

**TWiMUI**

<b>COLLABORATORS</b>
----------------------

	<i>TITLE :</i> TWiMUI	
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>
WRITTEN BY		July 19, 2024
<i>SIGNATURE</i>		

<b>REVISION HISTORY</b>
-------------------------

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>TWiMUI</b>	<b>1</b>
1.1	TWiMUI.guide	1
1.2	TWiMUI.guide/Vorwort	1
1.3	TWiMUI.guide/Anforderungen	2
1.4	TWiMUI.guide/Klassen	2
1.5	TWiMUI.guide/Hilfsklassen	2
1.6	TWiMUI.guide/TWiArrayList	3
1.7	TWiMUI.guide/TWiArrayCursor	4
1.8	TWiMUI.guide/TWiBuff	6
1.9	TWiMUI.guide/TWiFormat	7
1.10	TWiMUI.guide/TWiMemX	9
1.11	TWiMUI.guide/TWiStr	10
1.12	TWiMUI.guide/TWiStrArray	13
1.13	TWiMUI.guide/TWiStrCursor	14
1.14	TWiMUI.guide/TWiTag	14
1.15	TWiMUI.guide/TWiTagCursor	16
1.16	TWiMUI.guide/MUI-Basisklassen	18
1.17	TWiMUI.guide/MUIApplicationBrokerHook	19
1.18	TWiMUI.guide/MUIApplicationRexxHook	20
1.19	TWiMUI.guide/MUIDirlistFilterHook	21
1.20	TWiMUI.guide/MUIGroupLayoutHook	22
1.21	TWiMUI.guide/MUIListCompareHook	23
1.22	TWiMUI.guide/MUIListConstructHook	24
1.23	TWiMUI.guide/MUIListDestructHook	25
1.24	TWiMUI.guide/MUIListDisplayHook	26
1.25	TWiMUI.guide/MUIListMultiTestHook	27
1.26	TWiMUI.guide/MUIPopaslStartHook	28
1.27	TWiMUI.guide/MUIPopaslStopHook	29
1.28	TWiMUI.guide/MUIPopobjectObjStrHook	31
1.29	TWiMUI.guide/MUIPopobjectStrObjHook	32

---

---

1.30	TWiMUI.guide/MUIPopobjectWindowHook	33
1.31	TWiMUI.guide/MUIPopstringCloseHook	34
1.32	TWiMUI.guide/MUIPopstringOpenHook	35
1.33	TWiMUI.guide/MUIStringEditHook	36
1.34	TWiMUI.guide/MUILabelHelp	37
1.35	TWiMUI.guide/MUIT	38
1.36	TWiMUI.guide/MUI-Klassen	39
1.37	TWiMUI.guide/UserDispatch	41
1.38	TWiMUI.guide/MUIAboutmui	41
1.39	TWiMUI.guide/MUIApplication	42
1.40	TWiMUI.guide/MUIArea	43
1.41	TWiMUI.guide/MUIBalance	44
1.42	TWiMUI.guide/MUIBitmap	45
1.43	TWiMUI.guide/MUIBodychunk	46
1.44	TWiMUI.guide/MUIBoopsi	47
1.45	TWiMUI.guide/MUIButton	48
1.46	TWiMUI.guide/MUILabButton	48
1.47	TWiMUI.guide/MUICheckmark	49
1.48	TWiMUI.guide/MUILabCheckmark	50
1.49	TWiMUI.guide/MUIColoradjust	50
1.50	TWiMUI.guide/MUIColorfield	51
1.51	TWiMUI.guide/MUICycle	52
1.52	TWiMUI.guide/MUILabCycle	53
1.53	TWiMUI.guide/MUIDataspace	54
1.54	TWiMUI.guide/MUIDirlist	55
1.55	TWiMUI.guide/MUIFamily	56
1.56	TWiMUI.guide/MUIFloattext	56
1.57	TWiMUI.guide/MUIGadget	57
1.58	TWiMUI.guide/MUIGauge	58
1.59	TWiMUI.guide/MUIGroup	59
1.60	TWiMUI.guide/MUIGroupH	60
1.61	TWiMUI.guide/MUIGroupV	60
1.62	TWiMUI.guide/MUIGroupCol	60
1.63	TWiMUI.guide/MUIGroupRow	60
1.64	TWiMUI.guide/MUIImage	61
1.65	TWiMUI.guide/MUIKnob	61
1.66	TWiMUI.guide/MUILabel	62
1.67	TWiMUI.guide/MUILevelmeter	63
1.68	TWiMUI.guide/MUIList	64

---

---

1.69	TWiMUI.guide/MUIListView	65
1.70	TWiMUI.guide/MUIMenu	66
1.71	TWiMUI.guide/MUIMenuitem	67
1.72	TWiMUI.guide/MUIMenusep	68
1.73	TWiMUI.guide/MUIMenustrip	68
1.74	TWiMUI.guide/MUINotify	69
1.75	TWiMUI.guide/MUINumeric	70
1.76	TWiMUI.guide/MUINumericbutton	71
1.77	TWiMUI.guide/MUILabNumericbutton	72
1.78	TWiMUI.guide/MUIPalette	72
1.79	TWiMUI.guide/MUIPendisplay	73
1.80	TWiMUI.guide/MUIPopasl	74
1.81	TWiMUI.guide/MUIPopbutton	75
1.82	TWiMUI.guide/MUIPoplist	76
1.83	TWiMUI.guide/MUIPopobject	76
1.84	TWiMUI.guide/MUIPoppen	77
1.85	TWiMUI.guide/MUIPopstring	78
1.86	TWiMUI.guide/MUIProp	79
1.87	TWiMUI.guide/MUIRadio	80
1.88	TWiMUI.guide/MUILabRadio	81
1.89	TWiMUI.guide/MUIRectangle	82
1.90	TWiMUI.guide/MUIHBar	83
1.91	TWiMUI.guide/MUIVBar	83
1.92	TWiMUI.guide/MUIRegister	84
1.93	TWiMUI.guide/MUIRequest	85
1.94	TWiMUI.guide/MUIScale	86
1.95	TWiMUI.guide/MUIScrollbar	87
1.96	TWiMUI.guide/MUIScrollgroup	88
1.97	TWiMUI.guide/MUISemaphore	89
1.98	TWiMUI.guide/MUISlider	89
1.99	TWiMUI.guide/MUILabSlider	90
1.100	TWiMUI.guide/MUIString	91
1.101	TWiMUI.guide/MUILabString	92
1.102	TWiMUI.guide/MUIText	93
1.103	TWiMUI.guide/MUIVirtgroup	94
1.104	TWiMUI.guide/MUIVolumelist	95
1.105	TWiMUI.guide/MUIWindow	96
1.106	TWiMUI.guide/inline's	97
1.107	TWiMUI.guide/Danksagungen	98

---

---

1.108TWiMUI.guide/Author . . . . .	98
1.109TWiMUI.guide/MUI . . . . .	99
1.110TWiMUI.guide/Registrierung . . . . .	99
1.111TWiMUI.guide/Einschränkung . . . . .	100
1.112TWiMUI.guide/Haftungsausschluß . . . . .	100

---

# Chapter 1

## TWiMUI

### 1.1 TWiMUI.guide

TWiMUI

\*\*\*\*\*

TWiMUI ist eine C++-Klassenbibliothek für MUI.

Vorwort	Dies ist das Vorwort
Anforderungen	Was wird benötigt
Klassen	Beschreibung der Klassen
Danksagungen	Wem muß ich danken
Author	Wer bin ich und wie kann man mich erreichen
MUI	MUI
Registrierung	Formular für die Registrierung
Einschränkung	Einschränkungen in der unregistrierten Version
Haftungsausschluß	Ich hafte für gar nichts

### 1.2 TWiMUI.guide/Vorwort

Vorwort

\*\*\*\*\*

Diese Klassenbibliothek definiert C++-Klassen für MUI (See MUI.). Für jede öffentliche MUI-Klasse existiert eine C++Klasse mit einigen Konstruktoren und natürlich allen Attributen und Methoden, die in der entsprechenden Klasse definiert sind.

Die Vererbungshierarchie leitet sich aus der Hierarchie der MUI-Klassen ab.

Es sind auch einige weitere Klassen definiert, die es in dieser Form in MUI nicht gibt. Dies sind Klassen, die originäre MUI-Klassen ergänzen.

---

## 1.3 TWiMUI.guide/Anforderungen

Anforderungen

\*\*\*\*\*

Um diese Klassenbibliothek einsetzen zu können, sollten Sie schon mit der herkömmlichen MUI-Programmierung vertraut sein.

Ansonsten benötigen Sie einen C++-Compiler, der auch Templates und Exceptions beherrscht und MUI 3.3. natürlich müssen auch deren Anforderungen erfüllt sein.

Im Moment funktioniert diese Klassenbibliothek nur mit "Far Code" und "Far Data". Aber mit der nächsten Version meines Compilers sollte diese Einschränkung wegfallen.

## 1.4 TWiMUI.guide/Klassen

Klassen

\*\*\*\*\*

Folgend nun die Beschreibungen der einzelnen Klassen. Dieses Kapitel ist aufgegliedert in drei Teile. Zuerst kommen verschiedene Hilfsklassen, die für ein komfortables Programmieren einer Klassenbibliothek notwendig sind. Danach kommen die MUI-Basisklassen. Das sind Klassen, von denen keine eigene Instanz gebildet werden kann bzw sollte, da die eigentlichen MUI-Klassen von diesen Klassen erben. Die eigentlichen MUI-Klassen werden dann im dritten Teil beschrieben. Im vierten Teil sind noch ein paar definierte inline-Funktionen aufgeführt.

Hilfsklassen	verschiedene Hilfsklassen
MUI-Basisklassen	Basisklassen für MUI-Klassen
MUI-Klassen	Das sind die eigentlichen Klassen
inline's	einige per inline definierte Funktionen

## 1.5 TWiMUI.guide/Hilfsklassen

MUI-Basisklassen

=====

Hier werden jetzt alle Hilfsklassen beschrieben, die sozusagen ein Abfallprodukt bei der Entwicklung der vorliegenden Klassenbibliothek waren.

TWiArrayList	Klasse für Arrays beliebiger Elemente
TWiArrayCursor	Klasse, um die Array-Klasse per Cursor zu durchlaufen
TWiBuff	Klasse, um einen Datenpuffer zu verwalten
TWiFormat	Klasse, um RawDoFmt komfortabel zu benutzen

TWiMemX	Exception-Klasse für fehlerhafte Speicher-Anforderung
TWiStr	Klasse, um einen String zu verwalten
TWiStrArray	Klasse für String-Arrays
TWiStrCursor	Klasse, um die String-Array-Klasse per Cursor zu durchlaufen ↔
TWiTag	Klasse, um TagItem-Strukturen zu verwalten
TWiTagCursor	Klasse, um die Tag-Klasse per Cursor zu durchlaufen

## 1.6 TWiMUI.guide/TWiArrayList

TWiArrayList

-----

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

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Misc.h

Diese Klasse realisiert ein Array für beliebige Elemente.

