The OSF/Motif based X Windows Object Library (XOL)

Gunar Schorcht
Technical University of Ilmenau
Department of Computer Science and Automation
Gunar@TheoInf.TU-Ilmenau.DE

Version 0.2

Technical Report SST-02/96

April 29, 1996

Contents

List of Figures

1 Introduction

Graphical user interfaces of applications that run under *UNIX/X* Windows are normally programmed by using the *X Toolkit Intrinsics (Xt)* [Nye92a] and the *OSF/Motif* Toolkit [Ope93] [HF94]. These toolkits provide a set of more or less sophisticated widgets. Widgets are objects that handle a window and hold additional information about this window. They have procedures, so-called methods, to handle these data and its window. The application uses the methods and data elements of the widgets, rather than it uses the widget's windows directly. Both toolkits use the basic *X Windows* library (*Xlib*) [Nye92b]. The *Xt* as well as the *OSF/Motif* toolkit have an application programming interface (API) written in the *C* programming language. Though, their basic structure can be considered as object-oriented. However, to use them within an object-oriented programming language like *C++*, they have to be encapsulated within a class library. For this reason there are a number of class libraries that encapsulate the *OSF/Motif* toolkit. The most popular one is the *Motif++* library [Unk]. It provides a *C++* interface and encapsulates each *OSF/Motif* widget class within one *C++* class. The functions that are applicable to a *OSF/Motif* widget class are realized by member functions. This approach has the advantage that the hierarchy of the *OSF/Motif* widget set is not broken. Thus the library is very easy to use for programmers that are familiar with the *OSF/Motif* widget set. However, this approach realizes only an one by one mapping of the application programming interface written in the *C* to a *C++* programming interface without any extension by functionality. It does not provide a framework for graphical user interface programming. Therefore, the gain by using this library is very small. A fully different approach is used by the class library *GINA* [BGS92]. It provides an own complex class hierarchy. It uses the *OSF/Motif* widget set to realize a framework class hierarchy for graphical user interface programming. Although this library is high sophisticated and a lot of work was spend to develop it, the library is often too complex for daily work. For example, each callback function has to be realized as a special callback class respective a callback object before it can be used to handle callback events. This lead to a lot of programming overhead because the programmer has to encapsulate also the simplest things within classes.
What the programmer needs is a *C++* class library

- that encapsulates the application programming interface of the *OSF/Motif* toolkit written in *C*,
- that is small enough so that the effort for learning something about the library is small,
- that is intuitive so that the programming with the class library is very easy,
- that is as flexible as possible so that the programmer can use any feature of the underlying *OSF/Motif* toolkit, and
- that offers classes which serve as a framework for easy user interface programming.

The meet of all of these features is a heavy task. Therefore, we have developed a very small library that realizes a framework for graphical user interface programming based on the *OSF/Motif* widget set. The class library has only few but as flexible as possible classes. The library is described in section 3.1.

2 OSF/Motif Class Hierarchy

The *OSF/Motif* widget class hierarchy, Fig. 1 [Fer93] can be divided in three mainly groups of widgets [Ope93]

- primitives and gadgets,
- managers, and
- shells.

Primitives are widgets that have no children. They are the controls, that is, the fundamental units of input and output of an user interface over that the user interacts with the application directly. Primitive widgets are usually building blocks for composite widgets. Within a widget tree, the widgets instanced from primitive subclasses are the leafs. The group of *OSF/Motif* primitives classes comprises labels, separators, buttons, scroll bars, lists, and text widgets. Some primitive widget classes have equivalent objects called gadgets. These are just like primitives except that, to enhance the performance, they have no associated windows. Gadgets use the window of their parent manager widget to process the input and output.

Manager widgets are composite widgets that contain one or more primitive widgets, primitive gadgets, or other manager widgets. Manager widgets are responsible for the geometrical arrangement of their children. They also process and dispatch the input of their gadget children. The group of *OSF/Motif* managers classes

