class more possibilities are given to create instances. A
pointer to a null-terminated array or string pointers can be given.
These strings will be the content of the cycle gadget. If desired, a
second parameter can bi given as an UBYTE. This character becomes the
control character.

   *Example:*
     void main()
         {
         static const STRPTR ent[] =
             {
             "Entry 1",
             "Entry 2",
             "Entry 3",
             NULL
             };
         MUICycle cyc(ent);
```

```
        .
        .
        .
        };
```

1.53 TWiMUI.guide/MUILabCycle

```
MUILabCycle
-----------

     class MUILabCycle : public MUILabelHelp, public MUICycle
         {
         public:
             MUILabCycle(const STRPTR, const STRPTR *);
             virtual ~MUILabCycle();
             MUILabCycle &operator= (const MUILabCycle &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Cycle.h

   This class is a further support in the creation of Cycle objects. A
string pointer is given to the constructor. This string will precede
the Cycle object as its label. Furthermore, the string will be parsed
for an underscore and following character will be underlined in the
label. Furthermore, this character becomes the control character.

   *Example:*
     void main()
         {
         static const STRPTR ent[] =
             {
             "Entry 1",
             "Entry 2",
             "Entry 3",
             NULL
             };
         MUILabCycle cyc("_Cycle:",ent);
         .
         .
         .
         };
```

1.54 TWiMUI.guide/MUIDataspace

```
MUIDataspace
------------
```

```
     class MUIDataspace : public MUISemaphore
         {
         public:
             MUIDataspace(const struct TagItem *);
             MUIDataspace(const Tag, ...);
             MUIDataspace();
             MUIDataspace(MUIDataspace &);
             virtual ~MUIDataspace();
             MUIDataspace &operator= (const MUIDataspace &);
         };

Derived classes
     none

Include file:
     classes/TWiMUI/Dataspace.h This is the C++ class for the MUI
Dataspace class. Tags are given to the constructors that are needed to
create a Dataspace object. In this, tags can be given as variable
parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Dataspace object. To this method you can give variable
tag lists or pointers to a tag list.

   *Example:*
     void main()
         {
         APTR pool = CreatePool(...);
         MUIDataspace ds(MUIA_Dataspace_Pool, pool, TAG_DONE);
         .
         .
         .
         };
```

1.55 TWiMUI.guide/MUIDirlist

```
MUIDirlist
----------

     class MUIDirlist : public MUIList
         {
         public:
             MUIDirlist(const struct TagItem *);
             MUIDirlist(const Tag, ...);
             MUIDirlist();
             MUIDirlist(MUIDirlist &);
             virtual ~MUIDirlist();
             MUIDirlist &operator= (const MUIDirlist &);
         };

Derived classes:
     none
```

```
Include file:
     classes/TWiMUI/Dirlist.h

   This is the C++ class for the MUI Dirlist class. Tags are given to
the constructors that are needed to create a Dirlist object. In this,
tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Dirlist object. To this method you can give variable tag
lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIDirlist dl(
             MUIA_Dirlist_DrawersOnly, TRUE,
                .
                .
                .
             TAG_DONE);
         .
         .
         .
         };
```

1.56 TWiMUI.guide/MUIFamily

```
MUIFamily
---------

Derived classes:
     MUIMenustrip  (See MUIMenustrip.)
     MUIMenu       (See MUIMenu.)
     MUIMenuitem   (See MUIMenuitem.)
Include file:
     classes/TWiMUI/Family.h

   This is the C++ class for MUI Family class. No instance can be
created from this class as it acts as a superclass for other MUI
classes.
```

1.57 TWiMUI.guide/MUIFloattext

```
MUIFloattext
------------

     class MUIFloattext : public MUIList
         {
```

```
        public:
            MUIFloattext(const struct TagItem *);
            MUIFloattext(const Tag, ...);
            MUIFloattext();
            MUIFloattext(MUIFloattext &);
            virtual ~MUIFloattext();
            MUIFloattext &operator= (const MUIFloattext &);
        };

Derived classes:
     none

Include file:
     classes/TWiMUI/Floattext.h

   This is the C++ class for the MUI Floattext class. Tags are given to
the constructors that are needed to create a Floattext object. In this,
tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Floattext object. To this method you can give variable
tag lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIFloattext ft(
             MUIA_Floattext_Justify, TRUE,
                .
                .
                .
             TAG_DONE);
         .
         .
         .
         };
```

1.58 TWiMUI.guide/MUIGadget

```
MUIGadget
---------

     class MUIGadget : public MUIArea
         {
         public:
             MUIGadget(const struct TagItem *);
             MUIGadget(const Tag, ...);
             MUIGadget();
             MUIGadget(MUIGadget &);
             virtual ~MUIGadget();
             MUIGadget &operator= (const MUIGadget &);
         };
```

```
Derived classes:
     MUIString  (See MUIString.)
     MUIBoopsi  (See MUIBoopsi.)
     MUIProp    (See MUIProp.)
Include file:
     classes/TWiMUI/Gadget.h
```

   This is the C++ class for the MUI Gadget class. Tags are given to the constructors that are needed to create a Gadget object. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Gadget object. To this method you can give variable tag lists or pointers to a tag list.

   Example:

```
     void main()
         {
         MUIGadget gad(TAG_DONE);
         .
         .
         .
         };
```

1.59 TWiMUI.guide/MUIGauge

```
MUIGauge
--------

     class MUIGauge : public MUIArea
         {
         public:
             MUIGauge(const struct TagItem *);
             MUIGauge(const Tag, ...);
             MUIGauge();
             MUIGauge(MUIGauge &);
             virtual ~MUIGauge();
             MUIGauge &operator= (const MUIGauge &);
         };

Derived classes
     none

Include file:
     classes/TWiMUI/Gauge.h
```

   This is the C++ class for the MUI Gauge class. Tags are given to the constructors that are needed to create a Gauge object. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Gauge object. To this method you can give variable tag

```
lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIGauge dl(
             MUIA_Gauge_Horiz, TRUE,
                .
                .
                .
             TAG_DONE);
         .
         .
         .
         };
```

1.60 TWiMUI.guide/MUIGroup

```
MUIGroup
--------

     class MUIGroup : public MUIArea
         {
         public:
             MUIGroup(const struct TagItem *);
             MUIGroup(const Tag, ...);
             MUIGroup();
             MUIGroup(MUIGroup &);
             virtual ~MUIGroup();
             MUIGroup &operator= (const MUIGroup &);
         };

Derived classes:
     MUIColoradjust  (See MUIColoradjust.)
     MUICycle        (See MUICycle.)
     MUIGroupCol     (See MUIGroupCol.)
     MUIGroupH       (See MUIGroupH.)
     MUIGroupRow     (See MUIGroupRow.)
     MUIGroupV       (See MUIGroupV.)
     MUIListview     (See MUIListview.)
     MUIPalette      (See MUIPalette.)
     MUIPopstring    (See MUIPopstring.)
     MUIRadio        (See MUIRadio.)
     MUIRegister     (See MUIRegister.)
     MUIScrollbar    (See MUIScrollbar.)
     MUIScrollgroup  (See MUIScrollgroup.)
     MUIVirtgroup    (See MUIVirtgroup.)
Include file:
     classes/TWiMUI/Group.h

   This is the C++ class for the MUI Group class. Tags are given to the
constructors that are needed to create a Group object. In this, tags
can be given as variable parameters or as pointers in a list.
```

```
   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Group object. To this method you can give variable tag
lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIGroup grp(
             MUIA_Group_PageMode, TRUE,
                 .
                 .
                 .
             TAG_DONE);
         .
         .
         .
         };
```

1.61 TWiMUI.guide/MUIGroupH

```
MUIGroupH
---------

   This Group is identical to the MUIGroup (See MUIGroup.) excepted
that it is automatically an horizontal group.
```

1.62 TWiMUI.guide/MUIGroupV