TWiArrayList (ULONG)

Der Konstruktor dieser Klasse initialisiert das Array und bekommt die Anfangs-Anzahl der Elemente übergeben. Sollte das Anlegen des Array fehlschlagen, so wird die Exception \*TWiMemX\* (See TWiMemX.) ausgeworfen. Der Default ist 16.

count ()

Diese Methode liefert die Anzahl der angelegten Elemente.

length ()

Diese Methode liefert die Anzahl der wirklich vorhandenen Elemente.

operator APTR

Diese Methode liefert die Anfangs-Adresse des Array.

operator [] (ULONG)

Diese Methode liefert eine Referenz des Elementes, dessen Index übergeben wird.

addTail()

Diese Methode liefert eine Referenz des Elementes, das hinter dem letzten Element liegt. Diese Methode wird dazu verwendet, ein Element hinter dem letzten, vorhandenen Element einzufügen. Sollte dort kein Platz für ein neues Element vorhanden sein, so wird versucht, das Array zu vergrößern. Schlägt dies fehl, so wird die Exception \*TWiMemX\* (See TWiMemX.) ausgeworfen. Das einzufügende Element muß an das Ergebnis der Methode zugewiesen werden.

insert (ULONG)

Diese Methode liefert eine Referenz des Elementes, dessen Index übergeben wird. Alle dahinter liegenden Elemente werden um eins nach rechts verschoben. Diese Methode wird dazu verwendet, ein Element in das Array einzufügen. Sollte kein Platz für ein neues Element vorhanden sein, so wird versucht das Array zu vergrößern. Schlägt dies fehl, so wird die Exception \*TWiMemX\* (See TWiMemX.) ausgeworfen. Das einzufügende Element muß an das Ergebnis der Methode zugewiesen werden.

remove (ULONG)

Diese Methode entfernt das Element, dessen Index übergeben wird. Alle dahinter liegenden Elemente werden um eins nach links verschoben.

remTail()

Diese Methode entfernt das letzte Element.

clear()

Diese Methode entfernt alle Elemente.

\*Beispiel:\*

```
.
.
.
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();
};
```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Misc.h

Diese Klasse realisiert einen Cursor, um damit komfortabel Schleifen über Instanzen der Klasse \*TWiArrayList\* (See TWiArrayList.) definieren zu können.

TWiArrayCursor(TWiArrayList<T>)

Der Konstruktor dieser Klasse initialisiert den Cursor für das Array, für das eine Referenz übergeben wird.

first()

Diese Methode setzt den Cursor auf das erste Element.

last()

Diese Methode setzt den Cursor auf das letzte Element.

next()

Diese Methode setzt den Cursor auf das nächste Element.

prev()

Diese Methode setzt den Cursor auf das vorherige Element.

item()

Diese Methode liefert eine Referenz auf das aktuelle Element, auf das der Cursor gerade zeigt.

isDone()

Diese Methode liefert FALSE, wenn der Cursor auf ein gültiges Element zeigt und TRUE, wenn der Cursor aus den Bounds geraten ist.

\*Beispiel:\*

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

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Misc.h

Diese Klasse stellt einen einfachen Daten-Buffer zur Verfügung.

TWiBuff(const ULONG)

Dieser Konstruktor definiert einen Buffer in der angegebenen Größe. Sollte das Anlegen des Buffers fehlschlagen, so wird die Exception \*TWiMemX\* (See TWiMemX.) ausgeworfen. Der Default ist 256.

TWiBuff(const APTR, const ULONG)

Dieser Konstruktor übernimmt den angegebenen APTR und definiert dies als Anfangsadresse für einen Buffer in der angegebenen Größe. Wird als Adresse NULL angegeben, so wird versucht, einen Buffer in der angegebenen Größe zu allozieren. Sollte das Anlegen des Buffers fehlschlagen, so wird die Exception \*TWiMemX\* (See TWiMemX.) ausgeworfen.

TWiBuff(const TWiBuff &)

Dieser Konstruktor übernimmt die Elemente aus der angegebenen Klasse, legt einen eigenen Buffer der entsprechenden Größe an und kopiert den Buffer der Parameter-Klasse in seinen eigenen. Sollte das Anlegen des Buffers fehlschlagen, so wird die Exception \*TWiMemX\* (See TWiMemX.) ausgeworfen.

operator=(const TWiBuff &)

Dieser Operator übernimmt die Elemente aus der angegebenen Klasse, legt einen eigenen Buffer der entsprechenden Größe an und kopiert den Buffer der Parameter-Klasse in seinen eigenen. Sollte das anlegen des Buffers fehlschlagen, so wird die Exception \*TWiMemX\* (See TWiMemX.) ausgeworfen.

buffer()

Diese Methode liefert die Anfangs-Adresse des Buffers.

size()

Diese Methode liefert die Größe des Buffers.

doubleBuff()

Diese Methode verdoppelt den Buffer. Sollte das Vergrößern des Buffers fehlschlagen, so wird die Exception \*TWiMemX\* (See TWiMemX.) ausgeworfen.

setBuffSize(ULONG)

Diese Methode setzt die Buffer-Größe auf den angegebenen Wert. Sollte die neue Größe kleiner als die Alte sein, wird der Inhalt abgeschnitten. Sollte das Anlegen des Buffers fehlschlagen, so wird die Exception \*TWiMemX\* (See TWiMemX.) ausgeworfen.

\*Beispiel:\*

```
.
.
.
STRPTR aaa = "Das ist nur ein 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();
    void setFormat(const STRPTR);
    void setBuffer(const ULONG);
    STRPTR format(const ULONG, ...);
    STRPTR format(const APTR);
    STRPTR getFormat() const;
    STRPTR getBuff() const;
};
```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Misc.h

Diese Klasse stellt Methoden zur Verfügung um komfortabel die Amiga-Exec-Funktion `RawDoFmt()` aufzurufen.

`TWiFormat(const STRPTR)`

Dieser Konstruktor setzt das printf-Style Format, mit welchem dann formatiert wird, auf den angegebenen Parameter. Die Größe des Ziel-Buffers wird in diesem Konstruktor auf 0 gesetzt und muß deshalb später mit der Methode `setBuffer()` nachträglich gesetzt werden.

`TWiFormat(const STRPTR, const ULONG)`

Dieser Konstruktor setzt das printf-Style Format mit welchem dann formatiert wird auf den String des ersten Parameters. Die Größe des Ziel-Buffers wird auf den Wert des zweiten Parameters gesetzt. Sollte das Anlegen des Buffers fehlschlagen, so wird die Exception `*TWiMemX*` (See `TWiMemX.`) ausgeworfen.

`TWiFormat(const ULONG)`

Dieser Konstruktor setzt Die Größe des Ziel-Buffers auf den Wert des zweiten Parameters. Sollte das Anlegen des Buffers fehlschlagen, so wird die Exception `*TWiMemX*` (See `TWiMemX.`) ausgeworfen. Das printf-Style-Format wird in diesem Konstruktor nicht gesetzt und muß deshalb später mit der Methode `setFormat()` nachträglich gesetzt werden. Sollte das Anlegen des Buffers fehlschlagen, so wird die Exception `*TWiMemX*` (See `TWiMemX.`) ausgeworfen.

`setFormat(const STRPTR)`

Diese Methode setzt das printf-Style-Format.

`setBuffer()`

Diese Methode setzt den ZielBuffer auf die angegebene Größe. Sollte das Anlegen des Buffers fehlschlagen, so wird die Exception `*TWiMemX*` (See `TWiMemX.`) ausgeworfen.

`format(ULONG, ...)`

Diese Methode ruft die Amiga-Exec Funktion `RawDoFmt()` auf. Die Argumente für die Formatierung werden als Parameterliste mitgegeben. Als Ergebnis liefert die Funktion die Adresse des Ziel-Buffers.

`format(APTR)`

Diese Methode ruft die Amiga-Exec Funktion `RawDoFmt()` auf. Die Argumente für die Formatierung als Array-Adresse mitgegeben. Als Ergebnis liefert die Funktion die Adresse des Ziel-Buffers.

`getFormat()`

Diese Methode liefert die Adresse des printf-Style-Format.

`getBuffer()`

Diese Methode liefert die Adresse des Ziel-Buffer.

`*Beispiel:*`

```
.  
.   
.   
TWiFormat form("Formatiert wird die zahl: %ld",50);
```

```

    cout << form.format(28) << endl;
    .
    .
    .

```

## 1.10 TWiMUI.guide/TWiMemX

TWiMemX

-----

```

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

```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Misc.h

Diese Klasse ist zum Auswerfen einer Exception gedacht, nachdem eine Speicheranforderung nicht geklappt hat.

TWiMemX(ULONG, ULONG)

Der Konstruktor bekommt als ersten Parameter die Größe des angeforderten Speicherbereiches übergeben. Als zweiten Parameter werden die Flags des angeforderten Speicherbereiches angegeben. Dieser zweite Parameter kann auch weggelassen werden. Der Default-Wert ist dann MEMF\_ANY.

size()

Diese Methode liefert die Größe des angeforderten Speicherbereiches. Damit hier ein sinnvoller Wert geliefert werden kann, muß natürlich dem Konstruktor auch ein solcher übergeben werden.

flags()

Diese Methode liefert die Flags des angeforderten Speicherbereiches. Damit hier ein sinnvoller Wert geliefert werden kann, muß natürlich dem Konstruktor auch ein solcher übergeben werden.

\*Beispiel:\*

```

.
.
void Klasse::Methode()
{
.
.
    APTR mem = AllocMem(1024, MEMF_CLEAR);

```

```

        if (!mem)
            throw TWiMemX(1024, MEMF_CLEAR);
        .
        .
    }

void main()
{
    try
    {
        Klasse xx();
        xx.Methode();
    }
    catch (TWiMemX(memx))
    {
        cout << "Exception TWiMemX! Größe: " << memx.size() << endl;
        cout << "                          Flags: " << memx.flags() << endl;
    }
}

```

## 1.11 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 bufsize() 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);
};

```

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Misc.h

---

Diese Klasse stellt ein komfortables String-Handling Verfügung.

`TWiStr(const STRPTR)`

Dieser Konstruktor legt einen Speicherbereich in der Länge des angegebenen Strings an und kopiert den angegebenen String hinein. Sollte das Anlegen des Bereichs fehlschlagen, so wird die Exception `*TWiMemX*` (See `TWiMemX.`) ausgeworfen. Der Default ist ein leerer String in einem 64 Byte langen Bereich.

`TWiStr(const ULONG, const STRPTR)`

Dieser Konstruktor legt einen Speicher in der Länge des ersten Parameters an und kopiert den Inhalt des zweiten Parameters hinein. Ist der angegebene String länger als der erste Parameter, so wird der Rest abgeschnitten. Sollte das Anlegen des Bereichs fehlschlagen, so wird die Exception `*TWiMemX*` (See `TWiMemX.`) ausgeworfen.

`TWiStr(const TWiStr &)`

Dieser Konstruktor erstellt eine Kopie der angegebenen Instanz. Sollte das Anlegen des Bereichs fehlschlagen, so wird die Exception `*TWiMemX*` (See `TWiMemX.`) ausgeworfen.

`TWiStr(const UBYTE)`

Dieser Konstruktor erstellt einen einstelligen String plus abschließendes Nullbyte aus dem angegeben UBYTE. Sollte das Anlegen des Bereichs fehlschlagen, so wird die Exception `*TWiMemX*` (See `TWiMemX.`) ausgeworfen.

`operator STRPTR()`

Dieser Operator liefert die Anfangs-Adresse des Strings.

`operator=(const TWiStr &)`

Dieser Operator übernimmt den String aus der angegebenen Klasse, legt einen eigenen Bereich der entsprechenden Größe an und kopiert den String der Parameter-Klasse in seinen eigenen. Sollte das Anlegen des Buffers fehlschlagen, so wird die Exception `*TWiMemX*` (See `TWiMemX.`) ausgeworfen.

`operator=(const STRPTR)`

Dieser Operator übernimmt den übergebenen String, legt einen Bereich der entsprechenden Größe an und kopiert den String in seinen eigenen. Sollte das Anlegen des Buffers fehlschlagen, so wird die Exception `*TWiMemX*` (See `TWiMemX.`) ausgeworfen.

`operator+=(const TWiStr &)`

Dieser Operator nimmt den String aus der angegebenen Klasse und hängt diesen an seinen eigenen an. Falls der eigene Bereich zu klein ist, wird er vergrößert. Sollte das Vergrößern des Bereichs fehlschlagen, so wird die Exception `*TWiMemX*` (See `TWiMemX.`) ausgeworfen.

`operator+=(const STRPTR)`

Dieser Operator nimmt den angegebenen String und hängt diesen an seinen eigenen an. Falls der eigene Bereich zu klein ist, wird er vergrößert. Sollte das Vergrößern des Bereichs fehlschlagen, so wird die Exception `*TWiMemX*` (See `TWiMemX.`) ausgeworfen.

length()

Diese Methode liefert die Länge des Strings.

buffsize()

Diese Methode liefert die Größe des Bereichs.

left(ULONG)

Diese Methode liefert eine Instanz der Klasse TWiStr, welche den linken Teil des Strings enthält. Die Anzahl der gewünschten Stellen werden als Parameter übergeben. Sollte das Anlegen des Bereichs fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

right(ULONG)

Diese Methode liefert eine Instanz der Klasse TWiStr, welche den rechten Teil des Strings enthält. Die Anzahl der gewünschten Stellen werden als Parameter übergeben. Sollte das Anlegen des Bereichs fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

mid(ULONG)

Diese Methode liefert eine Instanz der Klasse TWiStr, welche den mittleren Teil des Strings enthält. Die Anzahl der gewünschten Stellen und der Index der ersten gewünschten Stelle werden als Parameter übergeben. Der erste Parameter bezeichnet die Länge und der Zweite bezeichnet den Index. Sollte das Anlegen des Bereichs fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

doubleBuff()

Diese Methode verdoppelt den Bereich. Sollte das Vergrößern des Bereichs fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

shrinkBuff()

Diese Methode setzt die Größe des Bereichs auf die Länge des Strings plus das Nullbyte. Sollte das Anlegen des Bereichs fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

setBuffSize(ULONG)

Diese Methode setzt die Bereichs-Größe auf den angegebenen Wert. Sollte die neue Größe kleiner als die Alte sein, wird der Inhalt abgeschnitten. Sollte das Anlegen des Bereichs fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

\*Beispiel:\*