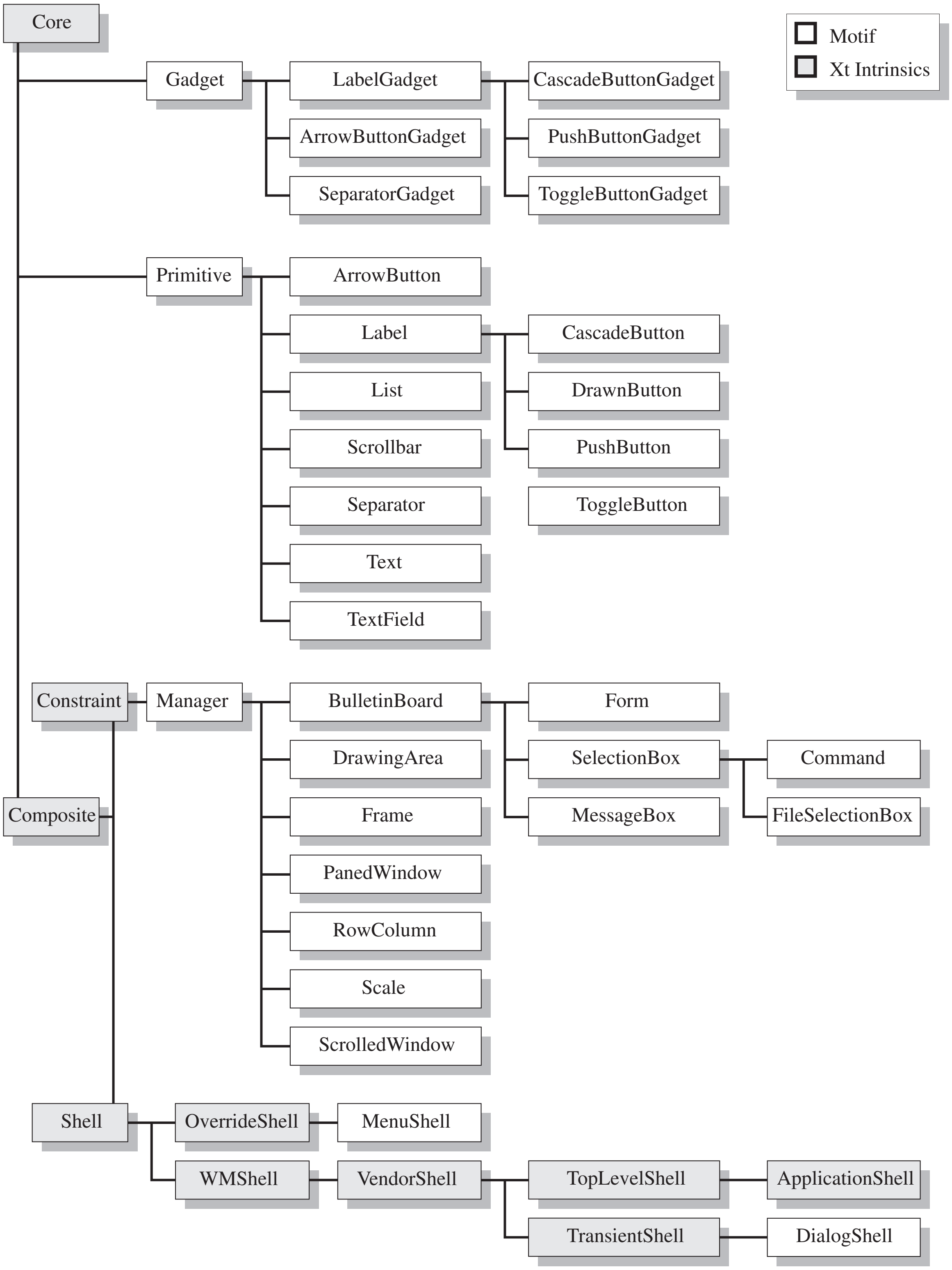

Figure 1: Motif widget hierarchy

comprises frames, scrolled and paned windows, menus, constraint-based geometry managers, and several kinds of dialogs.

Shells are widgets whose main purpose is to communicate with the window manager. Normally, shells have only one child, and they have the same size and position as the child. Specialised *OSF/Motif* shells exist to envelop application, dialogs, and menus. Each widget with a top-level window , that is, a window whose parent is the root window of the screen, needs to be enclosed in a shell. The application shell is the root within the widget tree.

For more information about the *OSF/Motif* widget classes see [Ope93].

3 The XOL Class Library

3.1 Class Hierarchy

The *XOL* class library can be divided in four different groups of classes

- special widget classes,
- common widget classes,
- framework classes, and
- miscellaneous classes.

Common widget classes encapsulate the most common *OSF/Motif* . Common widget classes are the root class of the *OSF/Motif* widget class hierarchy as well the *OSF/Motif* widget classes on the second and third hierarchy level, respectively. Thus common *OSF/Motif* widget classes are for example the classes *Core*, *Composite*, *Constraint*, *Shell*, *Gagdet*, and *Primitive*, Fig. 1. Normally, widgets are not instanced directly from these common *OSF/Motif* widget classes to develop the user interface, because they provide only the most common functionality without any special features. However, the most common *OSF/Motif* widget classes are encapsulated by the common *XOL* widget classes in a way, so that a widget object instanced from one of these common *XOL* widget object classes can realize any widget of a derived *OSF/Motif* widget class. Thus, the common *XOL* widget object classes provide a mechanism to encapsulate all different *OSF/Motif* widget classes by a simple common C++ interface. That is, the common *XOL* widget object classes are general enough to handle widgets of any derived *OSF/Motif* widget class. This has the advantage that the widget class hierarchy is not broken. The common widget object class are used to realize specialised widget object class.

Special widget object classes encapsulates simply specialised *OSF/Motif* widget classes. Specialised *OSF/Motif* widget classes are leaf classes or derived classes deep in the *OSF/Motif* widget class hierarchy, that provide a specialised functionality to realize user interface elements. For convenience, a number of specialised *OSF/Motif* classes are encapsulated by specialised *XOL* widget object classes.

Framework classes are classes that do not encapsulate *OSF/Motif* widgets directly. Rather, they comprise the functionality of several *OSF/Motif* widgets as well use the *OSF/Motif* functionality to provide the user of the *XOL* with more complex user interface classes in a convenient way.

Miscellaneous classes are those classes that do not encapsulate *OSF/Motif* widgets and are not framework classes, but are simple classes that are needed for user interface programming.

3.2 Callback Method Mechanism

An application under *X Windows* is event driven. There is a infinite loop that waits for events, e.g., user inputs. It reacts to occurring events by calling action routines, the so-called *callback functions* . A number of callback functions can be registered for each event. Thus the application consists of functions that react to occurring events. In object-oriented domain, functions that are associated with objects are called methods. In this document the methods that serve as callback functions are called *callback methods* . Since, the callback methods are used as ordinary *C* language function, they have to be declared as static. Therefore, within a