```
MUIGroupV
---------

   This Group is identical to the MUIGroup (See MUIGroup.) excepted
that it is automatically a vertical group.
```

1.63 TWiMUI.guide/MUIGroupCol

```
MUIGroupCol
-----------

   This Group is identical to the MUIGroup (See MUIGroup.) excepted
that a LONG value is given as first parameter which will define the
number of columns.
```

1.64 TWiMUI.guide/MUIGroupRow

```
MUIGroupRow
-----------
```

   This Group is identical to the MUIGroup (See MUIGroup.) excepted that a LONG value is given as first parameter which will define the number of rows.

1.65 TWiMUI.guide/MUIImage

```
MUIImage
--------

     class MUIImage : public MUIArea
         {
         public:
             MUIImage(const struct TagItem *);
             MUIImage(const Tag, ...);
             MUIImage();
             MUIImage(MUIImage &);
             virtual ~MUIImage();
             MUIImage &operator= (const MUIImage &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Image.h
```

   This is the C++ class for the MUI Image class. Tags are given to the constructors that are needed to create an Image object. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Image object. To this method you can give variable tag lists or pointers to a tag list.

```
   *Example:*
     void main()
         {
         MUIImage img(
             MUIA_Image_FontMatch, TRUE,
                 .
                 .
                 .
             TAG_DONE);
         .
         .
         .
         };
```

1.66 TWiMUI.guide/MUIKnob

```
MUIKnob
-------

     class MUIKnob : public MUINumeric
         {
         public:
             MUIKnob(const struct TagItem *);
             MUIKnob(const Tag, ...);
             MUIKnob();
             MUIKnob(MUIKnob &);
             virtual ~MUIKnob();
             MUIKnob &operator= (const MUIKnob &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Knob.h

   This is the C++ class for the MUI Knob class. Tags are given to the
constructors that are needed to create a Knob object. In this, tags can
be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Knob object. To this method you can give$ variable tag
lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIKnob knob(TAG_DONE);
         .
         .
         .
         };
```

1.67 TWiMUI.guide/MUILabel

```
MUILabel
--------

     class MUILabel : public MUIText
         {
         public:
             MUILabel(const struct TagItem *);
```

```
            MUILabel(const Tag, ...);
            MUILabel();
            MUILabel(MUILabel &);
            virtual ~MUILabel();
            MUILabel &operator= (const MUILabel &);
        };

Derived classes:
     none

Include file:
     classes/TWiMUI/Label.h
```

This is a C++ class to create a label from the MUIText (See MUIText.). To any label defined in the MUI data header corresponds a MUILabel class. A string pointer, which is the label, is given to constructors. An UBYTE as a second parameter can be given to the Key-Labels. This character will be underlined in the label.

Example:

```
  void main()
      {
      MUILabel2 lab1("Label1: ");
      MUIKeyLabel2 lab2("Label2: ",'l');
      .
      .
      .
      };
```

1.68 TWiMUI.guide/MUILevelmeter

```
MUILevelmeter
-------------

     class MUILevelmeter : public MUINumeric
         {
         public:
             MUILevelmeter(const struct TagItem *);
             MUILevelmeter(const Tag, ...);
             MUILevelmeter();
             MUILevelmeter(MUILevelmeter &);
             virtual ~MUILevelmeter();
             MUILevelmeter &operator= (const MUILevelmeter &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Levelmeter.h
```

This is the C++ class for the MUI Levelmeter class. Tags are given to the constructors that are needed to create a Levelmeter object. In this, tags can be given as variable parameters or as pointers in a list.

```
   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Levelmeter object. To this method you can give§
variable tag lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUILevelmeter lm(MUIA_Levelmeter_Label, "Label", TAG_DONE);
         .
         .
         .
         };
```

1.69 TWiMUI.guide/MUIList

```
MUIList
-------

     class MUIList
         :   public MUIArea,
             public MUIListCompareHook,
             public MUIListConstructHook,
             public MUIListDestructHook,
             public MUIListDisplayHook,
             public MUIListMultiTestHook
         {
         public:
             MUIList(const struct TagItem *);
             MUIList(const Tag, ...);
             MUIList();
             MUIList(MUIList &);
             virtual ~MUIList();
             MUIList &operator= (const MUIList &);
         };

Derived classes:
     MUIDirlist      (See MUIDirlist.)
     MUIFloattext    (See MUIFloattext.)
     MUIVolumelist   (See MUIVolumelist.)
Include file:
     classes/TWiMUI/List.h

   This is the C++ class for the MUI List class. Tags are given to the
constructors that are needed to create a List object. In this, tags can
be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI List object. To this method you can give variable tag
lists or pointers to a tag list.

   *Example:*
```

```
     void main()
         {
         MUIList lst(
             MUIA_List_Active, MUIV_List_Active_Top,
             .
             .
             .
             TAG_DONE);
         .
         .
         .
         };
```

1.70 TWiMUI.guide/MUIListview

```
MUIListview
-----------

     class MUIListview : public MUIGroup
         {
         public:
             MUIListview(const struct TagItem *);
             MUIListview(const Tag, ...);
             MUIListview();
             MUIListview(MUIListview &);
             virtual ~MUIListview();
             MUIListview &operator= (const MUIListview &);
         };

Derived classes
     none

Include file:
     classes/TWiMUI/Listview.h

   This is the C++ class for the MUI Listview class. Tags are given to
the constructors that are needed to create a Listview object. In this,
tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Listview object. To this method you can give variable
tag lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIListview lv(
             MUIA_Listview_Input, TRUE,
                 .
                 .
                 .
             TAG_DONE);
         .
```

```
        .
        .
        };
```

1.71 TWiMUI.guide/MUIMenu

```
MUIMenu
-------

     class MUIMenu : public MUIFamily
         {
         public:
             MUIMenu(const struct TagItem *);
             MUIMenu(const Tag, ...);
             MUIMenu();
             MUIMenu(MUIMenu &);
             MUIMenu(const STRPTR, const Object *, ...);
             MUIMenu(const STRPTR, const MUIMenuitem *, ...);
             virtual ~MUIMenu();
             MUIMenu &operator= (const MUIMenu &);
         };

derived classes
     none

Include file:
     classes/TWiMUI/Menu.h

   This is the C++ class for the MUI Menu class. Tags are given to the
constructors that are needed to create a Menu object. In this, tags can
be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Menu object. To this method you can give variable tag
lists or pointers to a tag list.

   For this class two more constructors are implemented. To both of
them a pointer to a string as first parameter is given. This string
becomes the title of the menu. After this, either a list of Menuitem
object pointers or a list of pointers to MUIMenuitem class instances
can be given. Both lists must be null-terminated.

   *Example:*
     void main()
         {
         MUIMenu men(
             MUIA_Menu_Title, "Title",
                 .
                 .
                 .
             TAG_DONE);
         .
         .
```

```
        .
        };
```

1.72 TWiMUI.guide/MUIMenuitem

```
MUIMenuitem
-----------

     class MUIMenuitem : public MUIFamily
         {
         public:
             MUIMenuitem(const struct TagItem *);
             MUIMenuitem(const Tag, ...);
             MUIMenuitem();
             MUIMenuitem(MUIMenuitem &);
             MUIMenuitem(const STRPTR, const Object *, ...);
             MUIMenuitem(const STRPTR, const MUIMenuitem *, ...);
             virtual ~MUIMenuitem();
             MUIMenuitem &operator= (const MUIMenuitem &);
         };

Derived classes
     MUIMenusep   (See MUIMenusep.)
Include file:
     classes/TWiMUI/Menu.h

   This is the C++ class for the MUI Menuitem class. Tags are given to
the constructors that are needed to create a Menuitem object. In this,
tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Menuitem object. To this method you can give variable
tag lists or pointers to a tag list.