```
.  
.TWiStr str1("Das ist ein Test");  
TWiStr str2 = str1.mid(3,4);  
. .
```

## 1.12 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);
};
```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Misc.h

Diese Klasse realisiert einen Array für String-Pointer.

TWiStrArray(STRPTR, ...)

Dieser Konstruktor initialisiert den Array mit den angegebenen Pointern. die Liste muß mit NULL beendet sein. Sollte das Anlegen des Arrays fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

TWiStrArray(STRPTR \*)

Dieser Konstruktor initialisiert den Array mit den angegebenen Pointern. die Liste muß mit NULL beendet sein. Sollte das Anlegen des Arrays fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

length()

Diese Methode liefert die Anzahl der vorhandenen Elemente.

strings()

Diese Methode liefert die Anfangs-Adresse des Array.

operator[] (ULONG)

Diese Methode liefert eine Referenz des Elementes, dessen Index übergeben wird.

addTail(const STRPTR)

Diese Methode fügt den angegebenen Pointer als letztes Element dem Array hinzu. Sollte das Vergrößern des Arrays fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

insert(const ULONG, const STRPTR)

Diese Methode fügt den angegebenen Pointer in den Array an der Stelle, die der erste parameter bezeichnet, ein. Sollte das Vergrößern des Arrays fehlschlagen, so wird die Exception `*TWiMemX*` (See `TWiMemX.`) ausgeworfen.

`remTail()`

Diese Methode löscht das letzte Element aus dem Array.

`remove(const ULONG)`

Diese Methode löscht das angegebene Element aus dem Array.

`*Beispiel:*`

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

## 1.13 TWiMUI.guide/TWiStrCursor

`TWiStrCursor`

-----

Diese Klasse ist in der Verwendung identisch mit der Klasse `*TWiArrayCursor*` (See `TWiArrayCursor.`). Die Dokumentation ist deshalb bitte dort nachzulesen.

## 1.14 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 taglType, ...);
    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, ... );
};
```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Misc.h

Diese Klasse verwaltet ein Array von TagItem-Strukturen.

TWiTag()

Dieser Konstruktor legt ein leeres Array an.

TWiTag(const Tag, ...)

Dieser Konstruktor legt ein Array von TagItem-Strukturen an und kopiert die angegebene Parameter-Liste dort hinein. Das letzte Element der Liste muß TAG\_DONE sein. Sollte das Anlegen des Arrays fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

TWiTag(const struct TagItem \*)

Dieser Konstruktor legt ein Array von TagItem-Strukturen an und kopiert die angegebene TagItem-Liste dort hinein. Sollte das Anlegen des Arrays fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

TWiTag(const TWiTag &)

Dieser Konstruktor erstellt eine Kopie der angegebenen Instanz. Sollte das Anlegen des Arrays fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

operator=(const TWiTag &)

Dieser Konstruktor gibt sein eigenes Array frei und erstellt eine Kopie der angegebenen Instanz. Sollte das Anlegen des Arrays fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

tags()

Diese Methode liefert die Adresse des Array.

append(const struct TagItem \*)

Diese Methode hängt die angegebene Liste an das eigene Array an. Sollte das Vergrößern des Arrays fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

append(const Tag, ...)

Diese Methode hängt die angegebene Parameter-Liste an das eigene Array an. Das letzte Element der Liste muß TAG\_DONE sein. Sollte das Vergrößern des Arrays fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

append(const TWiTag &)

Diese Methode hängt die angegebene Instanz an das eigene Array an.

---

Sollte das Vergrößern des Arrays fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

set(const struct TagItem \*)

Diese Methode mischt die übergebene Tag-Liste in die eigene Tag-Liste ein. Wenn in der Parameter-Liste ein Tag-Wert steht, der auch in der eigenen Liste vorhanden ist, so wird der Data-Wert dieses Tags überschrieben. Tags, die in der eigenen Liste nicht vorhanden sind, werden angehängt. Sollte das Anhängen des Arrays fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

set(const Tag, ...)

Diese Methode mischt die übergebene Tag-Liste in die eigene Tag-Liste ein. Wenn in der Parameter-Liste ein Tag-Wert steht, der auch in der eigenen Liste vorhanden ist, so wird der Data-Wert dieses Tags überschrieben. Tags, die in der eigenen Liste nicht vorhanden sind, werden angehängt. Das letzte Element der Liste muß TAG\_DONE sein. Sollte das Anhängen des Arrays fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

set(const TWiTag &)

Diese Methode mischt die Tag-Liste der übergebenen Instanz in die eigene Tag-Liste ein. Wenn in der Parameter-Instanz ein Tag-Wert steht, der auch in der eigenen Liste vorhanden ist, so wird der Data-Wert dieses Tags überschrieben. Tags, die in der eigenen Liste nicht vorhanden sind, werden angehängt. Sollte das Anhängen des Arrays fehlschlagen, so wird die Exception \*TWiMemX)\* (See TWiMemX.) ausgeworfen.

find(const Tag)

Diese Methode ruft die Amiga-Utility-Funktion FindTagItem() mit dem angegebenen Parameter auf und gibt deren Wert zurück.

getData(const Tag, const ULONG)

Diese Methode ruft die Amiga-Utility-Funktion GetTagData() mit den angegebenen Parametern auf und gibt deren Wert zurück.

filter(const LONG, const Tag)

Diese Methode ruft die Amiga-Utility-Funktion FilterTagItems() mit den angegebenen Parametern auf und gibt deren Wert zurück.

\*Beispiel:\*

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

## 1.15 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;
};
```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Misc.h

Diese Klasse realisiert einen Cursor, um damit komfortabel Schleifen über Instanzen der Klasse \*TWiTag\* (See TWiTag.) oder auch über herkömmlich erstellte TagItem-Listen definieren zu können.

TWiTagCursor(const TWiTag &)

Dieser Konstruktor initialisiert den Cursor für das Array, für das eine Referenz übergeben wird.

TWiTagCursor(const struct TagItem \*)

Dieser Konstruktor initialisiert den Cursor für die TagItem-Liste, deren Adresse übergeben wird.

isDone()

Diese Methode liefert FALSE, wenn der Cursor auf ein gültiges Element zeigt und TRUE, wenn der Cursor aus den Bounds geraten ist.

first()

Diese Methode setzt den Cursor auf das erste Element.

next()

Diese Methode setzt den Cursor auf das nächste Element.

item()

Diese Methode liefert die Adresse des aktuellen Elements, auf das der Cursor gerade zeigt.

itemTag()

Diese Methode liefert den ti\_Tag-Wert des aktuellen Elements, auf den der Cursor gerade zeigt.

itemData()

Diese Methode liefert den ti\_Data-Wert des aktuellen Elements, auf den der Cursor gerade zeigt.

```

*Beispiel:*
.
.
.
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)
        .
        .
        .
}
.
.
.

```

## 1.16 TWiMUI.guide/MUI-Basisklassen

### MUI-Basisklassen

=====

Hier werden jetzt alle MUI-Basisklassen beschrieben. Diese Klassen werden zur eigenen Programm-Entwicklung normalerweise nicht benötigt. Sie werden hier deshalb beschrieben, da einzelne Methoden eventuell überladen werden sollten.

MUIApplicationBrokerHook	Hilfe für BrokerHook in einer Application
MUIApplicationRexxHook	Hilfe für RexxHook in einer Application
MUIDirlistFilterHook	Hilfe für FilterHook in einer Dirlist
MUIGroupLayoutHook	Hilfe für LayoutHook in einer Group
MUIListCompareHook	Hilfe für CompareHook in einer List
MUIListConstructHook	Hilfe für ConstructHook in einer List
MUIListDestructHook	Hilfe für DestructHook in einer List
MUIListDisplayHook	Hilfe für DisplayHook in einer List
MUIListMultiTestHook	Hilfe für MultiTestHook in einer List
MUIPopaslStartHook	Hilfe für StartHook in einem Popasl
MUIPopaslStopHook	Hilfe für StopHook in einem Popasl
MUIPopobjectObjStrHook	Hilfe für ObjStrHook in einem Popobject
MUIPopobjectStrObjHook	Hilfe für StrObjHook in einem Popobject
MUIPopobjectWindowHook	Hilfe für WindowHook in einem Popobject
MUIPopstringCloseHook	Hilfe für CloseHook in einem Popstring
MUIPopstringOpenHook	Hilfe für OpenHook in einem Popstring
MUIStringEditHook	Hilfe für EditHook in einem String
MUILabelHelp	Hilfe für Label mit Control-Character
MUIT	Exception-Klasse

## 1.17 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();
};
```

abgeleitete Klassen:

MUIApplication (See MUIApplication.)

Include-File:

classes/TWiMUI/Application.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen BrokerHook in einer Application zu definieren. Dazu gibt man in dem Tag `*MUIA_Application_BrokerHook*` einfach den Return-Wert der Methode `*broker()*` an und überlädt die Methode `*BrokerHookFunc()*`.

\*Beispiel:\*

```
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)
{
    eigener Code.
};

```

## 1.18 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();
};

```

abgeleitete Klassen:

MUIApplication (See MUIApplication.)

Include-File:

classes/TWiMUI/Application.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen RexxHook in einer Application zu definieren. Dazu gibt man in dem Tag `*MUIA_Application_RexxHook*` einfach den Return-Wert der Methode `*rexx()*` an und überlädt die Methode `*RexxHookFunc()*`.