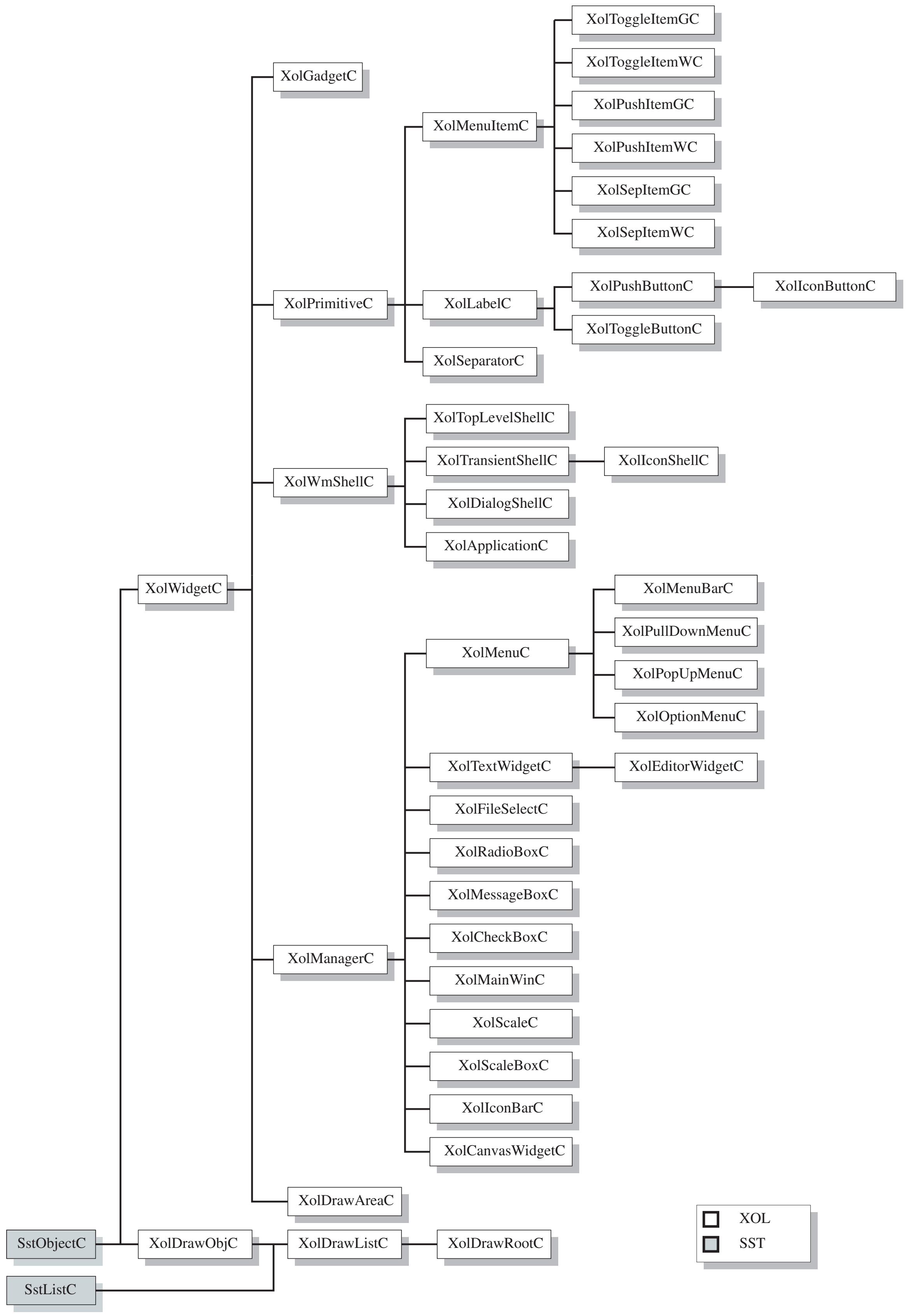

Figure 2: *XOL* class hierarchy

callback method the *this* pointer is not available. A callback method has three parameters that are delivered from the *X Windows* system:

- The first parameter specifies the widget that produced the event.
- The second parameter contains the callback user data as registered during the callback method registration.
- The third parameter contains the *callback data structure* in which specific information about the event are delivered.

The *XOL* class library realizes two different callback method using schemes

- the *callback handler scheme* and
- the *callback method scheme.*

Callback Handler Scheme: In this scheme a special callback method, the callback handler, reacts to all callback events. All Widget object classes that belong to the *XOL* and have to react to callback events register for each possible callback event a special callback method, the so-called *callback handler*. Furthermore, these classes register the *this* pointer as callback user data. Thus the callback handler can determine the widget object that is invoked for the callback event processing. Based on the information delivered by the callback data structure the callback handler choices a special method of the invoked object that is called for further processing of the callback event. The most of these methods are declared virtual dummy methods without any real functionality. They have to be overwritten within derived classes that processes the callback events.. The callback handler itself preprocesses in most cases the callback data structure so that it can call the special method for further event processing with specific parameters.

Callback Method Scheme: In this scheme the application declares its own static methods as callback methods. The declaration of these methods has to correspond to the pre-described callback method prototype. The callback methods are registered as well unregistered by using callback method management methods of the *XOL* widget objects. As callback user data the *this* pointer or any other information can be registered. Thus the *XOL* provides a very flexible mechanism to realize callback methods.

3.3 Naming Mechanism

Within the *XOL* library each widget object is a part of a widget object tree. Each widget object in this widget object tree has a name. There are two types of names, the short name and the full name. The short name of a widget object is the name that is specified during the widget object construction. The full name is the concatenation of all widget object short names on the path within the widget object tree beginning with root object name and ending with the short name of the widget object itself. The concatenated names are separated by dots.

3.4 Using Scheme

The using of *XOL* widget objects is divided in two mainly tasks

1. the construction (creating the object itself), and
2. the creation(creating the underlying *OSF/Motif* widgets)

Therefore in the following the words construction and creating are different in its meaning and have to be handled carefully by the reader.

4 Naming Conventions

The following rules are valid for the naming of identifiers used in the class library.

- All class identifiers have the prefix *XOL*. Furthermore, they have the postfix *C* that is the abbreviation for *class*. The identifier *XolWidgetC* is an example for a class identifier.

- Data members of a class have the prefix *m* that stands for *member*, e.g, the data member *XolWidgetC::mChilds*. Exceptions are public data members. In case of public data member the identifier have not to have the a special prefix. For more convenience, it is sometimes reasonable to have very short data member names.

- Formal parameter identifier of member functions have the prefix *p* in most cases that stands for *parameter*. An example for an formal parameter identifier is *XolWidgetC::Create(Widget pWidget)*.
- All type identifiers consist of the prefix *XOL* , the identifier body, and the postfix *T* that stands for *type*, e.g., *XolPointerT*.
- Global variables carry *s* as prefix that stands for *static*, e.g., *sDebug*.
- Identifiers that consist of more than one character are starting with uppercase letters. Separations of identifier parts are realized by uppercase letters within he identifier, e.g., *sBottomShadowColor*, that stands for bottom shadow colour.
- Underscore characters are not allowed within an identifier with the exception of constant identifiers.
- Constant identifiers consist of uppercase letters. Separations within a constant identifier are realized by underscores, e.g., *XOL_BROWSE_SIZE*.

5 Defined Constants

The following constants are defined globally for using it in the whole library.

5.1 Constant Declaration

```
#include <XOL/Defines.h>

#define XOL_DEF_XPOS      10
#define XOL_DEF_YPOS      10
#define XOL_DEF_XSIZE     800
#define XOL_DEF_YSIZE     600

// ---------------------------------------------------------------------------

#include <XOL/MsgBox.h>

#define XOL_ERROR       XmDIALOG_ERROR
#define XOL_INFO        XmDIALOG_INFORMATION
#define XOL_MESSAGE     XmDIALOG_MESSAGE
#define XOL_QUESTION    XmDIALOG_QUESTION
#define XOL_WARNING     XmDIALOG_WARNING
#define XOL_WORKING     XmDIALOG_WORKING

// ---------------------------------------------------------------------------

#include <XOL/Scale.h>

#define XOL_MOD_PREC    1
#define XOL_MOD_MINMAX  2
#define XOL_MOD_BOTH    3
```

5.2 Constant Description

XOL_DEF_XPOS
XOL_DEF_YPOS
XOL_DEF_XSIZE
XOL_DEF_YSIZE

Description: These constants define the default position and size values for created widget objects.

XOL_ERROR
XOL_INFO
XOL_MESSAGE
XOL_QUESTION
XOL_WARNING
XOL_WORKING

Description: These constants are only aliases for *OSF/Motif* constants

XOL_MOD_PREC
XOL_MOD_MINMAX
XOL_MOD_BOTH

Description: These constants define the possible values of the changing mode used to determine the strategy for value changing within a scale box widget object.