   For this class two more constructors are implemented. To both of
them a pointer to a string as first parameter is given. This string
becomes the title of the menuitems. After this either a list of
Menuitem object pointers or a list of pointers to MUIMenuitem class
instances can be given. Both lists must be null-terminated.

   *Example:*
     void main()
         {
         MUIMenuitem men(
             MUIA_Menuitem_Title, "Title",
                 .
                 .
                 .
             TAG_DONE);
         .
         .
         .
         };
```

1.73 TWiMUI.guide/MUIMenusep

```
MUIMenusep
----------

     class MUIMenusep : public MUIMenuitem
         {
         public:
             MUIMenusep();
             MUIMenusep(MUIMenusep &);
             virtual ~MUIMenusep();
             MUIMenusep &operator= (const MUIMenusep &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Menu.h

   This class is a support class to easily create a separation line
inside a menu.

   *Example:*
     void main()
         {
         MUIMenusep sep();
         .
         .
         .
         };
```

1.74 TWiMUI.guide/MUIMenustrip

```
MUIMenustrip
------------

     class MUIMenustrip : public MUIFamily
         {
         public:
             MUIMenustrip(const struct TagItem *);
             MUIMenustrip(const Tag, ...);
             MUIMenustrip();
             MUIMenustrip(MUIMenustrip &);
             MUIMenustrip(const STRPTR, const Object *, ...);
             MUIMenustrip(const STRPTR, const MUIMenu *, ...);
             virtual ~MUIMenustrip();
             MUIMenustrip &operator= (const MUIMenustrip &);
         };
```

```
Derived classes:
     none

Include file:
     classes/TWiMUI/Menu.h
```

   This is the C++ class for the MUI Menustrip class. Tags are given to the constructors that are needed to create a Menustrip object. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Menustrip object. To this method you can give variable tag lists or pointers to a tag list.

   For this class two more constructors are implemented. Either a list of Menu object pointers or a list of pointers to MUIMenu class instances can be given. Both lists must be null-terminated

```
   *Example:*
     void main()
         {
         MUIMenustrip men(
             MUIA_Menustrip_Enabled, TRUE,
                .
                .
                .
             TAG_DONE);
         .
         .
         .
         };
```

1.75 TWiMUI.guide/MUINotify

```
MUINotify
---------

Derived classes:
     MUIApplication  (See MUIApplication.)
     MUIArea         (See MUIArea.)
     MUIFamily       (See MUIFamily.)
     MUIWindow       (See MUIWindow.)
Include file:
     classes/TWiMUI/Notify.h
```

   This is the C++ class for the MUI Notify class. From this class no instances should be created as this class acts as a super class for all other MUI classes.

   Generally the *Create()* method is available. This method is used to consequently create a MUI object, when the corresponding constructor has been invoked without parameters. To this method tags that are

```
needed to create the corresponding MUI object are given. With this,
tags can be given as variable parameters or also as a list of pointers.

   For this class one more method exists:
AppClass
     This method returns a pointer to the MUIApplication instance to
     which this object is linked.
```

1.76 TWiMUI.guide/MUINumeric

```
MUINumeric
----------

     class MUINumeric : public MUIArea
         {
         public:
             MUINumeric(const struct TagItem *);
             MUINumeric(const Tag, ...);
             MUINumeric();
             MUINumeric(MUINumeric &);
             virtual ~MUINumeric();
             MUINumeric &operator= (const MUINumeric &);
         };

Derived classes:
     MUIKnob            (See MUIKnob.)
     MUILevelmeter      (See MUILevelmeter.)
     MUINumericbutton   (See MUINumericbutton.)
     MUISlider          (See MUISlider.)
Include file:
     classes/TWiMUI/Numeric.h

   This is the C++ class for the MUI Numeric class. Tags are given to
the constructors that are needed to create a Numeric object. In this,
tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Numeric object. To this method you can give variable tag
lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUINumeric num(
             MUIA_Numeric_Min, 0,
             MUIA_Numeric_Max, 20,
             .
             .
             .
             TAG_DONE);
         .
         .
         .
```

```
        };
```

1.77 TWiMUI.guide/MUINumericbutton

```
MUINumericbutton
----------------

     class MUINumericbutton : public MUINumeric
         {
         public:
             MUINumericbutton(const struct TagItem *);
             MUINumericbutton(const Tag, ...);
             MUINumericbutton(const STRPTR, const ULONG, const ULONG);
             MUINumericbutton(const STRPTR, const ULONG, const ULONG, const UBYTE) ↩
                 ;
             MUINumericbutton();
             MUINumericbutton(MUINumericbutton &);
             virtual ~MUINumericbutton();
             MUINumericbutton &operator= (const MUINumericbutton &);
         };

Derived classes:
     MUILabNumericbutton  (See MUILabNumericbutton.)
Include-File:
     classes/TWiMUI/Numericbutton.h

   This is the C++ class for the MUI Numericbutton class. Tags are
given to the constructors that are needed to create a Numericbutton
object. In this, tags can be given as variable parameters or as
pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Numericbutton object. To this method you can give
variable tag lists or pointers to a tag list.

   There are two more constructors that make building of Numericbutton
object easy. To both a string pointer is first given. This string will
define the prtintf-style format to which the object content should be
formatted for displaying. Second and third parameters define minimum
and maximum values for this object. If desired, than, you can provide
an UBYTE also, which will become the control character.

   *Example:*
     void main()
         {
         MUINumericbutton nb("%ld",0,20);
         .
         .
         .
         };
```

1.78 TWiMUI.guide/MUILabNumericbutton

```
MUILabNumericbutton
-------------------

     class MUILabNumericbutton
         :   public MUILabelhelp,
             public MUINumericbutton
         {
         public:
             MUILabNumericbutton(const STRPTR, const STRPTR,
                                 const ULONG, const ULONG);
             MUILabNumericbutton();
             MUILabNumericbutton(MUILabNumericbutton &);
             virtual ~MUILabNumericbutton();
             MUILabNumericbutton &operator= (const MUILabNumericbutton &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Numericbutton.h

   This class provides further support for creating a Numericbutton
object. A string pointer is given to the constructor as the first
parameter. This string will precede the object as its label.
Furthermore, this string is parsed for an underscore ("_") and the
following character will be underlined in the label. Furthermore this
character becomes the control character. The second parameter is again
a string pointer. This string will define the printf-style format to
which the object content should be formatted for displaying. Third and
fourth parameters define minimum and maximum values for this object.

   *Example:*
     void main()
         {
         MUILabNumericbutton nb("_Label: ","%ld",0,20);
         .
         .
         .
         };
```

1.79 TWiMUI.guide/MUIPalette

```
MUIPalette
----------

     class MUIPalette : public MUIGroup
         {
         public:
             MUIPalette(const struct TagItem *);
             MUIPalette(const Tag, ...);
```

```
            MUIPalette();
            MUIPalette(MUIPalette &);
            virtual ~MUIPalette();
            MUIPalette &operator= (const MUIPalette &);
        };

Derived classes:
     none

Include file:
     classes/TWiMUI/Palette.h

   This is the C++ class for the MUI Palette class. Tags are given to
the constructors that are needed to create a Palette object. In this,
tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Palette object. To this method you can give variable tag
lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIPalette lv(
             MUIA_Palette_Groupable, TRUE,
                .
                .
                .
             TAG_DONE);
         .
         .
         .
         };
```

1.80 TWiMUI.guide/MUIPendisplay

```
MUIPendisplay
-------------

     class MUIPendisplay : public MUIArea
         {
         public:
             MUIPendisplay(const struct TagItem *);
             MUIPendisplay(const Tag, ...);
             MUIPendisplay();
             MUIPendisplay(MUIPendisplay &);
             virtual ~MUIPendisplay();
             MUIPendisplay &operator= (const MUIPendisplay &);
         };

Derived classes:
     MUIPoppen  (See MUIPoppen.)
Include file:
```

```
     classes/TWiMUI/Pendisplay.h
```

   This is the C++ class for the MUI Pendisplay class. Tags are given to the constructors that are needed to create a Pendisplay object. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Pendisplay object. To this method you can give variable tag lists or pointers to a tag list.

   This class is mainly used as a superclass for the MUI Poppen class

1.81 TWiMUI.guide/MUIPopasl

```
MUIPopasl
---------

     class MUIPopasl
         :   public MUIPopstring,
             public MUIPopaslStartHook,
             public MUIPopaslStopHook
         {
         public:
             MUIPopasl(const struct TagItem *);
             MUIPopasl(const Tag, ...);
             MUIPopasl();
             MUIPopasl(MUIPopasl &);
             virtual ~MUIPopasl();
             MUIPopasl &operator= (const MUIPopasl &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Popasl.h
```

   This is the C++ class for the MUI Popasl class. Tags are given to the constructors that are needed to create a Popasl object. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Popasl object. To this method you can give variable tag lists or pointers to a tag list.