*\*Beispiel:\**

```

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);
    { };
};

```



```

    Dir(MUIA_Dirlist_Directory, "RAM:",
        MUIA_Dirlist_FilterHook, filter(),
        TAG_DONE);
    { };
    ~Dir() { };
};

ULONG Dir::FilterHookFunc(struct Hook *h,
                          Object *o,
                          struct ExAllData *d)
{
    eigener Code.
};

```

## 1.20 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();
};

```

abgeleitete Klassen:

MUIGroup (See MUIGroup.)

Include-File:

classes/TWiMUI/Group.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen LayoutHook in einer Group zu definieren. Dazu gibt man in dem Tag \*MUIA\_Group\_LayoutHook\* einfach den Return-Wert der Methode \*layout()\* an und überlädt die Methode \*LayoutHookFunc()\*.

\*Beispiel:\*

```

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)
{
    eigener Code.
};

```

## 1.21 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();
};

```

abgeleitete Klassen:

MUIList (See MUIList.)

Include-File:

classes/TWiMUI/List.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen CompareHook in einer List zu definieren. Dazu gibt man in dem Tag \*MUIA\_List\_CompareHook\* einfach den Return-Wert der Methode \*compare()\* an und überlädt die Methode \*CompareHookFunc()\*.

```

*Beispiel:*
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)
{
    eigener Code.
};

```

## 1.22 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();
};

```

abgeleitete Klassen:

MUIList (See MUIList.)

Include-File:

classes/TWiMUI/List.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache

Art und Weise einen ConstructHook in einer List zu definieren. Dazu gibt man in dem Tag \*MUIA\_List\_ConstructHook\* einfach den Return-Wert der Methode \*construct()\* an und überlädt die Methode \*ConstructHookFunc()\*.

```
*Beispiel:*
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)
{
    eigener Code.
};
```

## 1.23 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();
    MUIListDestructHook &operator= (const MUIListDestructHook &);

public:
    struct Hook *destruct();
};
```

abgeleitete Klassen:

MUIList (See MUIList.)

Include-File:

```
classes/TWiMUI/List.h
```

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen DestructHook in einer List zu definieren. Dazu gibt man in dem Tag `*MUIA_List_DestructHook*` einfach den Return-Wert der Methode `*destruct()*` an und überlädt die Methode `*DestructHookFunc()*`.

*\*Beispiel:\**

```
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)
{
    eigener Code.
};
```

## 1.24 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();
};
```

abgeleitete Klassen:

MUIList (See MUIList.)

Include-File:

classes/TWiMUI/List.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen DisplayHook in einer List zu definieren. Dazu gibt man in dem Tag \*MUIA\_List\_DisplayHook\* einfach den Return-Wert der Methode \*display()\* an und überlädt die Methode \*DisplayHookFunc()\*.

\*Beispiel:\*

```
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)
{
    eigener Code.
};
```

## 1.25 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();
};

```

abgeleitete Klassen:

MUIList (See MUIList.)

Include-File:

classes/TWiMUI/List.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen MultiTestHook in einer List zu definieren. Dazu gibt man in dem Tag \*MUIA\_List\_MultiTestHook\* einfach den Return-Wert der Methode \*multitest()\* an und überlädt die Methode \*MultiTestHookFunc()\*.

\*Beispiel:\*

```

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)
{
    eigener Code.
};

```

## 1.26 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();
};

```

abgeleitete Klassen:

MUIPopasl (See MUIPopasl.)

Include-File:

classes/TWiMUI/Popasl.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen StartHook in einem Popasl zu definieren. Dazu gibt man in dem Tag \*MUIA\_Popasl\_StartHook\* einfach den Return-Wert der Methode \*start()\* an und überlädt die Methode \*StartHookFunc()\*.

\*Beispiel:\*

```

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);

{
    eigener Code.
};

```

## 1.27 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();
};

```

abgeleitete Klassen:

MUIPopasl (See MUIPopasl.)

Include-File:

classes/TWiMUI/Popasl.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen StopHook in einem Popasl zu definieren. Dazu gibt man in dem Tag `*MUJA_Popasl_StopHook*` einfach den Return-Wert einer der Methoden `*stopFile()*`, `*stopFont()*` oder `*stopScreenMode()*` an und überlädt die Methode `*StopHookFunc()*`.

*\*Beispiel:\**

```

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

void Pop::StopHookFunc(struct Hook *h,
                      Object *o,
                      struct FileRequester *f);

```

```

{
    eigener Code.
};

```

## 1.28 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();
};

```

abgeleitete Klassen:

MUIPopobject (See MUIPopobject.)

Include-File:

classes/TWiMUI/Popobject.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen ObjStrHook in einem Popobject zu definieren. Dazu gibt man in dem Tag \*MUIA\_Popobject\_ObjStrHook\* einfach den Return-Wert der Methode \*ObjStr()\* an und überlädt die Methode \*ObjStrHookFunc()\*.

\*Beispiel:\*

```

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);
{
    eigener Code.
};

```

## 1.29 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();
};

```

abgeleitete Klassen:

MUIPopobject (See MUIPopobject.)

Include-File:

classes/TWiMUI/Popobject.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen StrObjHook in einem Popobject zu definieren. Dazu gibt man in dem Tag \*MUIA\_Popobject\_StrObjHook\* einfach den Return-Wert der Methode \*strobj()\* an und überlädt die Methode \*StrObjHookFunc()\*.

\*Beispiel:\*

```

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);
    {
        eigener Code.
    };

```

### 1.30 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();
};

```

abgeleitete Klassen:

MUIPopobject (See MUIPopobject.)

Include-File:

classes/TWiMUI/Popobject.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen WindowHook in einem Popobject zu definieren. Dazu gibt man in dem Tag \*MUIA\_Popobject\_WindowHook\* einfach den Return-Wert der Methode \*window()\* an und überlädt die Methode \*WindowHookFunc()\*.

\*Beispiel:\*

```

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);
{
    eigener Code.
};

```

### 1.31 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();
};

```

abgeleitete Klassen:

MUIPopstring (See MUIPopstring.)

Include-File:

classes/TWiMUI/Popstring.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen CloseHook in einem Popstring zu definieren. Dazu gibt man in dem Tag \*MUIA\_Popstring\_CloseHook\* einfach den Return-Wert der Methode \*close()\* an und überlädt die Methode \*CloseHookFunc()\*.

\*Beispiel:\*

```

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);

{
    eigener Code.
};

```

## 1.32 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();
};

```

abgeleitete Klassen:

MUIPopstring (See MUIPopstring.)

Include-File:

classes/TWiMUI/Popstring.h

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache

---

Art und Weise einen OpenHook in einem Popstring zu definieren. Dazu gibt man in dem Tag \*MUIA\_Popstring\_OpenHook\* einfach den Return-Wert der Methode \*open()\* an und überlädt die Methode \*OpenHookFunc()\*.

```
*Beispiel:*
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);

{
    eigener Code.
};
```

### 1.33 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();
};
```

abgeleitete Klassen:

MUIString (See MUIString.)

Include-File:

```
classes/TWiMUI/String.h
```

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, auf einfache Art und Weise einen EditHook in einem String zu definieren. Dazu gibt man in dem Tag `*MUIA_String_EditHook*` einfach den Return-Wert der Methode `*edit()*` an und überlädt die Methode `*EditHookFunc()*`.

`*Beispiel:*`

```
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);

{
    eigener Code.
};
```

## 1.34 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();
};
```

abgeleitete Klassen:

```
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

```

Diese Klasse hat keinen öffentlichen Konstruktor, da auch keine Instanzen davon gebildet werden sollten. Sie dient dazu, ein Object mit einem Label zu versehen und gleichzeitig ein Control-Character für die Tastatur-Bedienung zu definieren. Um dieses zu tun, muß der abgeleiteten Klasse für das Label ein String übergeben werden, in welchem dem Control-Character ein Unterstrich ("\_") vorangestellt ist. Dieser Unterstrich wird aus dem String entfernt und das Control-Character wird unterstrichen.

```

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

```

## 1.35 TWiMUI.guide/MUIT

```

MUIT
----

```

```

class MUIT
{
protected:
  LONG Typ;
public:
  MUIT(const LONG);
  MUIT(const MUIT &);
  ~MUIT();
  LONG typ() const;
};

```

```

abgeleitete Klassen:
  keine

```

```

Include-File:
  classes/TWiMUI/Notify.h

```

Diese Klasse wird ausgeworfen, falls bei einer MUI-Funktion ein Fehler auftrat. Wird diese Exception dann aufgefangen, so kann mit der Methode \*typ\* der Fehler-Typ herausgefunden werden. Dies ist natürlich nur der Fall wenn MUI eine entsprechende Fehlernummer setzt.

```

*Beispiel:*
void main()

```

```

{
try
{
MUIWindow(.....);
MUIApplication(.....);
}
catch(MUIT(m))
{
cout << "MUI-Fehler: " << m.typ() << endl;
}
catch(...)
{
cout << "unbekannte Exception!" << endl;
}
};

```

## 1.36 TWiMUI.guide/MUI-Klassen

MUI-Klassen

=====

UserDispatch	Wie definiere ich einen eigenen Dispatcher
MUIAboutmui	Die Klasse Aboutmui
MUIApplication	Die Klasse Application
MUIArea	Die Klasse Area
MUIBalance	Die Klasse Balance
MUIBitmap	Die Klasse Bitmap
MUIBodychunk	Die Klasse Bodychunk
MUIBoopsi	Die Klasse Boopsi
MUIButton	Die Klasse Button
MUILabButton	Die Klasse Button mit Label-Unterstützung
MUICheckmark	Die Klasse Checkmark
MUILabCheckmark	Die Klasse Checkmark mit Label-Unterstützung
MUIColoradjust	Die Klasse Coloradjust
MUIColorfield	Die Klasse Colorfield
MUICycle	Die Klasse Cycle
MUILabCycle	Die Klasse Cycle mit Label-Unterstützung
MUIDataspace	Die Klasse Dataspace
MUIDirlist	Die Klasse Dirlist
MUIFamily	Die Klasse Family
MUIFloattext	Die Klasse Floattext
MUIGadget	Die Klasse Gadget
MUIGauge	Die Klasse Gauge
MUIGroup	Die Klasse Group
MUIGroupH	Die Klasse Group mit horizontaler Ausrichtung
MUIGroupV	Die Klasse Group mit vertikaler Ausrichtung
MUIGroupCol	Die Klasse Group mit Spalten-Anzahl
MUIGroupRow	Die Klasse Group mit Zeilen-Anzahl
MUIImage	Die Klasse Image
MUIKnob	Die Klasse Knob
MUILabel	Die Klasse Label
MUILevelmeter	Die Klasse Levelmeter
MUIList	Die Klasse List

MUIListView	Die Klasse ListView
MUIMenu	Die Klasse Menu
MUIMenuItem	Die Klasse MenuItem
MUIMenusep	Die Klasse Menusep
MUIMenustrip	Die Klasse Menustrip
MUINotify	Die Klasse Notify
MUINumeric	Die Klasse Numeric
MUINumericbutton	Die Klasse Numericbutton
MUILabNumericbutton	Die Klasse Numericbutton mit Label-Unterstützung
MUIPalette	Die Klasse Palette
MUIPdisplay	Die Klasse Pdisplay
MUIPopasl	Die Klasse Popasl
MUIPopbutton	Die Klasse Popbutton
MUIPoplist	Die Klasse Poplist
MUIPopobject	Die Klasse Popobject
MUIPoppen	Die Klasse Poppen
MUIPopstring	Die Klasse Popstring
MUIProp	Die Klasse Prop
MUIRadio	Die Klasse Radio
MUILabRadio	Die Klasse Radio mit Label-Unterstützung
MUIRectangle	Die Klasse Rectangle
MUIHBar	Die Klasse Rectangle als HBar
MUIVBar	Die Klasse Rectangle als VBar
MUIRegister	Die Klasse Register
MUIRequest	Die Klasse Request
MUIScale	Die Klasse Scale
MUIScrollbar	Die Klasse Scrollbar
MUIScrollgroup	Die Klasse Scrollgroup
MUISemaphore	Die Klasse Semaphore
MUISlider	Die Klasse Slider
MUILabSlider	Die Klasse Slider mit Label-Unterstützung
MUIString	Die Klasse String
MUILabString	Die Klasse String mit Label-Unterstützung
MUIText	Die Klasse Text
MUIVirtgroup	Die Klasse Virtgroup
MUIVolumelist	Die Klasse Volumelist
MUIWindow	Die Klasse Window

Hier werden jetzt die C++-Klassen beschrieben, die sich direkt aus den einzelnen MUI-Klassen abgeleitet haben. Zusätzlich werden auch einige weitere Klassen beschrieben, die aus diesen Klassen abgeleitet werden und dem Programmierer weitere Unterstützung bieten.

Zu jedem MUI-Attribut, welches gelesen oder gesetzt werden kann, existiert eine entsprechende Methode mit gleichen Namen, um dieses Attribut zu setzen bzw. zu lesen. Die Methoden mit den entsprechenden Parameter dienen zum Setzen der Attribute und die Methode mit den Return-Werten zum Lesen.

Auch für die Methoden der Klassen gibt es entsprechende Methoden mit gleichen Namen.

Sollte es für die Attribute oder Methoden Standard-Parameter geben, wie zum Beispiel bei `*MUIM_Application_Load*` die Möglichkeiten `*MUIV_Application_Load_ENV*` oder auch `*...ENVARC*`, so gibt es auch dafür schon fertige Methoden, denen dann kein Parameter mehr übergeben werden muß. (In diesen Fällen also `*LoadENV*` und `*LoadENVARC*`.)

## 1.37 TWiMUI.guide/UserDispatch

UserDispatch  
-----

Da diese Klassen-Bibliothek auf den Custom-Classes von MUI basiert, wird es notwendig sein, eigene Disaptcher für die MUI-Klassen zu benutzen. Deshalb gibt es für jede Klasse eine virtuelle Funktion \*ULONG UserDispatch(struct IClass \*, Object \*, Msg)\*, die überladen werden kann. In dieser Funktion können dann eigene Methoden definiert werden. Es darf natürlich nicht vergessen werden, daß alle unbekanntenen Methoden direkt an die vererbende Klasse mit DoSuperMethod() weitergeleitet werden.

## 1.38 TWiMUI.guide/MUIAboutmui

MUIAboutmui  
-----

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

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Aboutmui.h

Dies ist die C++-Klasse für die MUI-Klasse Aboutmui. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Aboutmui-Fenster zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Aboutmui-Fenster zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

\*Beispiel:\*