6 Library Types

The following types are declared globally for using them in the whole library. Most types are only aliases of underlying *Motif/Xt* types. Other types are aliases for abbreviations of very long type declaration.

6.1 Type Declarations

```
#include <XOL/Defines.h>

typedef unsigned int   uint;
typedef unsigned char  byte;
typedef unsigned short word;
typedef unsigned long  dword;
typedef unsigned char  bool;

// ---------------------------------------------------------------------------

#include <XOL/Widget.h>

typedef XtPointer             XolPointerT;
typedef SstListC<XolWidgetC>  XolWidgetListT;
typedef void                  (*XolCallbackProcT)(Widget,XolWidgetC*
                                                 ,XolPointerT);
// ---------------------------------------------------------------------------

#include <XOL/Menu.h>

typedef struct _XolMenuItemTag {
    char*             Label;        // the label/name of the menu item
    WidgetClass       Class;        // can be XOL_PBG, XOL_PBW, ...
    char              Mnemonic;     // mnemonic character, 0 for none
    char*             Accel;        // accelerator key, NULL for none
    char*             AccelText;    // text for accelerator key
    XolCallbackProcT  Callback;     // routine to call on select, NULL if  none
    XolPointerT       ItemData;     // additional item data
} XolMenuItemT;

typedef enum { XOL_MB = XmMENU_BAR,
               XOL_PD = XmMENU_PULLDOWN,
               XOL_PU = XmMENU_POPUP,
               XOL_OM = XmMENU_OPTION
```

```
} XolMenuT;
```

6.2 Important Types

typedef XtPointer XolPointerT;

Description: This type is an alias for the *Motif/Xt* type *XtPointer*. This alias is defined for a homogeneous naming within the library. At each location where the type *XolPointerT* is required, the type *XtPointer* can also be used.

typedef SstListC<XolWidgetC> XolWidgetListT;

Description: This type defines a *XOL* widget object type by using the generic class *SstListC* [Sch96] and the *XOL* widget class as the template parameter. The *XOL* widget object list type is used for the definition of the *XOL* widget object hierarchy.

typedef void (*XolCallbackProcT) (Widget, XolWidgetC*, XolPointerT);

Description: This type is the callback function prototype declaration. All methods that are used as callback method of a *XOL* widget object have to be of this type. Furthermore, they have to be declared as static. However, it is possible to declare callback methods, that use as second parameter type not the class *XolWidgetC* *, but a pointer to any derived class. In this case, the method has to be casted to the type *XolCallbackProcT* for using it as a parameter of a callback method management method, e.g., the method *XolWidgetC::AddCallback*.

typedef struct _XolMenuItemTag XolMenuItemT;

Description: This type defines a menu item specification structure that can be used for the construction of a menu widget object from a menu item array, see section 8.5. The last entry of the menu item array have to be empty, i.e., all elements of the menu item specification structure are *NULL*. The meaning of the elements of this type is the same as the meaning of parameters of the default constructor of the class *XolMenuItemC*, see section 8.10.

7 Library Global Variables

7.1 Global Variable Declarations

```
#include <XOL/Widget.h>

extern Pixel sBackGroundColor,   // For internals only !!!
             sForeGroundColor,
             sTopShadowColor,
             sBottomShadowColor,
             sSelectColor;

extern bool  sDebug;      // TRUE -> debugging output will be generated
extern FILE* sDebugOut;   // Target for debugging outputs
extern bool  sFullName;   // TRUE -> widgets are identified by full name

// ----------------------------------------------------------------------------

#include <XOL/App.h>

extern String sFallbacks[];
//     "*background: grey" ,
//     "*fontList: -*-lucida-bold-r-*-*-11-*-*-*-*-*-*-* = XOL_DEFAULT_FONT"
```

```
extern XolApplicationC *sGlobApp; // global defined application object for
                                  // global access, the variable should be used
                                  // by the application for instantiation of
                                  // the only one needed application object
```

7.2 Important Global Variables

extern Pixel sBackGroundColor;
extern Pixel sForeGroundColor;
extern Pixel sTopShadowColor;
extern Pixel sBottomShadowColor;
extern Pixel sSelectColor;

Description: These global variables are used internally by *XOL* widget object class *XolWidgetC* and MUST NOT BE SET directly by user.

Values: Not specified

extern bool sDebug;

Description: This variable determines whether an output is generated by calling the method *XolWidgetC::dprintf*. By default this variable is set to *FALSE*.

Values: *TRUE* - debugging output is generated
FALSE - debugging output is not generated (default)

extern FILE* sDebugOut;

Description: This variable determines the destination for the debugging output generated by the method *XolWidgetC::dprintf* when the variable *sDebug* is switched on. By default this variable is set to the *stdout* device.

Values: File descriptor pointer

extern bool sFullName;

Description: This variable determines whether *XOL* widget objects are identified by their full hierarchical name or by their short name in the debugging output. By default this variable is set to *TRUE*.

Values: *TRUE* - full hierarchical names
FALSE - short names (default)

String sFallbacks[];

Description: This variable is a table of standard fallback resources. Standard fallback resources are used to specify the value of *X Windows* and *OSF/Motif* resources when neither an application specific nor an user specific resource definition exists. Within the comment the standard values of the fallback resources table is given. It is possible to add fallback resources to the table by using the method *XolApplicationC::AddFallbackRes*, see section 8.31.

Values: Fallback resource strings

XolApplicationC *sGlobApp;

Description: The application should use this global variable for instantiation of the only one allowed application object. This variable enables the global access to the application object.

Values: Pointer to the application object

8 Library Classes

8.1 Class XolWidgetC

The root class of the whole *OSF/Motif* widget hierarchy is the *Xt* widget class *Core*. This class defines all basic *OSF/Motif* widget properties. It provides a number of attributes and methods to handle with these attributes. The class *XolWidgetC* encapsulates these common properties and provides several methods for handling with it. The class *XolWidgetC* is derived from the object class *SstObjectC* [Sch96] so that each *XOL* widget object instanced from this or a derived class have the capabilities provided by the object class. Although this class realizes by default only the *Core* widget class it was designed so that it can also be used as an encapsulation of all other *OSF/Motif* widget classes without any restrictions. Thus the simple encapsulation of an arbitrary *OSF/Motif* widget class is possible by only one C++ class. This class serves as the root of the *XOL* widget class hierarchy. The class *XolWidgetC* has a number of data that are needed to handle with information about the *OSF/Motif* widgets as well the used *Xt* functionality. Furthermore, the class defines elements and methods that are needed to handle the widget object hierarchy. The using of objects is handled in two different phases. First the object is created by the constructor. The object get the name and is put into the child widget object list of its parent. After the construction the underlying *OSF/Motif* widget has to be created by using a special method. The class *XolWidgetC* realizes an callback method handler that handle the *XmDESTROY* event.