```
   *Example:*
     void main()
         {
         MUIPopasl pa(
             MUIA_Popasl_Type, ASL_FileRequest,
             .
             .
```

```
            .
            TAG_DONE);
        .
        .
        .
        };
```

1.82 TWiMUI.guide/MUIPopbutton

```
MUIPopbutton
------------

     class MUIPopbutton : public MUIImage
         {
         public:
             MUIPopbutton(const ULONG);
             MUIPopbutton(MUIPopbutton &);
             virtual ~MUIPopbutton();
             MUIPopbutton &operator= (const MUIPopbutton &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Popbutton.h

   This class supports button creation for use in the Pop-classes. The
number of a predefined button image is given to the constructor.

   *Example:*
     void main()
         {
         MUIPopbutton(MUII_PopFile);
         .
         .
         .
         };
```

1.83 TWiMUI.guide/MUIPoplist

```
MUIPoplist
----------

     class MUIPoplist : public MUIPopobject
         {
         public:
             MUIPoplist(const struct TagItem *);
             MUIPoplist(const Tag, ...);
             MUIPoplist();
```

```
            MUIPoplist(MUIPoplist &);
            virtual ~MUIPoplist();
            MUIPoplist &operator= (const MUIPoplist &);
        };

Derived classes:
     none

Include file:
     classes/TWiMUI/Poplist.h
```

   This is the C++ class for the MUI Poplist class. Tags are given to the constructors that are needed to create a Poplist object. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Poplist object. To this method you can give variable tag lists or pointers to a tag list.

1.84 TWiMUI.guide/MUIPopobject

```
MUIPopobject
------------

     class MUIPopobject
         :   public MUIPopstring,
             public MUIPopobjectObjStrHook,
             public MUIPopobjectStrObjHook,
             public MUIPopobjectStopHook
         {
         public:
             MUIPopobject(const struct TagItem *);
             MUIPopobject(const Tag, ...);
             MUIPopobject();
             MUIPopobject(MUIPopobject &);
             virtual ~MUIPopobject();
             MUIPopobject &operator= (const MUIPopobject &);
         };

Derived classes:
     MUIPoplist    (See MUIPoplist.)
Include file:
     classes/TWiMUI/Popobject.h
```

   This is the C++ class for the MUI Popobject class. Tags are given to the constructors that are needed to create a Popobject object. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Popobject object. To this method you can give variable tag lists or pointers to a tag list.

```
   *Example:*
     void main()
         {
         MUIPopobject po(
             MUIA_Popobject_Follow, TRUE,
             .
             .
             .
             TAG_DONE);
         .
         .
         .
         };
```

1.85 TWiMUI.guide/MUIPoppen

```
MUIPoppen
---------

     class MUIPoppen : public MUIPendisplay
         {
         public:
             MUIPoppen(const struct TagItem *);
             MUIPoppen(const Tag, ...);
             MUIPoppen();
             MUIPoppen(MUIPoppen &);
             virtual ~MUIPoppen();
             MUIPoppen &operator= (const MUIPoppen &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Poppen.h

   This is the C++ class for the MUI Poppen class. Tags are given to the
constructors that are needed to create a Poppen object. In this, tags
can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Poppen object. To this method you can give variable tag
lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIPoppen pp(
             MUIA_CycleChain  , 1,
             MUIA_Window_Title, "Followed Links Color",
             TAG_DONE);
         .
         .
```

```
        .
        };
```

1.86 TWiMUI.guide/MUIPopstring

```
MUIPopstring
------------

     class MUIPopstring
         :   public MUIGroup,
             public MUIPopstringCloseHook,
             public MUIPopstringOpenHook
         {
         public:
             MUIPopstring(const struct TagItem *);
             MUIPopstring(const Tag, ...);
             MUIPopstring();
             MUIPopstring(MUIPopstring &);
             virtual ~MUIPopstring();
             MUIPopstring &operator= (const MUIPopstring &);
         };

Derived classes:
     MUIPopasl     (See MUIPopasl.)
     MUIPopobject  (See MUIPopobject.)
Include file:
     classes/TWiMUI/Popstring.h

   This is the C++ class for the MUI Popstring class. Tags are given to
the constructors that are needed to create a Popstring object. In this,
tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Popstring object. To this method you can give variable
tag lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIPopbutton pb(MUII_PopUp);
         MUIPopstring ps(
             MUIA_Popstring_Button, pb.object(),
             .
             .
             .
             TAG_DONE);
         .
         .
         .
         };
```

1.87 TWiMUI.guide/MUIProp

```
MUIProp
-------

     class MUIProp : public MUIGadget
         {
         public:
             MUIProp(const struct TagItem *);
             MUIProp(const Tag, ...);
             MUIProp();
             MUIProp(MUIProp &);
             virtual ~MUIProp();
             MUIProp &operator= (const MUIProp &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Prop.h

   This is the C++ class for the MUI Prop class. Tags are given to the
constructors that are needed to create a Prop object. In this, tags can
be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Prop object. To this method you can give variable tag
lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIProp prop(
             MUIA_Prop_Horiz, TRUE,
             .
             .
             .
             TAG_DONE);
         .
         .
         .
         };
```

1.88 TWiMUI.guide/MUIRadio

```
MUIRadio
--------

     class MUIRadio : public MUIArea
         {
         public:
```

```
            MUIRadio(const struct TagItem *);
            MUIRadio(const Tag, ...);
            MUIRadio(const STRPTR);
            MUIRadio(const STRPTR, const UBYTE);
            MUIRadio();
            MUIRadio(MUIRadio &);
            virtual ~MUIRadio();
            MUIRadio &operator= (const MUIRadio &);
        };

Derived classes:
     MUILabRadio  (See MUILabRadio.)
Include file:
     classes/TWiMUI/Radio.h
```

This is the C++ class for the MUI Radio class. Tags are given to the constructors that are needed to create a Radio object. In this, tags can be given as variable parameters or as pointers in a list.

Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Radio object. To this method you can give variable tag lists or pointers to a tag list.

There are two more constructors that make building of Numericbutton object easy. To both a pointer to an array of string pointers is given. These strings define either the number of radio buttons and also their names. The array must be null-terminated. If desired, than, you can provide an UBYTE also, which will become the control character.

Example:

```
  void main()
      {
      STRPTR array[] =
          {
          "Entry 1",
          "Entry 2",
          "Entry 3",
          NULL
          };
      MUIRadio rad(array);
      .
      .
      .
      };
```

1.89 TWiMUI.guide/MUILabRadio

```
MUILabRadio
-----------

     class MUILabRadio
         :   public MUILabelhelp,
             public MUIRadio
```

```
        {
        public:
            MUILabRadio(const STRPTR, const STRPTR, const ULONG, const ULONG);
            MUILabRadio();
            MUILabRadio(MUILabRadio &);
            virtual ~MUILabRadio();
            MUILabRadio &operator= (const MUILabRadio &);
        };

Derived classes:
     none

Include file:
     classes/TWiMUI/Radio.h

   This class provides further support for creation of a Radio object.
A string pointer is given to the constructor as the first parameter.
This string will precede the object as its label. Furthermore, this
string is parsed for an underscore ("_") and the following character
will be underlined in the label. Furthermore, this character becomes
the control character. The second parameter is again a pointer to an
array of string pointers. These strings define either the number of
radio buttons and also their names. The array must be null-terminated.