```
void main()
```

```

{
MUIAboutmui About (
    MUIA_Window_RefWindow, win,
    MUIA_Aboutmui_Application, app,
    TAG_DONE);
.
.
.
};

```

## 1.39 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= (MUIApplication &);
    void Loop();
    void Add(MUIWindow &);
    void Rem(MUIWindow &);
};

```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Application.h

Dies ist die C++-Klasse, für die MUI-Klasse Application. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um eine Application zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um die Application zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

Für diese Klasse gibt es noch drei zusätzliche Methoden:

Loop

Diese Methode wird dafür verwendet, um MUI in den Wait()-Loop zu schicken und die einzelnen Signale zu empfangen und auszuwerten.

Da diese Klassen-Bibliothek auf Custom-Classes aufbaut, können eigene Methoden leicht über eigene Dispatcher (See UserDispatch.) implementiert werden und es ist nicht notwendig, MUIA\_Application\_ReturnID zu verwenden, mit Ausnahme von MUIV\_Application\_ReturnID\_Quit, mit dem die Methode Loop() beendet wird..

Add

Diese Methode wird dazu verwendet, um einer Application ein Window dynamisch anzuhängen.

Rem

Diese Methode wird dazu verwendet, um ein Window aus einer Application wieder zu entfernen.

\*Beispiel:\*

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

## 1.40 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= (MUIArea &);
    MUIWindow *WinClass() const;
};
```

abgeleitete Klassen:

```
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
```

Dies ist die C++-Klasse, für die MUI-Klasse Area. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um eine Area zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode `*Create()*` (See `MUINotify.`) benutzt werden, um die Area zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

Im Normalfall wird allerdings von dieser Klasse keine Instanz gebildet, da sie für fast alle MUI-Objekte Basisklasse ist.

Für diese Klasse gibt es noch eine zusätzliche Methode:

WinClass

Diese Methode gibt einen Zeiger auf eine Instanz der Klasse `MUIWindow`, der dieses Objekt zugeordnet ist, zurück.

`*Beispiel:*`

```

void main()
{
    MUIArea Area(
        MUIA_ObjectID, 1,
        .
        .
        .
        TAG_DONE);
    .
    .
    .
};

```

## 1.41 TWiMUI.guide/MUIBalance

MUIBalance

=====

```

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

```

```

        MUIBalance &operator= (MUIBalance &);
    };

```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Balance.h

Dies ist die C++-Klasse für die MUI-Klasse Balance. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Balance-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Balance-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

\*Beispiel:\*

```

void main()
{
    MUIBalance Bal(
        MUIA_ObjectID, 1,
        .
        .
        .
        TAG_DONE);
    .
    .
    .
};

```

## 1.42 TWiMUI.guide/MUIBitmap

MUIBitmap

=====

```

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

```

abgeleitete Klassen:

MUIBodychunk (See MUIBodychunk.)

Include-File:

classes/TWiMUI/Bitmap.h

Dies ist die C++-Klasse für die MUI-Klasse Bitmap. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Bitmap-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Bitmap-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

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

## 1.43 TWiMUI.guide/MUIBodychunk

MUIBodychunk

=====

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

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Bodychunk.h

Dies ist die C++-Klasse für die MUI-Klasse Bodychunk. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Bodychunk-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Bodychunk-Object zu erstellen. Dieser Methode können

auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

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

## 1.44 TWiMUI.guide/MUIBoopsi

MUIBoopsi  
=====

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

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Boopsi.h

Dies ist die C++-Klasse für die MUI-Klasse Boopsi. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Boopsi-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Boopsi-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

```
*Beispiel:*
void main()
{
    MUIBoopsi bc(
        MUIA_Boopsi_ClassID, "colorwheel.gadget",
```

```

        .
        .
        .
    TAG_DONE);
    .
    .
    .
};

```

## 1.45 TWiMUI.guide/MUIButton

MUIButton

=====

```

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

```

abgeleitete Klassen:

MUILabButton (See MUILabButton.)

Include-File:

classes/TWiMUI/Button.h

Dies ist die C++-Klasse, um aus der MUI-Klasse Text einen Button zu erstellen. Den Konstruktoren wird ein Pointer auf den Inhalt des Buttons übergeben. Wenn gewünscht, kann danach noch ein zweiter Parameter, ein UBYTE, übergeben werden. Dieses Zeichen wird dann im Button-Inhalt unterstrichen dargestellt und wird das Control-Zeichen.

\*Beispiel:\*

```

void main()
{
    MUIButton but("Button-Inhalt", 'b');
    .
    .
    .
};

```

## 1.46 TWiMUI.guide/MUILabButton

MUILabButton

=====

```

class MUILabButton

```

```

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

```

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Button.h

Diese Klasse ist eine weitere Unterstützung bei der Erstellung eines Buttons. Dem Konstruktor wird ein Pointer auf einen String übergeben. Dieser String wird der Inhalt des Buttons. Ausserdem wird dieser String nach einem Underscore ("\_") untersucht und das darauf folgende Zeichen wird in dem String unterstrichen dargestellt. Ausserdem wird dieses Zeichen das Control-Zeichen.

```

*Beispiel:*
void main()
{
    MUILabButton but ("_Save:");
    .
    .
    .
};

```

## 1.47 TWiMUI.guide/MUICheckmark

MUICheckmark  
=====

```

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

```

abgeleitete Klassen:  
MUILabCheckmark (See MUILabCheckmark.)

Include-File:  
classes/TWiMUI/Checkmark.h

Dies ist die C++-Klasse, um aus der MUI-Klasse Image ein Checkmark zu erstellen. Dem Konstruktor wird entweder ein Zeichen oder ein Pointer auf ein Zeichen übergeben. Dieses Zeichen wird dann das

Control-Zeichen.

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

## 1.48 TWiMUI.guide/MUILabCheckmark

MUILabCheckmark

=====

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

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Checkmark.h

Diese Klasse ist eine weitere Unterstützung bei der Erstellung eines Buttons. Dem Konstruktor wird ein Pointer auf einen String übergeben. Dieser String wird der Inhalt des Checkmarks. Ausserdem wird dieser String nach einem Underscore ("\_") untersucht und das darauf folgende Zeichen wird in dem String unterstrichen dargestellt. Ausserdem wird dieses Zeichen das Control-Zeichen.

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

## 1.49 TWiMUI.guide/MUIColoradjust

MUIColoradjust

=====

```

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

```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Coloradjust.h

Dies ist die C++-Klasse für die MUI-Klasse Coloradjust. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Coloradjust-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Coloradjust-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

```

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

```

## 1.50 TWiMUI.guide/MUIColorfield

MUIColorfield

=====

```

class MUIColorfield : public MUIGroup
{
public:
    MUIColorfield(const struct TagItem *);
    MUIColorfield(const Tag, ...);
    MUIColorfield();
};

```

```

        MUIColorfield(MUIColorfield &);
        virtual ~MUIColorfield();
        MUIColorfield &operator= (MUIColorfield &);
};

```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Colorfield.h

Dies ist die C++-Klasse für die MUI-Klasse Colorfield. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Colorfield-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden um das Colorfield-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

\*Beispiel:\*

```

void main()
{
    MUIColorfield cf(
        MUIA_Colorfield_Blue, 0,
        .
        .
        .
        TAG_DONE);
    .
    .
    .
};

```

## 1.51 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= (MUICycle &);
};

```

abgeleitete Klassen:

MUILabCycle (See MUILabCycle.)

Include-File:

classes/TWiMUI/Cycle.h

Dies ist die C++-Klasse für die MUI-Klasse Cycle. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Cycle-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Cycle-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

Bei dieser Klasse sind weitere Möglichkeiten zur Konstruktion einer Instanz gegeben. Es kann ein Pointer auf ein mit NULL abgeschlossenes Array von String-Pointern übergeben werden. Diese Strings bilden dann den Inhalt des Cycle-Gadgets. Wenn gewünscht, kann danach noch ein zweiter Parameter, ein UBYTE, übergeben werden. Dieses Zeichen wird dann das Control-Zeichen.

\*Beispiel:\*

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

## 1.52 TWiMUI.guide/MUILabCycle

MUILabCycle

=====

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

abgeleitete Klassen:

keine

Include-File:

```
classes/TWiMUI/Cycle.h
```

Diese Klasse ist eine weitere Unterstützung bei der Erstellung eines Cycle-Objekts. Dem Konstruktor wird ein Pointer auf einen String übergeben. Dieser String wird dem Cycle-Objekt als Label vorangestellt. Ausserdem wird dieser String nach einem Underscore ("\_") untersucht und das darauf folgende Zeichen wird in dem Label unterstrichen dargestellt. Ausserdem wird dieses Zeichen das Control-Zeichen.