8.1.1 Class Declaration

```
#include <XOL/Widget.h>

class XolWidgetC : public SstObjectC
{
  protected:

    static void CallbackHandler(Widget, XolWidgetC*, XolPointerT);

    XolWidgetListT mChilds;     // for holding of XOL Widget children, created
                                // with this XOL Widget as parent
                                // the child list is only used internally

  public:

    WidgetClass    mXtClass;    // holds the Xt or Motif class type
    Widget         mXtWidget;   // holds the Xt widget handle.
    Widget         mXtParent;   // holds the Xt typed parent widget
    Screen*        mScreen;     // holds the Screen
    Display*       mDisplay;    // holds the Display

    XolWidgetC*    mParent;     // holds the pointer to XOL Widget parent
                                // if the constructor is called with Xt typed
                                // parent, the element will be set to NULL

    // Constructors that create the widget control object.

    XolWidgetC (XolWidgetC *pParent, char *pName
                                  , WidgetClass pClass = widgetClass);
    XolWidgetC (Widget pParent, char *pName
                              , WidgetClass pClass = widgetClass);

    // Destructor

    virtual ~XolWidgetC();
```

```
// Methods that make the data members accessible

WidgetClass  XtClass () { return mXtClass;  }
Widget       XtWidget() { return mXtWidget; }
Widget       XtParent() { return mXtParent; }
XolWidgetC*  Parent  () { return mParent;   }


// Methods that get Xt/Motif widget information.

Display* GetDisplay() { return XtDisplay(mXtWidget); }
Screen*  GetScreen () { return XtScreen (mXtWidget); }
Window   GetWindow () { return XtWindow (mXtWidget); }


// Methods that that manage the real Xt/Motif widget.

virtual int  Create   (Widget pXtWidget);
virtual int  CreateVa (bool   pManaged = TRUE, ...);
virtual int  Create   (bool   pManaged = TRUE, ArgList pArgV = NULL
                                             , int     pArgC = 0);
virtual int  Destroy();

void AddChild(XolWidgetC *pChild);
void DelChild(XolWidgetC *pChild);

virtual void Manage  () { XtManageChild  (mXtWidget); }
virtual void UnManage() { XtUnmanageChild(mXtWidget); }


// Methods that managing child widget objects

XolWidgetC *GetChild(char*  pName, char pSep = '*');    // by its name
XolWidgetC *GetChild(Widget pWidget);                   // by its Xt widget

// Method that determines the name of the widget/widget control object.

virtual char *GetFullName();
virtual char *GetProject () { return "XOL"; }
virtual char *GetClass   () { return "XolWidgetC"; }
virtual char *GetObject  ();
virtual void Browse      (char *pRoot = NULL, word pMode = 0
                                            , FILE *pFile = stdout);


// Methods that register a callback function for a special event.
// If no callback data are given, the pointer to this widget object is
// registered by method as user data for call back.
// The callback function have to be specified as function with of type
// 'XolCallbackProcT' and have to be declared as static member function.

virtual void AddCallback(const char *pEvent, XolCallbackProcT pProc);
virtual void AddCallback(const char *pEvent, XolCallbackProcT pProc
                                           , XolWidgetC*      pData);
virtual void AddCallback(const char *pEvent, XolCallbackProcT pProc
                                           , XolPointerT      pData);
```

```
    // Methods that handles with widget attributes.

    int FillArgList (Arg *pArgV, int &pArgC, char *pRes, XtArgVal pValue, ...);

    void SetXmStr   (char *pResour, char *pValue
                                  , char *pCharSet = XmSTRING_DEFAULT_CHARSET);
    void SetOne     (char *pResour, XtArgVal  pValue);
    void GetOne     (char *pResour, XtArgVal* pValue);
    void Set        (char *pResour, XtArgVal  pValue, ...);
    void Get        (char *pResour, XtArgVal* pValue, ...);

    void SetPos     (long   x, long   y);
    void GetPos     (long  &x, long  &y);
    void SetSize    (long  dx, long  dy);
    void GetSize    (long &dx, long &dy);
    void SetColors  (char *pBackColor);
    void SetSense   (bool pSwitch = TRUE);

    void SetColorsToDef();


    // Methods that handling debugging output.

    void dprintf  (char *pFormat, ... /* va_dcl va_alist */); // -> DebugOut
    void dfprintf (FILE *pFile, char *pFormat, ... /* va_dcl va_alist */);
};
```

8.1.2 Important Data Members

XolWidgetListT mChilds;

Description: This data element is the list of the child *XOL* widget objects that have this widget object as parent. The *XOL* widget object class is using the class *SstListC* of the *SST common object library* [Sch96] to realize the child widget object list. With this data member a hierarchy of *XOL* widget objects can be built up and handled.

Values: Widget object list

WidgetClass mXtClass;

Description: This data element holds the *OSF/Motif* widget class of the *XOL* widget object.

Values: Widget class value

Widget mXtWidget;

Description: After the creation of the underlying *OSF/Motif* widget, this data member holds the *Xt* widget handle of the *XOL* widget object. If the underlying widget is not created this data member is set to *NULL*.

Values: Widget handle

Widget mXtParent;

Description: After the creation of the underlying *OSF/Motif* widget, this data member holds the handle of its parent *Xt* widget. If the underlying widget is not created this data member is set to *NULL*.

Values: Widget handle

Screen* mScreen;

Description: After the creation of the underlying *OSF/Motif* widget, this data member holds the *X Windows* screen of the *XOL* widget object. If the underlying widget is not created this data member is set to *NULL*.

Values: Pointer to the Screen Descriptor

Display* mDisplay;

Description: After the creation of the underlying *OSF/Motif* widget, this data member holds the *X Windows* display of the *XOL* widget object. If the underlying widget is not created this data member is set to *NULL*.

Values: Pointer to the Display Descriptor

XolWidgetC* mParent;

Description: This data member holds the pointer to the parent *XOL* widget object. If the constructor of this object is called with an *Xt* widget handle as the parent, this data member is set to *NULL*.