   *Example:*
     void main()
         {
         STRPTR array[] =
             {
             "Entry 1",
             "Entry 2",
             "Entry 3",
             NULL
             };
         MUILabRadio nb("_Label: ",array);
         .
         .
         .
         };
```

1.90 TWiMUI.guide/MUIRectangle

```
MUIRectangle
------------

     class MUIRectangle : public MUIArea
         {
         public:
             MUIRectangle(const struct TagItem *);
             MUIRectangle(const Tag, ...);
             MUIRectangle();
             MUIRectangle(MUIRectangle &);
             virtual ~MUIRectangle();
             MUIRectangle &operator= (const MUIRectangle &);
```

```
        };

Derived classes:
     MUIHBar  (See MUIHBar.)
     MUIVBar  (See MUIVBar.)
Include file:
     classes/TWiMUI/Rectangle.h

   This is the C++ class for the MUI Rectangle class. Tags are given to
the constructors that are needed to create a Rectangle object. In this,
tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Rectangle object. To this method you can give variable
tag lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIRectangle rec(
             MUIA_Rectangle_BarTitle, "Title of the rectangle",
             MUIA_Rectangle_HBar,     TRUE,
             TAG_DONE);
         .
         .
         .
         };
```

1.91 TWiMUI.guide/MUIHBar

```
MUIHBar
-------

     class MUIHBar : public MUIRectangle
         {
         public:
             MUIHBar(const ULONG);
             MUIHBar(MUIHBar &);
             virtual ~MUIHBar();
             MUIHBar &operator= (const MUIHBar &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Rectangle.h

   This class is for further support in creating empty horizontal
spaces. The size of the space is given to the constructor.

   *Example:*
     void main()
```

```
        {
        MUIHBar hb(50);
        .
        .
        .
        };
```

1.92 TWiMUI.guide/MUIVBar

```
MUIVBar
-------

     class MUIVBar : public MUIRectangle
         {
         public:
             MUIVBar(const ULONG);
             MUIVBar(MUIVBar &);
             virtual ~MUIVBar();
             MUIVBar &operator= (const MUIVBar &);
         };

Derived classes
     none

Include file:
     classes/TWiMUI/Rectangle.h

   This class is for further support in creating empty vertical spaces.
The size of the space is given to the constructor.

   *Example:*
     void main()
         {
         MUIVBar vb(50);
         .
         .
         .
         };
   /
```

1.93 TWiMUI.guide/MUIRegister

```
MUIRegister
-----------

     class MUIRegister : public MUIGroup
         {
         public:
             MUIRegister(const struct TagItem *);
             MUIRegister(const Tag, ...);
```

```
            MUIRegister();
            MUIRegister(MUIRegister &);
            virtual ~MUIRegister();
            MUIRegister &operator= (const MUIRegister &);
        };

Derived classes:
     none

Include file:
     classes/TWiMUI/Register.h

   This is the C++ class for the MUI Register class. Tags are given to
the constructors that are needed to create a Register object. In this,
tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Register object. To this method you can give variable
tag lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIRegister reg(
             MUIA_Register_Framed, TRUE,
             .
             .
             .
             TAG_DONE);
         .
         .
         .
         };
```

1.94 TWiMUI.guide/MUIRequest

```
MUIRequest
----------

     class MUIRequest
         {
         public:
             MUIRequest(const MUIApplication &,
                 const MUIWindow &,
                 const LONGBITS,
                 const STRPTR,
                 const STRPTR,
                 const STRPTR,
                 const ULONG,
                 const ULONG *);
             MUIRequest(const MUIWindow &,
                 const LONGBITS,
                 const STRPTR,
```

```
                const STRPTR,
                const STRPTR,
                const ULONG,
                const ULONG *);
            MUIRequest(const MUIApplication &,
                const LONGBITS,
                const STRPTR,
                const STRPTR,
                const STRPTR,
                const ULONG,
                const ULONG *);
            MUIRequest(const LONGBITS,
                const STRPTR,
                const STRPTR,
                const STRPTR,
                const ULONG,
                const ULONG *);
            MUIRequest();
            MUIRequest(MUIRequest &);
            virtual ~MUIRequest();
            MUIRequest &operator= (const MUIRequest &);
        };

Derived classes:
     none

Include file:
     classes/TWiMUI/Request.h

   This is a class to easily create MUI requesters. To the constructor
you can give, at the same time, parameters like for a MUI Request call.
Until the requester is not open these parameters can be changed by the
corresponding set...-method.

   The requester will be opened by the show() method. To this method
you can give also a MUIApplication, a MUIWindow, both or also none.

   *Example:*
     void main()
         {
         MUIRequest reg(0L,"Title","_Ok","The value is: %ld", val);
         reg.show(app,win);
         .
         .
         .
         };
```

1.95 TWiMUI.guide/MUIScale

```
MUIScale
--------

     class MUIScale : public MUIArea
         {
```

```
        public:
            MUIScale(const struct TagItem *);
            MUIScale(const Tag, ...);
            MUIScale();
            MUIScale(MUIScale &);
            virtual ~MUIScale();
            MUIScale &operator= (const MUIScale &);
        };

Derived classes:
     none

Include file:
     classes/TWiMUI/Scale.h
```

   This is the C++ class for the MUI Scale class. Tags are given to the constructors that are needed to create a Scale object. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Scale object. To this method you can give variable tag lists or pointers to a tag list.

```
   *Example:*
     void main()
         {
         MUIScale sc(MUIA_Scale_Horiz, TRUE, TAG_DONE);
         .
         .
         .
         };
```

1.96 TWiMUI.guide/MUIScrollbar

```
MUIScrollbar
------------

     class MUIScrollbar : public MUIGroup
         {
         public:
             MUIScrollbar(const struct TagItem *);
             MUIScrollbar(const Tag, ...);
             MUIScrollbar();
             MUIScrollbar(MUIScrollbar &);
             virtual ~MUIScrollbar();
             MUIScrollbar &operator= (const MUIScrollbar &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Scrollbar.h
```

```
   This is the C++ class for the MUI Scrollbar class. Tags are given to
the constructors that are needed to create a Scrollbar object. In this,
tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Scrollbar object. To this method you can give variable
tag lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIScrollbar sb(
             MUIA_Scrollbar_Type, MUIV_Scrollbar_Type_Default,
             TAG_DONE);
         .
         .
         .
         };
```

1.97 TWiMUI.guide/MUIScrollgroup

```
MUIScrollgroup
--------------

     class MUIScrollgroup : public MUIGroup
         {
         public:
             MUIScrollgroup(const struct TagItem *);
             MUIScrollgroup(const Tag, ...);
             MUIScrollgroup();
             MUIScrollgroup(MUIScrollgroup &);
             virtual ~MUIScrollgroup();
             MUIScrollgroup &operator= (const MUIScrollgroup &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Scrollgroup.h

   This is the C++ class for the MUI Scrollgroup class. Tags are given
to the constructors that are needed to create a Scrollgroup object. In
this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Scrollgroup object. To this method you can give variable
tag lists or pointers to a tag list.