\*Beispiel:\*

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

## 1.53 TWiMUI.guide/MUIDataspace

MUIDataspace

=====

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

abgeleitete Klassen:

keine

Include-File:

```
classes/TWiMUI/Dataspace.h
```

Dies ist die C++-Klasse für die MUI-Klasse Dataspace. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Dataspace-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode `*Create()*` (See `MUINotify.`) benutzt werden, um das Dataspace-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

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

## 1.54 TWiMUI.guide/MUIDirlist

MUIDirlist

=====

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

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Dirlist.h

Dies ist die C++-Klasse für die MUI-Klasse `Dirlist`. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein `Dirlist`-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode `*Create()*` (See `MUINotify.`) benutzt werden, um das `Dirlist`-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

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

```

        .
        .
        TAG_DONE);
    .
    .
    .
};

```

## 1.55 TWiMUI.guide/MUIFamily

MUIFamily  
=====

abgeleitete Klassen:

```

    MUIMenustrip (See MUIMenustrip.)
    MUIMenu      (See MUIMenu.)
    MUIMenuItem  (See MUIMenuItem.)

```

Include-File:

```

    classes/TWiMUI/Family.h

```

Dies ist die C++-Klasse für die MUI-Klasse Family. Von dieser Klasse kann keine Instanz gebildet werden, da sie als Basisklasse für andere MUI-Klassen verwendet wird.

## 1.56 TWiMUI.guide/MUIFloattext

MUIFloattext  
=====

```

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

```

abgeleitete Klassen:

keine

Include-File:

```

    classes/TWiMUI/Floattext.h

```

Dies ist die C++-Klasse für die MUI-Klasse Floattext. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Floattext-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode `*Create()*` (See `MUINotify.`) benutzt werden, um das `Floattext`-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

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

## 1.57 TWiMUI.guide/MUIGadget

MUIGadget

=====

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

abgeleitete Klassen:

```
MUIString (See MUIString.)
MUIBoopsi (See MUIBoopsi.)
MUIProp (See MUIProp.)
```

Include-File:

```
classes/TWiMUI/Gadget.h
```

Dies ist die C++-Klasse für die MUI-Klasse `Gadget`. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein `Gadget`-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode `*Create()*` (See `MUINotify.`) benutzt werden, um das `Gadget`-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

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

## 1.58 TWiMUI.guide/MUIGauge

MUIGauge  
=====

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

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Gauge.h

Dies ist die C++-Klasse für die MUI-Klasse Gauge. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Gauge-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Gauge-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

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

## 1.59 TWiMUI.guide/MUIGroup

MUIGroup

=====

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

abgeleitete Klassen:

```
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
```

Dies ist die C++-Klasse für die MUI-Klasse Group. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Group-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Group-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

\*Beispiel:\*

```
void main()
{
    MUIGroup grp(
        MUIA_Group_PageMode, TRUE,
        .
        .
        .
        TAG_DONE);
```

```
.  
. .  
. .  
};
```

## 1.60 TWiMUI.guide/MUIGroupH

MUIGroupH  
=====

Diese Gruppe ist identisch mit der Gruppe MUIGroup (See MUIGroupH.) mit der Ausnahme, daß sie automatisch eine horizontale Gruppe ist.

## 1.61 TWiMUI.guide/MUIGroupV

MUIGroupV  
=====

Diese Gruppe ist identisch mit der Gruppe MUIGroup (See MUIGroupH.) mit der Ausnahme, daß sie automatisch eine vertikale Gruppe ist.

## 1.62 TWiMUI.guide/MUIGroupCol

MUIGroupCol  
=====

Diese Gruppe ist identisch mit der Gruppe MUIGroup (See MUIGroupH.) mit der Ausnahme, daß als erster Parameter ein LONG-Wert übergeben wird, der die Anzahl der Spalten definiert.

## 1.63 TWiMUI.guide/MUIGroupRow

MUIGroupRow  
=====

Diese Gruppe ist identisch mit der Gruppe MUIGroup (See MUIGroupH.) mit der Ausnahme, daß als erster Parameter ein LONG-Wert übergeben wird, der die Anzahl der Zeilen definiert.

---

## 1.64 TWiMUI.guide/MUIImage

MUIImage

=====

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

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Image.h

Dies ist die C++-Klasse für die MUI-Klasse Image. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Image-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Image-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

\*Beispiel:\*

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

## 1.65 TWiMUI.guide/MUIKnob

MUIKnob

=====

```
class MUIKnob : public MUINumeric
{
```

```

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

```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Knob.h

Dies ist die C++-Klasse für die MUI-Klasse Knob. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Knob-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Knob-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

```

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

```

## 1.66 TWiMUI.guide/MUILabel

MUILabel

=====

```

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

```

abgeleitete Klassen:

keine

Include-File:

```
classes/TWiMUI/Label.h
```

Dies ist die C++-Klasse, um aus der Klasse MUIText (See MUIText.) ein Label zu kreieren. Für alle, in der MUI-Header-Datei definierten Label gibt es eine korrespondierende MUILabel-Klasse. Den Konstruktoren dieser Klassen werden ein Pointer auf einen String übergeben, welcher dann das Label darstellt. Den Key-Labels wird als zweiter Parameter noch ein UBYTE übergeben. Dieses Zeichen wird dann in dem Label unterstrichen dargestellt.

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

## 1.67 TWiMUI.guide/MUILevelmeter

```
MUILevelmeter
=====
```

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

```
abgeleitete Klassen:
keine
```

```
Include-File:
classes/TWiMUI/Levelmeter.h
```

Dies ist die C++-Klasse für die MUI-Klasse Levelmeter. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Levelmeter-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Levelmeter-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

```
*Beispiel:*
```

```

void main()
{
    MUILevelmeter lm(MUIA_Levelmeter_Label, "Label", TAG_DONE);
    .
    .
    .
};

```

## 1.68 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= (MUIList &);
};

```

abgeleitete Klassen:

```

MUIDirlist      (See MUIDirlist.)
MUIFloattext    (See MUIFloattext.)
MUIVolumelist   (See MUIVolumelist.)

```

Include-File:

```

classes/TWiMUI/List.h

```

Dies ist die C++-Klasse für die MUI-Klasse List. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein List-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das List-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

\*Beispiel:\*

```

void main()
{
    MUIList lst(
        MUIA_List_Active, MUIV_List_Active_Top,
        .
        .

```

```

    .
    TAG_DONE);
    .
    .
    .
};

```

## 1.69 TWiMUI.guide/MUIListView

MUIListView  
 =====

```

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

```

abgeleitete Klassen:  
 keine

Include-File:  
 classes/TWiMUI/Listview.h

Dies ist die C++-Klasse für die MUI-Klasse Listview. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Listview-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Listview-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

```

*Beispiel:*
void main()
{
    MUIListView lv(
        MUIA_Listview_Input, TRUE,
        .
        .
        .
        TAG_DONE);
    .
    .
    .
};

```

## 1.70 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= (MUIMenu &);
};
```

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Menu.h

Dies ist die C++-Klasse für die MUI-Klasse Menu. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Menu-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Menu-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

Für diese Klasse sind noch zwei weitere Konstruktoren implementiert. Beiden wird als erster Parameter ein Pointer auf einen String übergeben. Dieser String wird der Titel des Menüs. Danach können dann entweder eine Liste von Pointern auf Menuitem-Objekte oder eine Liste von Pointern auf Instanzen der MUIMenuitem-Klasse übergeben werden. Beide Listen werden mit NULL terminiert.

```
*Beispiel:*
void main()
{
    MUIMenu men(
        MUIA_Menu_Title, "Titel",
        .
        .
        .
        TAG_DONE);
    .
    .
    .
};
```

## 1.71 TWiMUI.guide/MUIMenuitem

MUIMenuitem

=====

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

abgeleitete Klassen:

MUIMenusep (See MUIMenusep.)

Include-File:

classes/TWiMUI/Menu.h

Dies ist die C++-Klasse für die MUI-Klasse Menuitem. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Menuitem-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Menuitem-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

Für diese Klasse ist noch ein weiterer Konstruktor implementiert. Diesem wird als erster Parameter ein Pointer auf einen String übergeben. Dieser String wird der Titel des Menüitems. Danach wird dann eine Liste von Pointern auf Menuitem-Objekte übergeben. Diese Liste wird mit NULL terminiert.

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

## 1.72 TWiMUI.guide/MUIMenusep

MUIMenusep

=====

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

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Menu.h

Diese Klasse ist eine Hilfsklasse um einen Separator-Balken in einem Menü einfach erstellen zu können.

\*Beispiel:\*

```
void main()
{
    MUIMenusep sep();
    .
    .
    .
};
```

## 1.73 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= (MUIMenustrip &);
};
```

abgeleitete Klassen:

keine

Include-File:  
 classes/TWiMUI/Menu.h

Dies ist die C++-Klasse für die MUI-Klasse Menustrip. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Menustrip-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Menustrip-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

Für diese Klasse sind noch zwei weitere Konstruktoren implementiert. Es wird entweder eine Liste von Pointern auf Menu-Objekte oder eine Liste von Pointern auf Instanzen der MUIMenu-Klasse übergeben. Beide Listen werden mit NULL terminiert.

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

## 1.74 TWiMUI.guide/MUINotify

MUINotify  
 =====

abgeleitete Klassen:  
 MUIApplication (See MUIApplication.)  
 MUIArea (See MUIArea.)  
 MUIFamily (See MUIFamily.)  
 MUIWindow (See MUIWindow.)

Include-File:  
 classes/TWiMUI/Notify.h

Dies ist die C++-Klasse für die MUI-Klasse Notify. Von dieser Klasse kann keine Instanz gebildet werden, da sie als Basisklasse für andere MUI-Klassen verwendet wird.

Allgemein verfügbar ist die Methode \*Create()\*. Diese Methode dient zum nachträglichen Kreieren eines MUI-Objektes, wenn der entsprechende Konstruktor ohne Parameter aufgerufen wurde. Dieser Methode werden die Tags übergeben, die notwendig sind, um das entsprechende MUI-Objekt zu

erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Für diese Klasse gibt es noch eine zusätzliche Methode:

AppClass

Diese Methode liefert einen Pointer auf eine Instanz der Klasse MUIApplication, der dieses Objekt zugeordnet ist.

## 1.75 TWiMUI.guide/MUINumeric

MUINumeric

=====

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

abgeleitete Klassen:

```
MUIKnob          (See MUIKnob.)
MUILevelmeter    (See MUILevelmeter.)
MUINumericbutton (See MUINumericbutton.)
MUISlider        (See MUISlider.)
```

Include-File:

```
classes/TWiMUI/Numeric.h
```

Dies ist die C++-Klasse für die MUI-Klasse Numeric. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Numeric-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine List übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Numeric-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

\*Beispiel:\*

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

```
.
};
```

## 1.76 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= (MUINumericbutton &);
};
```

abgeleitete Klassen:

```
MUILabNumericbutton (See MUILabNumericbutton.)
```

Include-File:

```
classes/TWiMUI/Numericbutton.h
```

Dies ist die C++-Klasse für die MUI-Klasse Numericbutton. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Numericbutton-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Numericbutton-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

Zusätzlich dazu sind noch zwei weitere Konstruktoren implementiert, die das erstellen von Numericbutton-Objekten vereinfachen. Beiden werden am Anfang ein Pointer auf einen String übergeben. Dieser String bezeichnet das printf-Style-Format in welcher Form der Inhalt des Objektes dargestellt werden soll. Der zweite und dritte Parameter bezeichnen das Minimum und das Maximum, das dieses Objekt annehmen darf. Wenn gewünscht, kann dann noch ein UBYTE übergeben werden, welches dann das Control-Zeichen wird.