Values: Pointer to the parent widget object

8.1.3 Usable Methods and Operators

XolWidgetC (XolWidgetC *pParent, char *pName
, WidgetClass pClass = widgetClass);
XolWidgetC (Widget pParent, char *pName
, WidgetClass pClass = widgetClass);

Description: With each of this constructors a *XOL* widget object is created. However, the underlying *OSF/Motif* widget is not created by these constructors. This has to be done with specialised methods. These constructors create only the *XOL* widget object, initialise the data members, give it the name and register it as a child widget object by its parent. A *XOL* widget object can also be created with a *Xt* widget handle as its parent. in this case the *XOL* widget object cannot be registered as a child of its parent. The data member *mParent* is set to *NULL* .

Parameters:
pParent - Pointer to the parent widget object or the parent *Xt* widget handle
pName - Name of the widget object
pClass - *OSF/Motif* class of the widget object (by default the *OSF/Motif* class *Core* is used)

Results: None

virtual ~XolWidgetC ();

Description: The destructor destroys all its child *XOL* widget objects and unregister itself by its parent *XOL* widget object. Furthermore, it destroys the underlying *OSF/Motif* widget if it exists. The destructor is automatically called by the callback handler on the occurrence of the *XmDESTROY* event.

Parameters: None

Results: None

WidgetClass XtClass ();
Widget XtWidget ();
Widget XtParent ();
XolWidgetC* Parent ();

Description: These methods deliver the contents of several data members of the *XOL* widget object.

Parameters: None

Results: - *OSF/Motif* widget class

- *Xt* widget handle of the underlying *OSF/Motif* widget (*NULL* - the widget is not created yet)
- *Xt* widget handle of the parent *XOL* widget object (*NULL* - the widget is not created yet)
- parent *XOL* widget object (*NULL* - the widget object has no parent *XOL* widget object)

Display* GetDisplay ();
Screen* GetScreen ();
Window GetWindow();

Description: These methods deliver information about the used *OSF/Motif* widget from the underlying *X Window* system. For this purpose they use functions of the *Xt*.

Parameters: None

Results:
- Pointer to the display descriptor of the widget object
- Pointer to the screen descriptor of the widget object
- Windows handle of the widget object

virtual int CreateVa (bool pManaged = TRUE, ...);
virtual int Create (bool pManaged = TRUE, ArgList pArgV = NULL, int pArgC = 0);

Description: These methods create an underlying *OSF/Motif* widget for the *XOL* widget object and store it in the data member *mXtWidget*. The created *OSF/Motif* widget can be created as managed or unmanaged widget. Except for shell widgets, all widgets have to be managed by their parent widgets. The created *OSF/Motif* widget can be managed automatically after their creation. This is determined by the parameter *pManaged*. By default created widgets are managed automatically. Nothing more is done by these methods. If *this* object has already an underlying *OSF/Motif* widget, the method is broken and no new *OSF/Motif* widget is created. It is possible to define parameters by name/value pairs. The pairs of parameters can be defined as a sequence separated by commas or as a vector of such pairs. For the former kind of parameter definition the method *CreateVa* has to be used. The parameter sequence has to end with *NULL*. If the parameters are defined by an vector the number of name/value pairs has to be defined also as a parameter. The method *Create* can also be used without any parameter.

Parameters:
- *pManaged* - Determines whether the created widget should be managed automatically by its parent during the creation
  - *TRUE* - the widget is managed automatically (default)
  - *FALSE* - the widget is not managed automatically
- *pArgV* - Vector with widget parameters given by their name / value pairs
- *pArgC* - Number of parameter definitions within the parameter vectors *pArgV*
- ... - Widget parameters given as a sequence of name / value pairs and terminated with *NULL*

Results: None

virtual int Create (Widget pXtWidget);

Description: This method creates no new *OSF/Motif* widget. Rather, it builds up an association between the already existing *OSF/Motif* widget *pXtWidget* with the *XOL* widget object.

Parameters: *pXtWidget*- the already existing *OSF/Motif* widget

Results: None

virtual int Destroy ();

Description: This method destroys the *XOL* widget object by calling its destructor, see the method *XolWidgetC:: XolWidgetC*.

Parameters: None

Results: None

virtual void Manage ();
virtual void UnManage();

Description: These methods manage respective unmanage the underlying *OSF/Motif* widget of *this XOL* widget object by its parent widget. A widget becomes visible after the managing it by its parent or becomes invisible after unmanaging it by its parent. Thus this methods are useful to make a widget visible or invisible.

Parameters: None

Results: None

XolWidgetC *GetChild (char* pName, char pSep = '*');
XolWidgetC *GetChild (Widget pWidget);

Description: These methods get the pointer to the child *XOL* widget object. The object is specified by its Name *pName* or the handle of its underlying *OSF/Motif* widget. If the *XOL* widget object is specified by its Name, a separator character can be specified. With the separator the hierarchical name of a child deeper in the hierarchy can be specified, e.g., *child1*child2*child3* that is the name of the child3 of the child2 of the child1 of *this XOL* widget object.

Parameters: *pName* - Name of the specified object
pSep - Separator character for full hierarchical child name (by default *)
pWidget - *OSF/Motif* widget handle of the specified object

Results: Pointer to the *XOL* child widget object

void AddChild (XolWidgetC *pChild);
void DelChild (XolWidgetC *pChild);

Description: These methods registers/unregisters a *XOL* widget object as a child to the child widget object list *mChilds* of *this XOL* widget object. The child widget object is not destroyed by this method on deleting. *This method must not be called directly. Any widget object is registered automatically during its construction.*

Parameters: *pChild* - Pointer to the child object that is added/deleted from the child widget object list of *this XOL* widget object

Results: None

virtual char *GetFullName ();

Description: This method gets the full hierarchical name of *this XOL* widget object. The name is starting with dot. The partial names of the widget object on each hierarchy level are also separated also by dots.

Parameters: None

Results: Full hierarchical name of *this* widget object

virtual char *GetProject ();
virtual char *GetClass ();
virtual char *GetObject ();

Description: These methods overwrite the virtual methods of the base class *SstObjectC*. These method deliver the project, class and the *XOL* widget object identifiers. If the global variable *sFullName* is set the method *GetObject* delivers the *XOL* widget name in the long form using the method *GetFullName*. Otherwise the method *GetObject* delivers only the *XOL* widget name.