   *Example:*
     void main()
```

```
        {
        MUIScrollgroup sg(
            MUIA_Scrollgroup_FreeHoriz, TRUE,
            .
            .
            .
            TAG_DONE);
        .
        .
        .
        };
```

1.98 TWiMUI.guide/MUISemaphore

```
MUISemaphore
------------

     class MUISemaphore : public MUIArea
         {
         public:
             MUISemaphore(const struct TagItem *);
             MUISemaphore(const Tag, ...);
             MUISemaphore();
             MUISemaphore(MUISemaphore &);
             virtual ~MUISemaphore();
             MUISemaphore &operator= (const MUISemaphore &);
         };

Derived classes
     MUIDataspace  (See MUIDataspace.)
Include file:
     classes/TWiMUI/Semaphore.h

   This is the C++ class for the MUI Semaphore class. Tags are given to
the constructors that are needed to create a Semaphore object. In this,
tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Semaphore object. To this method you can give variable
tag lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUISemaphore se(TAG_DONE);
         .
         .
         .
         };
```

1.99 TWiMUI.guide/MUISlider

```
MUISlider
---------

     class MUISlider : public MUINumeric
         {
         public:
             MUISlider(const struct TagItem *);
             MUISlider(const Tag, ...);
             MUISlider(const ULONG, const ULONG);
             MUISlider(const ULONG, const ULONG, const UBYTE);
             MUISlider();
             MUISlider(MUISlider &);
             virtual ~MUISlider();
             MUISlider &operator= (const MUISlider &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Slider.h

   This is the C++ class for the MUI Slider class. Tags are given to the
constructors that are needed to create a Slider object. In this, tags
can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Slider object. To this method you can give variable tag
lists or pointers to a tag list.

   There are two more constructors that make creation of Slider object
easy. First and second parameters define minimum and maximum for the
object. If desired, than, you can provide an UBYTE also, which will
become the control character.

   *Example:*
     void main()
         {
         MUISlider sl(0,20);
         .
         .
         .
         };
```

1.100 TWiMUI.guide/MUILabSlider

```
MUILabSlider
------------

     class MUILabSlider
```

```
        :   public MUILabelhelp,
            public MUISlider
        {
        public:
            MUILabSlider(const STRPTR, const ULONG, const ULONG);
            MUILabSlider();
            MUILabSlider(MUILabSlider &);
            virtual ~MUILabSlider();
            MUILabSlider &operator= (const MUILabSlider &);
        };

Derived classes
     none

Include file:
     classes/TWiMUI/Slider.h

   This class provides further support for creation of a Slider object.
A string pointer is given to the constructor as the first parameter.
This string will precede the object as its label. Furthermore, this
string is parsed for an underscore ("_") and the following character
will be underlined in the label. Furthermore, this character becomes
the control character. The second and third parameters define the
minimum and maximum values for this object.

   *Example:*
     void main()
         {
         MUILabSlider nb("_Label: ","%ld",0,20);
         .
         .
         .
         };
```

1.101 TWiMUI.guide/MUIString

```
MUIString
---------

     class MUIString
         :   public MUIGadget,
             public MUIStringEditHook
         {
         public:
             MUIString(const struct TagItem *);
             MUIString(const Tag, ...);
             MUIString(const STRPTR, const ULONG);
             MUIString(const STRPTR, const ULONG, const UBYTE);
             MUIString(const ULONG, const ULONG);
             MUIString(const ULONG, const ULONG, const UBYTE);
             MUIString(const ULONG);
             MUIString(const ULONG, const UBYTE);
             MUIString();
             MUIString(MUIString &);
```

```
            virtual ~MUIString();
            MUIString &operator= (const MUIString &);
        };

Derived classes:
     MUILabString  (See MUILabString.)
Include file:
     classes/TWiMUI/String.h
```

   This is the C++ class for the MUI String class. Tags are given to the constructors that are needed to create a String object. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI String object. To this method you can give variable tag lists or pointers to a tag list.

   Furthermore, there are six more constructors implemented that make creation of String or Integer objects easy.

   The first two constructors are added here to create simple String objects. To both a string pointer is given as first parameter, that will become the object's initial content. The second parameter establishes the object's maximum length. If desired you can also pass an UBYTE. This character will become the control character.

   The other constructors are meant for creating Integer objects. If the first two parameters are both ULONG values, then the first one defines the content of the object and is turned into a string, so that the content can be shown. The second parameter defines, then, the object's maximum length. Anyway, if only one ULONG parameter is given there is no initial content of the object and the given parameter defines the maximum object's length. To all these objects an UBYTE parameter can be given also. This character, then, becomes the control character.

   Example:

```
     void main()
         {
         MUIString str1("Content 1",20);
         MUIString str2("Content 2",20,'a');
         MUIString int1(5,20);
         .
         .
         .
         };
```

1.102 TWiMUI.guide/MUILabString

```
MUILabString
------------

     class MUILabString
```

```
        :   public MUILabelhelp,
            public MUIString
        {
        public:
            MUILabString(const STRPTR, const STRPTR, const ULONG);
            MUILabString(const STRPTR, const ULONG, const ULONG);
            MUILabString();
            MUILabString(MUILabString &);
            virtual ~MUILabString();
            MUILabString &operator= (const MUILabString &);
        };

Derived classes:
     none

Include file:
     classes/TWiMUI/String.h
```

This class provides further support for creation of a String object. A string pointer is given to the constructor as the first parameter. This string will precede the object as its label. Furthermore, this string is parsed for an underscore ("_") and the following character will be underlined in the label. Furthermore, this character becomes the control character. The second parameter is then either again a string pointer or an ULONG. If this parameter is a string pointer the string becomes the content of the String object. If it is an ULONG value, this is turned into a string and this string becomes, then, the content of the object. Furthermore, the object becomes an Integer object. Third parameter is an ULONG value, which gives the object's maximum length.

Example:

```
  void main()
      {
      MUILabString str1("_Label: ","Content",20);
      MUILabString int1("_Label: ",5,20);
      .
      .
      .
      };
```

1.103 TWiMUI.guide/MUIText

```
MUIText
-------

     class MUIText : public MUIArea
         {
         public:
             MUIText(const struct TagItem *);
             MUIText(const Tag, ...);
             MUIText();
             MUIText(MUIText &);
             virtual ~MUIText();
```

```
            MUIText &operator= (const MUIText &);
        };

Derived classes:
     none

Include file:
     classes/TWiMUI/Text.h
```

   This is the C++ class for the MUI Text class. Tags are given to the constructors that are needed to create a text object. In this, tags can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Text object. To this method you can give variable tag lists or pointers to a tag list.

   Example:

```
     void main()
         {
         MUIText tct(
             MUIA_Text_Contents, "Text content",
             .
             .
             .
             TAG_DONE);
         .
         .
         .
         };
```

1.104 TWiMUI.guide/MUIVirtgroup

```
MUIVirtgroup
------------

     class MUIVirtgroup : public MUIGroup
         {
         public:
             MUIVirtgroup(const struct TagItem *);
             MUIVirtgroup(const Tag, ...);
             MUIVirtgroup();
             MUIVirtgroup(MUIVirtgroup &);
             virtual ~MUIVirtgroup();
             MUIVirtgroup &operator= (const MUIVirtgroup &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Virtgroup.h
```

This is the C++ class for the MUI Virtgroup class. Tags are given to the constructors that are needed to create a Virtgroup object. In this, tags can be given as variable parameters or as pointers in a list.

Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Virtgroup object. To this method you can give variable tag lists or pointers to a tag list.

Example:

```
  void main()
      {
      MUIVirtgroup vg(
          MUIA_Virtgroup_Input, FALSE,
          .
          .
          .
          TAG_DONE);
      .
      .
      .
      };
```

1.105 TWiMUI.guide/MUIVolumelist

```
MUIVolumelist
-------------

     class MUIVolumelist : public MUIList
         {
         public:
             MUIVolumelist(const struct TagItem *);
             MUIVolumelist(const Tag, ...);
             MUIVolumelist();
             MUIVolumelist(MUIVolumelist &);
             virtual ~MUIVolumelist();
             MUIVolumelist &operator= (const MUIVolumelist &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Volumelist.h
```

This is the C++ class for the MUI Volumelist class. Tags are given to the constructors that are needed to create a Volumelist object. In this, tags can be given as variable parameters or as pointers in a list.

Alternatively the constructor can also be used without parameters. In this case the *Create()* (See MUINotify.) method must be used to create the MUI Volumelist object. To this method you can give variable tag lists or pointers to a tag list.