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

```
};
```

## 1.77 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= (MUILabNumericbutton &);
};
```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Numericbutton.h

Diese Klasse unterstützt noch etwas mehr bei der Erstellung eines Numericbutton-Objektes. Dem Konstruktor wird als erster Parameter ein Pointer auf einen String übergeben. Dieser String kann dem Objekt als Label vorangestellt werden. Ausserdem wird dieser String nach einem Underscore ("\_") untersucht und das darauf folgende Zeichen wird in dem String unterstrichen dargestellt. Ausserdem wird dieses Zeichen das Control-Zeichen. Der zweite Parameter ist noch ein Pointer auf einen String. Dieser String bezeichnet das printf-Style-Format, in welcher Form der Inhalt des Objektes dargestellt werden soll. Der dritte und vierte Parameter bezeichnen das Minimum und das Maximum, das dieses Objekt annehmen darf.

\*Beispiel:\*

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

## 1.78 TWiMUI.guide/MUIPalette

MUIPalette

=====

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

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Palette.h

Dies ist die C++-Klasse für die MUI-Klasse Palette. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Palette-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Palette-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

\*Beispiel:\*

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

## 1.79 TWiMUI.guide/MUIPdisplay

MUIPdisplay

=====

```
class MUIPdisplay : public MUIArea
{
public:
    MUIPdisplay(const struct TagItem *);
```

```

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

```

abgeleitete Klassen:

MUIPoppen (See MUIPoppen.)

Include-File:

classes/TWiMUI/Pdisplay.h

Dies ist die C++-Klasse für die MUI-Klasse Pdisplay. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Pdisplay-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Pdisplay-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

Diese Klasse wird hauptsächlich als Basisklasse für die Klasse MUIPoppen verwendet.

## 1.80 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= (MUIPopasl &);
};

```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Popasl.h

Dies ist die C++-Klasse für die MUI-Klasse Popasl. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Popasl-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als

Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode `*Create()*` (See `MUINotify.`) benutzt werden, um das `Popasl`-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

```
*Beispiel:*
void main()
{
    MUIPopasl pa(
        MUIA_Popasl_Type, ASL_FileRequest,
        .
        .
        .
        TAG_DONE);
    .
    .
    .
};
```

## 1.81 TWiMUI.guide/MUIPopbutton

MUIPopbutton

=====

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

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Popbutton.h

Diese Klasse unterstützt die Erstellung eines Buttons zur Verwendung in den Pop-Klassen. Dem Konstruktor wird die Nummer eines vordefinierten Button-Image übergeben.

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

## 1.82 TWiMUI.guide/MUIPoplist

MUIPoplist

=====

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

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Poplist.h

Dies ist die C++-Klasse für die MUI-Klasse Poplist. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Poplist-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Poplist-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

## 1.83 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= (MUIPopobject &);
    };

```

abgeleitete Klassen:

MUIPoplist (See MUIPoplist.)

Include-File:

classes/TWiMUI/Popobject.h

Dies ist die C++-Klasse für die MUI-Klasse Popobject. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Popobject-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Popobject-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

\*Beispiel:\*

```

void main()
{
    MUIPopobject po(
        MUIA_Popobject_Follow, TRUE,
        .
        .
        .
        TAG_DONE);
    .
    .
    .
};

```

## 1.84 TWiMUI.guide/MUIPoppen

MUIPoppen

=====

```

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

```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Poppen.h

Dies ist die C++-Klasse für die MUI-Klasse Poppen. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Poppen-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUIINotify.) benutzt werden, um das Poppen-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

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

## 1.85 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= (MUIPopstring &);
};
```

abgeleitete Klassen:

```
MUIPopasl      (See MUIPopasl.)
MUIPopobject  (See MUIPopobject.)
```

Include-File:

```
classes/TWiMUI/Popstring.h
```

Dies ist die C++-Klasse für die MUI-Klasse Popstring. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Popstring-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden.

In diesem Fall muß die Methode `*Create()*` (See `MUINotify.`) benutzt werden, um das `Popstring-Object` zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

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

## 1.86 TWiMUI.guide/MUIProp

MUIProp  
=====

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

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Prop.h

Dies ist die C++-Klasse für die MUI-Klasse `Prop`. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein `Prop-Object` zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode `*Create()*` (See `MUINotify.`) benutzt werden, um das `Prop-Object` zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

```
*Beispiel:*
void main()
```

```

{
MUIProp prop(
    MUIA_Prop_Horiz, TRUE,
    .
    .
    .
    TAG_DONE);
.
.
.
};

```

## 1.87 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= (MUIRadio &);
};

```

abgeleitete Klassen:

MUILabRadio (See MUILabRadio.)

Include-File:

classes/TWiMUI/Radio.h

Dies ist die C++-Klasse für die MUI-Klasse Radio. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Radio-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Radio-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

Zusätzlich dazu sind noch zwei weitere Konstruktoren implementiert, die das erstellen von Radio-Objekten vereinfachen. Beiden wird ein Pointer auf ein Array aus String-Pointern übergeben. Diese Strings definieren sowohl die Anzahl der Radio-Buttons als auch die Bezeichnung der Buttons. Das Array muß mit NULL terminiert werden. Wenn gewünscht, kann dann noch ein UBYTE übergeben werden, welches dann das Control-Zeichen wird.

\*Beispiel:\*

```

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

```

## 1.88 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= (MUILabRadio &);
};

```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Radio.h

Diese Klasse unterstützt noch etwas mehr bei der Erstellung eines Radio-Objektes. Dem Konstruktor wird als erster Parameter ein Pointer auf einen String übergeben. Dieser String kann dem Objekt als Label vorangestellt werden. Ausserdem wird dieser String nach einem Underscore ("\_") untersucht und das darauf folgende Zeichen wird in dem String unterstrichen dargestellt. Ausserdem wird dieses Zeichen das Control-Zeichen. Der zweite Parameter ist ein Pointer auf ein Array aus String-Pointern. Diese Strings definieren sowohl die Anzahl der Radio-Buttons als auch die Bezeichnung der Buttons. Das Array muß mit NULL terminiert werden.

\*Beispiel:\*

```

void main()
{
    STRPTR array[] =
    {

```

```

        "Entry 1",
        "Entry 2",
        "Entry 3",
        NULL
    };
    MUILabRadio nb("_Label: ",array);
    .
    .
    .
};

```

## 1.89 TWiMUI.guide/MUIRectangle

MUIRectangle

=====

```

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

```

abgeleitete Klassen:

```

    MUIHBar (See MUIHBar.)
    MUIVBar (See MUIVBar.)

```

Include-File:

```

    classes/TWiMUI/Rectangle.h

```

Dies ist die C++-Klasse für die MUI-Klasse Rectangle. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Rectangle-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Rectangle-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

\*Beispiel:\*

```

void main()
{
    MUIRectangle rec(
        MUIA_Rectangle_BarTitle, "Titel des Rectangles",
        MUIA_Rectangle_HBar,     TRUE,
        TAG_DONE);
    .
    .
    .
}

```

```
};
```

## 1.90 TWiMUI.guide/MUIHBar

MUIHBar

=====

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

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Rectangle.h

Diese Klasse ist eine weitere Unterstützung, um horizontalen Leerraum zu erstellen. Dem Konstruktor wird die Größe des Platzes übergeben.

\*Beispiel:\*

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

## 1.91 TWiMUI.guide/MUIVBar

MUIVBar

=====

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

abgeleitete Klassen:

keine

---

Include-File:

```
classes/TWiMUI/Rectangle.h
```

Diese Klasse ist eine weitere Unterstützung, um vertikalen Leerraum zu erstellen. Dem Konstruktor wird die Größe des Platzes übergeben.

\*Beispiel:\*

```
void main()
{
    MUIVBar vb(50);
    .
    .
    .
};
```

## 1.92 TWiMUI.guide/MUIRegister

MUIRegister

=====

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

abgeleitete Klassen:

keine

Include-File:

```
classes/TWiMUI/Register.h
```

Dies ist die C++-Klasse für die MUI-Klasse Register. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Register-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Register-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

\*Beispiel:\*

```
void main()
{
    MUIRegister reg(
        MUIA_Register_Framed, TRUE,
    .
};
```

```

    .
    .
    TAG_DONE);
    .
    .
    .
};

```

## 1.93 TWiMUI.guide/MUIRequest

MUIRequest

=====

```

class MUIRequest
{
public:
    MUIRequest(const MUIApplication &, ←
               NG c, const ULONG *p)
        const MUIWindow &,
        const LONGBITS,
        const STRPTR,
        const STRPTR,
        const STRPTR,
        const ULONG,
        const ULONG *);
    MUIRequest(const MUIWindow &, ←
               NG c, const ULONG *p)
        const LONGBITS,
        const STRPTR,
        const STRPTR,
        const STRPTR,
        const ULONG,
        const ULONG *);
    MUIRequest(const MUIApplication &, ←
               NG c, const ULONG *p)
        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= (MUIRequest &);

```

```
};
```

abgeleitete Klassen:

```
keine
```

Include-File:

```
classes/TWiMUI/Request.h
```

Dies ist eine Klasse um MUI-Requester leicht erstellen zu können. Den Konstruktoren können die Parameter wie bei einem MUIRequest-Call mitgegeben werden. Diese Parameter können, solange der Requester nicht offen ist, auch mit der entsprechenden set...-Methode noch geändert werden.

Geöffnet wird der Requester mit der methode show(). Dieser Methode kann auch noch eine MUIApplication, ein MUIWindow, beides oder auch keines der beiden mitgegeben werden.

\*Beispiel:\*

```
void main()
{
    MUIRequest reg(0L, "Titel", "_Ok", "Der Wert ist: %ld", wert);
    reg.show(app, win);
    .
    .
    .
};
```

## 1.94 TWiMUI.guide/MUIScale

MUIScale

=====

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

abgeleitete Klassen:

```
keine
```

Include-File:

```
classes/TWiMUI/Scale.h
```

Dies ist die C++-Klasse für die MUI-Klasse Scale. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Scale-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode `*Create()*` (See `MUINotify.`) benutzt werden, um das `Scale-Object` zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

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

## 1.95 TWiMUI.guide/MUIScrollbar

MUIScrollbar

=====

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

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Scrollbar.h

Dies ist die C++-Klasse für die MUI-Klasse `Scrollbar`. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein `Scrollbar-Object` zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode `*Create()*` (See `MUINotify.`) benutzt werden, um das `Scrollbar-Object` zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

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

```

.
.
.
};

```

## 1.96 TWiMUI.guide/MUIScrollgroup

MUIScrollgroup  
=====

```

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

```

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Scrollgroup.h

Dies ist die C++-Klasse für die MUI-Klasse Scrollgroup. Den Konstruktoren werden die Tags übergeben, die notwendig sind um ein Scrollgroup-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Scrollgroup-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

```

*Beispiel:*
void main()
{
    MUIScrollgroup sg(
        MUJA_Scrollgroup_FreeHoriz, TRUE,
        .
        .
        .
        TAG_DONE);
    .
    .
    .
};

```

## 1.97 TWiMUI.guide/MUISemaphore

MUISemaphore

=====

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

abgeleitete Klassen:

MUIDataspace (See MUIDataspace.)

Include-File:

classes/TWiMUI/Semaphore.h

Dies ist die C++-Klasse für die MUI-Klasse Semaphore. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Semaphore-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Semaphore-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

\*Beispiel:\*

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

## 1.98 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= (MUISlider &);
};

```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/Slider.h

Dies ist die C++-Klasse für die MUI-Klasse Slider. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Slider-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Slider-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

Zusätzlich dazu sind noch zwei weitere Konstruktoren implementiert, die das erstellen von Slider-Objekten vereinfachen. Die ersten beiden Parameter bezeichnen das Minimum und das Maximum, das dieses Objekt annehmen darf. Wenn gewünscht, kann dann noch ein UBYTE übergeben werden, welches dann das Control-Zeichen wird.