Parameters: None

Results:
- Project name (by default *XOL*)
- Class name (by default *XolWidgetC*)
- Object name in short or long form (see *sFullName*)

virtual void Browse (char *pRoot = NULL, word pMode = 0
, FILE *pFile = stdout);

Description: With this method the *XOL* widget object hierarchy with *this* widget object as the root can be browsed. Several information determined by the browse mode *pMode* can be print out. The method print out the wished information about *this XOL* widget object. After this it calls this method of all child objects stored in the child widget object list. The output is print out to the file given parameter *pFile*.

Parameters:
- *pRoot* - Root starting string (by default it is empty)
- *pMode* - Browser Mode, several browsing modes can be merged by adding constants
  - *XOL_BROWSE_ADDR* - print out the address of *this XOL* widget object and its parent *XOL* widget object
  - *XOL_BROWSE_POS* - print out the position of *this XOL* widget object
  - *XOL_BROWSE_SIZE* - print out the size of *this XOL* widget object
- *pFile* - File specifier for output (by default *stdout*)

Results: None

virtual void AddCallback (const char *pEvent, XolCallbackProcT pProc);
virtual void AddCallback (const char *pEvent, XolCallbackProcT pProc
, XolWidgetC* pData);
virtual void AddCallback (const char *pEvent, XolCallbackProcT pProc
, XolPointerT pData);

Description: These methods register a callback method for a special event given by the *pEvent*. Methods that serve as callback methods has to be declared as static. If no callback data are given, i.e. the first form of these methods is used, the pointer to *this XOL* widget object is registered by the method as callback user data. This has the purpose that the none existing *this* pointer in static member methods can be determined by means of the callback user data, see section 3.2. In the second form of these methods, the *XOL* widget object pointer passed as the callback user data to the callback method can be specified explicitly by the parameter *pData*. By using the third form any kind of information given by the parameter *pData* can be specified as callback user data. Thus a callback method that does not need the *this* pointer can register any information for passing it as callback user data to the callback method. If the information given by parameter *pData* is not of type *XolPointerT*, it has to be casted to this type. The callback method itself given by the parameter *pProc* have to be declared as a function of type *XolCallbackProcT*.

Parameters:
- *pEvent* - Callback event for which the function given by *pProc* is registered
- *pProc* - Callback method
- *pData* - Data that are passed as the callback user data to the callback method (if this parameter is not given the *this* pointer is registered as the callback user data parameter)

Results: None

int FillArgList (Arg *pArgV, int &pArgC, char *pRes, XtArgVal pValue, ...);

Description: The method deals with an *Motif/Xt* parameter array with a depth of *MaxCreatePar*. It converts a sequence of *Motif/Xt* parameters given by their name/value pairs to a vector of such pairs. The method checks the parameter array for enough space based on the number of already contained parameter pairs given by the counter *pArgC*. The counter is incremented by the number of stored parameter pairs. The *Motif/Xt* parameter sequence has to be terminated by *NULL*.

Parameters:
- *pArgV* - Pointer to the first parameter pair
- *pArgC* - Number of already contained before and after the execution of this method
- *pRes* - Name of the first *Motif/Xt* parameter pair
- *pValue* - Value of the first *Motif/Xt* parameter pair
- *...* - Sequence of name/value pairs terminated by *NULL*

Results: Number of handled parameter pairs

void SetXmStr (char *pResour, char *pValue
, char *pCharSet = XmSTRING_DEFAULT_CHARSET);

Description: This method set the resource given by its name *pResour* to the string given by *pValue* based on the character set given by *pCharSet*. This method encapsulate the more complex *OSF/Motif* string creation methodology.

Parameters:
pResour - Resource name
pValue - String
pCharSet - Character set

Results: None

void SetOne (char *pResour, XtArgVal pValue);
void GetOne (char *pResour, XtArgVal* pValue);
void Set (char *pResour, XtArgVal pValue, ...);
void Get (char *pResour, XtArgVal* pValue, ...);

Description: These method set/get one or more *Motif/Xt* parameter. If more than one parameter is set/get, a sequence of name/value terminated by *NULL* has to be given as the parameter. Furthermore, if the get methods are used the parameters are given as pointers.

Parameters:
pResour - Name (Pointer to Name) of the first *Motif/Xt* parameter
pValue - Value (Pointer to Value) of the first *Motif/Xt* parameter
... - Sequence of name/value (pointer to name/pointer to value) with *NULL* -termination

Results: None

void SetPos (long x, long y);
void GetPos (long &x, long &y);
void SetSize (long dx, long dy);
void GetSize (long &dx, long &dy);

Description: These methods get/set the position/size of an *this XOL* widget object.

Parameters:
x - x position
y - y position
dx - x size
dy - y size

Results: None

void SetColors (char *pBackColor);

Description: With this method the colours of *this XOL* widget object are set. The used colour set is calculated by means of an *OSF/Motif* library function based on the background colour given by *pBackColor*. See the *OSF/Motif* manual for more information.

Parameters: *pBackColor*- Background colour name

Results: None

void SetSense (bool pSwitch = TRUE);

Description: With this method the sensitivity for inputs of *this XOL* widget object is switched on or off. This means *this XOL* widget object is set to the active of inactive state.

Parameters: *pSwitch* - New sensitivity state
TRUE - *this XOL* widget object is sensitive for inputs
FALSE - *this XOL* widget object is insensitive for inputs

Results: None

void SetColorsToDef();

Description: With this method the colours of *this XOL* widget object are set to colours stored in the global static variables *sBackGroundColor, sForeGroundColor,..., sSelectColor.*

Parameters: None

Results: None

void dprintf (char *pFormat, ...);
void dfprintf (FILE *pFile, char *pFormat, ...);

Description: With this method the values given by the variable parameter list are print out as debugging output to the stdout device or a file. The format of the output is given by the format string *pFormat*. The debugging output contains also the short name or the long name of *this XOL* widget object.

Parameters: *pFile* - Debugging output file (not given -¿ the output is written to the stdout device)
pFormat - Format string

Results: None

8.2 Class XolManagerC

The class *XolManagerC* encapsulates the *OSF/Motif* widget class *Manager* within an *XOL* widget object class. Manager widgets are composite widgets that contain primitive widgets, gadgets, or other manager widgets. Manager widgets are responsible for the geometrical arrangement of their children. They also process and dispatch the input of their gadget children. Specialised *OSF/Motif* managers include frames, scrolled and paned windows, menus, constraint-based geometry managers, and several kinds of dialogs.
The class is derived from the general *XOL* widget object class *XolWidgetC*. The *OSF/Motif Manager* widget class is the basis class for all widgets that require geometry management, i.e., all more complex widgets. They manage *OSF/Motif* controls that are instanced from *OSF/Motif* primitive subclasses, see section 8.3. This class registers a callback handler for the *OSF/Motif* callback reason *XmCR_HELP* automatically. The callback handler executes the method *HelpEvent* that is a virtual dummy method. The method *HelpEvent* can be overwritten with real functionality by derived widget object classes. Thus, help functionality can be realized for all widget objects that are derived from the manager widget object class.