```
   *Example:*
     void main()
         {
         MUIVolumelist vl(TAG_DONE);
         .
         .
         .
         };
```

1.106 TWiMUI.guide/MUIWindow

```
MUIWindow
---------

     class MUIWindow : public MUINotify
         {
         public:
             MUIWindow(const struct TagItem *);
             MUIWindow(const Tag, ...);
             MUIWindow();
             MUIWindow(MUIWindow &);
             virtual ~MUIWindow();
             MUIWindow &operator= (const MUIWindow &);
         };

Derived classes:
     none

Include file:
     classes/TWiMUI/Window.h

   This is the C++ class for the MUI Window class. Tags are given to the
constructors that are needed to create a Window object. In this, tags
can be given as variable parameters or as pointers in a list.

   Alternatively the constructor can also be used without parameters.
In this case the *Create()* (See MUINotify.) method must be used to
create the MUI Window object. To this method you can give variable tag
lists or pointers to a tag list.

   *Example:*
     void main()
         {
         MUIWindow win(
             MUIA_Window_Title, "Window Title",
             .
             .
             .
             TAG_DONE);
         .
         .
         .
         };
```

1.107 TWiMUI.guide/Inline functions

```
Inline functions
================
```

Here we show some inline-defined functions. They are some calls to the MUI_MakeObject() function. These calls have been newly defined here to have a strong type checking. You must be very careful as objects created by this functions are not automatically destroyed as they have not any destructor. They should be used, then, in a parameter list during creation of other objects. These functions are then defined also for objects which normally don't need any method or don't have any further attribute only.

For the exact meaning of these objects please refer to the MUI documentation.

```
inline Object *MakeLabel(const STRPTR lab)
inline Object *MakeLabel1(const STRPTR lab)
inline Object *MakeLabel2(const STRPTR lab)
inline Object *MakeLLabel(const STRPTR lab)
inline Object *MakeLLabel1(const STRPTR lab)
inline Object *MakeLLabel2(const STRPTR lab)
inline Object *MakeCLabel(const STRPTR lab)
inline Object *MakeCLabel1(const STRPTR lab)
inline Object *MakeCLabel2(const STRPTR lab)
inline Object *MakeFreeLabel(const STRPTR lab)
inline Object *MakeFreeLabel1(const STRPTR lab)
inline Object *MakeFreeLabel2(const STRPTR lab)
inline Object *MakeFreeLLabel(const STRPTR lab)
inline Object *MakeFreeLLabel1(const STRPTR lab)
inline Object *MakeFreeLLabel2(const STRPTR lab)
inline Object *MakeFreeCLabel(const STRPTR lab)
inline Object *MakeFreeCLabel1(const STRPTR lab)
inline Object *MakeFreeCLabel2(const STRPTR lab)
inline Object *MakeKeyLabel(const STRPTR lab, const UBYTE key)
inline Object *MakeKeyLabel1(const STRPTR lab, const UBYTE key)
inline Object *MakeKeyLabel2(const STRPTR lab, const UBYTE key)
inline Object *MakeKeyLLabel(const STRPTR lab, const UBYTE key)
inline Object *MakeKeyLLabel1(const STRPTR lab, const UBYTE key)
inline Object *MakeKeyLLabel2(const STRPTR lab, const UBYTE key)
inline Object *MakeKeyCLabel(const STRPTR lab, const UBYTE key)
inline Object *MakeKeyCLabel1(const STRPTR lab, const UBYTE key)
inline Object *MakeKeyCLabel2(const STRPTR lab, const UBYTE key)
inline Object *MakeFreeKeyLabel(const STRPTR lab, const UBYTE key)
inline Object *MakeFreeKeyLabel1(const STRPTR lab, const UBYTE key)
inline Object *MakeFreeKeyLabel2(const STRPTR lab, const UBYTE key)
inline Object *MakeFreeKeyLLabel(const STRPTR lab, const UBYTE key)
inline Object *MakeFreeKeyLLabel1(const STRPTR lab, const UBYTE key)
inline Object *MakeFreeKeyLLabel2(const STRPTR lab, const UBYTE key)
inline Object *MakeFreeKeyCLabel(const STRPTR lab, const UBYTE key)
inline Object *MakeFreeKeyCLabel1(const STRPTR lab, const UBYTE key)
inline Object *MakeFreeKeyCLabel2(const STRPTR lab, const UBYTE key)
```

```
     inline Object *MakeHBar(const ULONG size)
     inline Object *MakeVBar(const ULONG size)
     inline Object *MakeHSpace(const ULONG size)
     inline Object *MakeVSpace(const ULONG size)
```

In addiction to these functions, one other function exists which makes ID creation relatively easy (e.g. MUIA_Window_ID).

```
     inline ULONG MakeId(const UBYTE a, const UBYTE b,
                         const UBYTE c, const UBYTE d)
```

1.108 TWiMUI.guide/Acknowledgements

Acknowledgements
****

First of all I have to thank my wife, whom it was almost impossible to see me around.

Secondly I have to thank StormC/C++ developers who made possible using a full-featured C++ language on the AMIGA.

And of course all those who contributed with their comments and incitements in building the library.

1.109 TWiMUI.guide/Author

Author
******

For suggestions, constructive critics and questions you can reach me as follows:

Snail mail
     Thomas Wilhelmi
     Taunusstraße 14
     D - 61138 Niederdorfelden
Fax
     06101/531061

E-mail
     willi@twi.rhein-main.de

Please do not hesitate to apply me for any questions, comments or suggestions. Anyhow you should preferably get in touch with me via E-mail.

You can also find my name in MUI mailinglist.

1.110 TWiMUI.guide/MUI

MUI
***

MUI is a system to more easily program and more comfortably configure a GUI.

MUI is shareware. An unregistered version is available on AmiNet and on various BBS.

MUI is (C) 1993/94 Stefan Stuntz.

By sending 30,-- DM to him you will get registered as a user:

Stefan Stuntz

Eduard-Spranger-Straße 7

D-80935 München

1.111 TWiMUI.guide/Registration

Registration
****

Please fill in this form and send it via E-mail, fax or snail mail to me (See Author.).

If someone is available to translate this documentation into other languages, he will be exempted from registration fees. Please give me anyhow notes in advance.

Registration

Herewith I would like to register for TWiMUI.

First Name:_____

Family Name:_____

Address:_____

ZIP:_____ City/State:_____

Telephone:_____

EMail:_____

Please cross preferred payment method:

* COD - Cash on delivery (DM 40,-, German only)

```
   *       by enclosing a cheque to receive a floppy disk by snail
           mail (DM 30,- for Germany, DM 35,- for Europe, DM 40,-
           all other countries)

   *       by bank transfer to receive a floppy disk by snail mail
           (DM 30,- for Germany, DM 35,- for Europe, DM 40,- all
           other countries)

   *       by enclosing a cheque to receive the registered version
           via EMail. (DM 30,-)

   *       by bank transfer to receive the registered version via
           E-Mail (DM 30,-)

   *       If you like to receive this documentation in printed
           form enclosed in snail mail orders please add DM 10,-

   These are my bank coordinates:
   Postgiroamt Ffm
   Account no. 3973 35-603
   Bank code 500 100 60
```

1.112 TWiMUI.guide/Limitations

```
Limitations
***********

   You can find a limitation in the unregistered version compared to
the full version, i.e. it is possible to create a maximum of 40 objects
in a program. If you try to create more objects the MUIErroX exception
will be thrown with error 999.