```

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

```

## 1.99 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= (MUILabSlider &);
};

```

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Slider.h

Diese Klasse unterstützt noch etwas mehr bei der Erstellung eines Slider-Objektes. Dem Konstruktor wird als erster Parameter ein Pointer auf einen String übergeben. Dieser String kann dem Objekt als Label vorangestellt werden. Ausserdem wird dieser String nach einem Underscore ("\_") untersucht und das darauf folgende Zeichen wird in dem String unterstrichen dargestellt. Ausserdem wird dieses Zeichen das Control-Zeichen. Der zweite und dritte Parameter bezeichnen das Minimum und das Maximum das dieses Objekt annehmen darf.

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

## 1.100 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= (MUIString &);
};
```

abgeleitete Klassen:  
MUILabString (See MUILabString.)

Include-File:  
classes/TWiMUI/String.h

Dies ist die C++-Klasse für die MUI-Klasse String. Den Konstruktoren

werden die Tags übergeben, die notwendig sind, um ein String-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode `*Create()*` (See `MUINotify.`) benutzt werden, um das String-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

Zusätzlich dazu sind noch sechs weitere Konstruktoren implementiert, die das Erstellen von String- oder Integer-Objekten vereinfachen.

Die ersten beiden Konstruktoren sind dazu da, um einfache String-Objekte zu erstellen. Beiden wird als erster Parameter ein Pointer auf einen String übergeben, welcher dann der Anfangs-Inhalt des Objektes wird. Der zweite Parameter bezeichnet die maximale Länge des String-Objektes. Wenn gewünscht, kann dann noch ein UBYTE angegeben werden. Dieses Zeichen wird dann das Control-Zeichen.

Die anderen Konstruktoren sind zum Erstellen eines Integer-Objektes gedacht. Wenn die zwei ersten Parameter beide ULONG-Werte sind, dann bezeichnet der erste davon den Inhalt des Integer-Objektes und wird in einen String umgewandelt, damit der Inhalt auch dargestellt werden kann. Der zweite Parameter bezeichnet dann die maximale Länge des Objektes. Wird aber nur ein ULONG-Parameter angegeben, so gibt es keinen Anfangs-Inhalt des Objektes und der angegebene Parameter bezeichnet die maximale Länge des Objektes. Allen dieser Objekte kann noch ein UBYTE-Parameter übergeben werden. Dieses Zeichen wird dann das Control-Zeichen.

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

## 1.101 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= (MUILabString &);
};

```

abgeleitete Klassen:

keine

Include-File:

classes/TWiMUI/String.h

Diese Klasse unterstützt noch etwas mehr bei der Erstellung eines String-Objektes. Dem Konstruktor wird als erster Parameter ein Pointer auf einen String übergeben. Dieser String kann dem Objekt als Label vorangestellt werden. Ausserdem wird dieser String nach einem Underscore ("\_") untersucht und das darauf folgende Zeichen wird in dem String unterstrichen dargestellt. Ausserdem wird dieses Zeichen das Control-Zeichen. Der zweite Parameter ist dann entweder wieder ein Pointer auf einen String oder aber ein ULONG. Ist dieser Parameter ein Pointer auf einen String, so wird dieser String der Inhalt des String-Objektes. Ist es ein ULONG-Wert, so wird dieser umgewandelt in einen String und dieser dann der Inhalt des Objektes. Ausserdem wird das Objekt ein Integer-Objekt. Der dritte Parameter ist ein ULONG-Wert, welcher die maximale Länge des Objektes angibt.

```

*Beispiel:*
void main()
{
    MUILabString str1("_Label: ", "Inhalt", 20);
    MUILabString int1("_Label: ", 5, 20);
    .
    .
    .
};

```

## 1.102 TWiMUI.guide/MUIText

MUIText

=====

```

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

```

abgeleitete Klassen:

keine

Include-File:

```
classes/TWiMUI/Text.h
```

Dies ist die C++-Klasse für die MUI-Klasse Text. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Text-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Text-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

\*Beispiel:\*

```
void main()
{
    MUIText tct(
        MUIA_Text_Contents, "Inhalt des Textes",
        .
        .
        .
        TAG_DONE);
    .
    .
    .
};
```

## 1.103 TWiMUI.guide/MUIVirtgroup

MUIVirtgroup

=====

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

abgeleitete Klassen:

keine

Include-File:

```
classes/TWiMUI/Virtgroup.h
```

Dies ist die C++-Klasse für die MUI-Klasse Virtgroup. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Virtgroup-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode `*Create()*` (See `MUINotify.`) benutzt werden. um das `Virtgroup`-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

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

## 1.104 TWiMUI.guide/MUIVolumelist

MUIVolumelist  
=====

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

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Volumelist.h

Dies ist die C++-Klasse für die MUI-Klasse `Volumelist`. Den Konstruktoren werden die Tags übergeben. die notwendig sind. um ein `Volumelist`-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode `*Create()*` (See `MUINotify.`) benutzt werden. um das `Volumelist`-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

```
*Beispiel:*
```

---

```

void main()
{
    MUIVolumelist vl(TAG_DONE);
    .
    .
    .
};

```

## 1.105 TWiMUI.guide/MUIWindow

MUIWindow  
=====

```

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

```

abgeleitete Klassen:  
keine

Include-File:  
classes/TWiMUI/Window.h

Dies ist die C++-Klasse für die MUI-Klasse Window. Den Konstruktoren werden die Tags übergeben, die notwendig sind, um ein Window-Object zu erstellen. Dabei können die Tags als variable Parameter oder auch als Pointer auf eine Liste übergeben werden.

Alternativ kann auch der Konstruktor ohne Parameter benutzt werden. In diesem Fall muß die Methode \*Create()\* (See MUINotify.) benutzt werden, um das Window-Object zu erstellen. Dieser Methode können auch variable Taglisten oder ein Pointer auf eine Tagliste übergeben werden.

```

*Beispiel:*
void main()
{
    MUIWindow win(
        MUIA_Window_Title, "Window-Titel",
        .
        .
        .
        TAG_DONE);
    .
    .
};

```

## 1.106 TWiMUI.guide/inline's

inline's  
 =====

Hier sind einige Funktionen aufgeführt, die per inline definiert wurden. Es handelt sich dabei um einige der Aufrufe der MUI\_MakeObject()-Funktion. Diese Aufrufe wurden hier neu definiert, damit eine effektive Typprüfung stattfinden kann. Es muß beachtet werden, daß mit diesen Funktionen erstellte Objekte nicht automatisch wieder destruiert werden, da es keinen Destruktor dafür gibt. Sie sollten deshalb nur in einer Parameter-Liste bei der Erstellung eines anderen Objektes aufgerufen werden. Diese Funktionen sind deshalb auch nur für Objekte definiert, die normalerweise keine Methoden benutzen oder keine weiteren Attribute haben.

Für die genaue Bedeutung dieser Objekte lesen Sie bitte die MUI-Dokumentation.

```

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)

```

Zusätzlich zu diesen Funktionen gibt es noch eine Funktion. die es ermöglicht. relativ einfach ein Id (z.B. MUIA\_Window\_ID) zu erstellen.

```

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

```

## 1.107 TWiMUI.guide/Danksagungen

Danksagungen  
\*\*\*\*\*

Als erstes muß ich natürlich meiner Frau danken. die mich teilweise gar nicht mehr gesehen hat.

Als nächstem den Entwicklern von StormC. die es ermöglichen. ein C++-Programm mit allen Sprachelementen auf dem AMIGA zu erstellen.

Und natürlich allen anderen. die mit Kritik und Anregungen zum Erstellen der Bibliothek beigetragen haben.

## 1.108 TWiMUI.guide/Author

Author  
\*\*\*\*\*

Für Anregungen, konstruktive Kritik und eventuelle Fragen bin ich auf folgenden Wegen zu erreichen:

Brief-Post  
 Thomas Wilhelmi  
 Taunusstraße 14  
 D - 61138 Niederdorfelden  
 Fax  
 06101/531061

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

Bei Fragen, Kritik oder Anregungen braucht niemand Hemmungen zu haben und kann sich gerne jederzeit an mich wenden. Bevorzugt behandelt werden natürlich E-Mail-Anfragen.

Ich bin auch Teilnehmer an der MUI Mailingliste.

---

## 1.109 TWiMUI.guide/MUI

MUI

\*\*\*

MUI ist ein System zum einfacheren Programmieren und sehr komfortablen Konfigurieren einer GUI.

MUI ist Shareware. Eine unregistrierte Version laesst sich vom AmiNet oder auch aus diversen Mailboxen beziehen.

MUI ist (C) 1993/94 Stefan Stuntz.

Bei ihm kann man sich auch mit der Übersendung von 30,-- DM als Benutzer registrieren lassen:

Stefan Stuntz

Eduard-Spranger-Straße 7

D-80935 München

## 1.110 TWiMUI.guide/Registrierung

Registrierung

\*\*\*\*\*

Bitte füllen Sie das beiliegende Formular aus und senden Sie es per E-Mail, Fax oder Brief an mich (See Author.).

Wenn sich jemand bereit erklären würde, diese Dokumentation zu übersetzen, braucht derjenige natürlich keine Registrierungs-Gebühr zu bezahlen. Aber ich bitte um vorherige Kontaktaufnahme.

Registrierung

Hiermit möchte ich mich für TWiMUI registrieren lassen.

Vorname: \_\_\_\_\_

Name: \_\_\_\_\_

Straße: \_\_\_\_\_

PLZ: \_\_\_\_\_ Ort: \_\_\_\_\_

Telefon: \_\_\_\_\_

---

E-Mail: \_\_\_\_\_

Gewünschte Zahlungsweise bitte ankreuzen:

- \* per Nachnahme (DM 40,-, nicht ins Ausland)
- \* per beiliegendem Vorkasse-Scheck mit Brief-Versand einer Diskette (DM 30,- in Deutschland, DM 35,- in Europa, DM 40,- Rest der Welt)
- \* per Überweisung mit Brief-Versand einer Diskette (DM 30,- in Deutschland, DM 35,- in Europa, DM 40,- Rest der Welt)
- \* per beiliegendem Vorkasse-Scheck mit E-Mail-Versand (DM 30,-)
- \* per Überweisung mit E-Mail-Versand (DM 30,-)
- \* Diese Dokumentation in gedruckter Form beim Briefversand beilegen (zzgl. DM 10,-)

Meine Bankverbindung:  
Postgiroamt Ffm  
Kontonummer 3973 35-603  
BLZ 500 100 60

## 1.111 TWiMUI.guide/Einschränkung

Einschränkung  
\*\*\*\*\*

In der unregistrierten Version befindet sich eine Einschränkung im Vergleich zur registrierten Vollversion. Und zwar ist es nur möglich, 40 Objekte in einem Programm zu kreieren. Wenn versucht wird, mehr Objekte zu kreieren, so wird die Exception MUIT mit dem Typ 100 geworfen.

Ansonsten gibt es keine weiteren Einschränkungen.

## 1.112 TWiMUI.guide/Haftungsausschluß

Haftungsausschluß  
\*\*\*\*\*

Durch die Verwendung dieses Programms akzeptiert der Benutzer die volle Verantwortung für alle Schäden, die durch seine Benutzung oder das Unvermögen seiner Benutzung auftreten können. Der Entwickler dieser Software kann nicht verantwortlich gemacht werden.

---