   Otherwise there are no other limitations.
```

1.113 TWiMUI.guide/Disclaimer

```
Disclaimer
**********

   By using this program you accept full responsibility for any damage
which may arise from its use or misuse. The author of this software
declines all responsibilities.
```

1.114 TWiMUI.guide/Index

Index
*****

Inline functions	Inline functions
Knob	MUIKnob
LabButton	MUILabButton
LabCheckmark	MUILabCheckmark
LabCycle	MUILabCycle
Label	MUILabel
LabelHelp	MUILabelHelp
LabNumericbutton	MUILabNumericbutton
LabRadio	MUILabRadio
LabSlider	MUILabSlider
LabString	MUILabString
LayoutHook	MUIGroupLayoutHook
Levelmeter	MUILevelmeter
Limitations	Limitations
List	MUIList
ListCompareHook	MUIListCompareHook
ListConstructHook	MUIListConstructHook
ListDestructHook	MUIListDestructHook
ListDisplayHook	MUIListDisplayHook
ListMultiTestHook	MUIListMultiTestHook
Listview	MUIListview
MemX	TWiMemX
Menu	MUIMenu
Menuitem	MUIMenuitem
Menusep	MUIMenusep
Menustrip	MUIMenustrip
Method	MUI Classes
MUI	MUI
MUI Classes	MUI Classes
MUI-Attribute	MUI Classes
MUI-Method	MUI Classes
MUIAboutmui	MUIAboutmui
MUIApplication	MUIApplication
MUIApplicationBrokerHook	MUIApplicationBrokerHook
MUIApplicationRexxHook	MUIApplicationRexxHook
MUIArea	MUIArea
MUIBalance	MUIBalance
MUIBitmap	MUIBitmap
MUIBodychunk	MUIBodychunk
MUIBoopsi	MUIBoopsi
MUIButton	MUIButton
MUICheckmark	MUICheckmark
MUIColoradjust	MUIColoradjust
MUIColorfield	MUIColorfield
MUICycle	MUICycle
MUIDataspace	MUIDataspace
MUIDirlist	MUIDirlist
MUIDirlistFilterHook	MUIDirlistFilterHook
MUIErrorX	MUIErrorX
MUIFamily	MUIFamily
MUIFloattext	MUIFloattext
MUIGadget	MUIGadget
MUIGauge	MUIGauge
MUIGroup	MUIGroup
MUIGroupCol	MUIGroupCol
MUIGroupH	MUIGroupH
MUIGroupLayoutHook	MUIGroupLayoutHook

```
MUIGroupRow
MUIGroupV
MUIHBar
MUIImage
MUIKnob
MUILabButton
MUILabCheckmark
MUILabCycle
MUILabel
MUILabelHelp
MUILabNumericbutton
MUILabRadio
MUILabSlider
MUILabString
MUILevelmeter
MUIList
MUIListCompareHook
MUIListConstructHook
MUIListDestructHook
MUIListDisplayHook
MUIListMultiTestHook
MUIListview
MUIMenu
MUIMenuitem
MUIMenusep
MUIMenustrip
MUINotify
MUINumeric
MUINumericbutton
MUIPalette
MUIPendisplay
MUIPopasl
MUIPopaslStartHook
MUIPopaslStopHook
MUIPopbutton
MUIPoplist
MUIPopobject
MUIPopobjectObjStrHook
MUIPopobjectStrObjHook
MUIPopobjectWindowHook
MUIPoppen
MUIPopstring
MUIPopstringCloseHook
MUIPopstringOpenHook
MUIProp
MUIRadio
MUIRectangle
MUIRegister
MUIRequest
MUIScale
MUIScrollbar
MUIScrollgroup
MUISemaphore
MUISlider
MUIString
MUIStringEditHook
MUIText
```


```
MUIGroupRow
MUIGroupV
MUIHBar
MUIImage
MUIKnob
MUILabButton
MUILabCheckmark
MUILabCycle
MUILabel
MUILabelHelp
MUILabNumericbutton
MUILabRadio
MUILabSlider
MUILabString
MUILevelmeter
MUIList
MUIListCompareHook
MUIListConstructHook
MUIListDestructHook
MUIListDisplayHook
MUIListMultiTestHook
MUIListview
MUIMenu
MUIMenuitem
MUIMenusep
MUIMenustrip
MUINotify
MUINumeric
MUINumericbutton
MUIPalette
MUIPendisplay
MUIPopasl
MUIPopaslStartHook
MUIPopaslStopHook
MUIPopbutton
MUIPoplist
MUIPopobject
MUIPopobjectObjStrHook
MUIPopobjectStrObjHook
MUIPopobjectWindowHook
MUIPoppen
MUIPopstring
MUIPopstringCloseHook
MUIPopstringOpenHook
MUIProp
MUIRadio
MUIRectangle
MUIRegister
MUIRequest
MUIScale
MUIScrollbar
MUIScrollgroup
MUISemaphore
MUISlider
MUIString
MUIStringEditHook
MUIText
```

MUIVBar	MUIVBar
MUIVirtgroup	MUIVirtgroup
MUIVolumelist	MUIVolumelist
MUIWindow	MUIWindow
MultiTestHook	MUIListMultiTestHook
Notify	MUINotify
Numeric	MUINumeric
Numericbutton	MUINumericbutton
ObjStrHook	MUIPopobjectObjStrHook
OpenHook	MUIPopstringOpenHook
Palette	MUIPalette
Pendisplay	MUIPendisplay
Popasl	MUIPopasl
PopaslStartHook	MUIPopaslStartHook
PopaslStopHook	MUIPopaslStopHook
Popbutton	MUIPopbutton
Poplist	MUIPoplist
Popobject	MUIPopobject
PopobjectObjStrHook	MUIPopobjectObjStrHook
PopobjectStrObjHook	MUIPopobjectStrObjHook
PopobjectWindowHook	MUIPopobjectWindowHook
Poppen	MUIPoppen
Popstring	MUIPopstring
PopstringCloseHook	MUIPopstringCloseHook
PopstringOpenHook	MUIPopstringOpenHook
Preface	Preface
Prop	MUIProp
Radio	MUIRadio
Rectangle	MUIRectangle
Register	MUIRegister
Registration	Registration
Request	MUIRequest
Requirements	Requirements
RexxHook	MUIApplicationRexxHook
Scale	MUIScale
Scrollbar	MUIScrollbar
Scrollgroup	MUIScrollgroup
Semaphore	MUISemaphore
Share	TWiShare
Slider	MUISlider
StartHook	MUIPopaslStartHook
StopHook	MUIPopaslStopHook
Str	TWiStr
StrArray	TWiStrArray
StrCursor	TWiStrCursor
String	MUIString
StringEditHook	MUIStringEditHook
StrObjHook	MUIPopobjectStrObjHook
Support Classes	Support Classes
Tag	TWiTag
TagCursor	TWiTagCursor
Text	MUIText
TWiArrayCursor	TWiArrayCursor
TWiArrayList	TWiArrayList
TWiBuff	TWiBuff
TWiFormat	TWiFormat
TWiMemX	TWiMemX

TWiShare	TWiShare
TWiStr	TWiStr
TWiStrArray	TWiStrArray
TWiStrCursor	TWiStrCursor
TWiTag	TWiTag
TWiTagCursor	TWiTagCursor
VBar	MUIVBar
Virtgroup	MUIVirtgroup
Volumelist	MUIVolumelist
Window	MUIWindow
WindowHook	MUIPopobjectWindowHook