8.2.1 Class Declaration

```
#include <XOL/Widget.h>

class XolManagerC : public XolWidgetC
{
   static void CallbackHandler(Widget, XolManagerC*, XolPointerT);

  public:

    XolManagerC (Widget pParent, char *pName,
                 WidgetClass pClass = xmManagerWidgetClass)
               : XolWidgetC(pParent, pName, pClass) {}

    XolManagerC (XolWidgetC *pParent, char *pName,
                 WidgetClass pClass = xmManagerWidgetClass)
               : XolWidgetC(pParent, pName, pClass) {}

    virtual int Create (bool pManaged = TRUE, ArgList pArgV = NULL
                                          , int pArgC = 0);

    virtual char *GetClass () { return "XolManagerC"; }
    virtual void HelpEvent () { return; }  // virtual dummy callback function
};
```

8.2.2 Important Data Members

This class defines no data members.

8.2.3 Usable Methods and Operators

XolManagerC (XolWidgetC *pParent, char *pName,
WidgetClass pClass = xmManagerWidgetClass);
XolManagerC (Widget pParent, char *pName,
WidgetClass pClass = xmManagerWidgetClass);

Description: The constructors execute only the constructors of the basis class. For further information see *XolWidgetC*.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle
pName - Name of the widget object
pClass - *OSF/Motif* class of the widget object (by default the *OSF/Motif* class *xmManagerWidgetClass* is used)

Results: None

virtual int Create (bool pManaged = TRUE, ArgList pArgV = NULL
, int pArgC = 0);

Description: This method overwrites the virtual method *XolWidgetC::CreateVa*. It executes the same method of the basis class. Furthermore, it registers the callback handler as the callback method for the *OSF/Motif* callback resource *XmCR_HELP*.

Parameters: Parameters are the same as for the method *XolWidgetC::CreateVa*

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolWidgetC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolManagerC*)

virtual void HelpEvent();

Description: This method is called by the callback handler of *this* manager widget object. The method is a virtual dummy method and has to be overwritten by derived class to realize real functionality.

Parameters: None

Results: None

8.3 Class XolPrimitiveC

The class *XolPrimitiveC* encapsulates the *OSF/Motif* widget class *Primitive*. The *OSF/Motif primitive* widget class is the basis class for all simple widget that realize controls. Controls are widgets with which the user interacts directly. They are managed by more complex widgets that are subclasses of the *OSF/Motif* manager widgets.
The class is derived from the general *XOL* widget object class *XolWidgetC*. It registers a callback handler for the *OSF/Motif* callback reason *XmCR_HELP* automatically. The callback handler executes the method *HelpEvent* that is a virtual dummy method. The method *HelpEvent* can be overwritten with real functionality by derived widget object classes. Thus, help functionality can be realized for all widget objects that are derived from the primitive widget object class.

8.3.1 Class Declaration

```
#include <XOL/Widget.h>

class XolPrimitiveC : public XolWidgetC
{
   static void CallbackHandler(Widget, XolPrimitiveC*, XolPointerT);

  public:

    XolPrimitiveC (XolWidgetC *pParent, char *pName,
                   WidgetClass pClass = xmPrimitiveWidgetClass)
                 : XolWidgetC(pParent, pName, pClass) {}

    XolPrimitiveC (Widget pParent, char *pName,
                   WidgetClass pClass = xmPrimitiveWidgetClass)
                 : XolWidgetC(pParent, pName, pClass) {}

    virtual int Create(bool pManaged = TRUE, ArgList pArgV = NULL
                                            , int pArgC = 0);

    virtual char *GetClass () { return "XolPrimitiveC"; }
    virtual void HelpEvent () { return; }  // virtual dummy callback function
};
```

8.3.2 Important Data Members

This class defines no data members.

8.3.3 Usable Methods and Operators

XolPrimitiveC (XolWidgetC *pParent, char *pName,
WidgetClass pClass = xmPrimitiveWidgetClass);
XolPrimitiveC (Widget pParent, char *pName,
WidgetClass pClass = xmPrimitiveWidgetClass);

Description: The constructors execute only the constructors of the basis class. For further information see *XolWidgetC*.

Parameters:
pParent - Pointer to the parent widget object or the parent *Xt* widget handle
pName - Name of the widget object
pClass - *OSF/Motif* class of the widget object (by default the *OSF/Motif* class *xmPrimitiveWidgetClass* is used)

Results: None

virtual int Create (bool pManaged = TRUE, ArgList pArgV = NULL
, int pArgC = 0);

Description: This method overwrites the virtual method *XolWidgetC::CreateVa*. It executes the same method of the basis class. Furthermore, it registers the callback handler as the callback method for the *OSF/Motif* callback resource *XmCR_HELP*.

Parameters: Parameters are the same as for the method *XolWidgetC::CreateVa*

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolWidgetC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolPrimitiveC*)

virtual void HelpEvent();

Description: This method is called by the callback handler of *this* primitive widget object. The method is a virtual dummy method and has to be overwritten by derived class to realize real functionality.

Parameters: None

Results: None

8.4 Class XolGadgetC

The class *XolGadgetC* encapsulates the *OSF/Motif* widget class *Gadget* within an *XOL* widget object class. For this purpose the class is derived from the general *XOL* widget object class *XolWidgetC*. The *OSF/Motif Gadget* is a widget without an own window. Only the most simplest widgets are also available as gadget classes. This class registers a callback handler for the *OSF/Motif* callback reason *XmCR_HELP* automatically. The callback handler executes the method *HelpEvent* that is a virtual dummy method. The method *HelpEvent* can be overwritten with real functionality by derived widget object classes. Thus, help functionality can be realized for all widget objects that are derived from the primitive widget object class.

8.4.1 Class Declaration

```
#include <XOL/Widget.h>

class XolGadgetC : public XolWidgetC
{
   static void CallbackHandler(Widget, XolGadgetC*, XolPointerT);

  public:

    XolGadgetC (XolWidgetC *pParent, char *pName,
                WidgetClass pClass = xmGadgetClass)
              : XolWidgetC(pParent, pName, pClass) {}

    XolGadgetC (Widget pParent, char *pName,
                WidgetClass pClass = xmGadgetClass)
              : XolWidgetC(pParent, pName, pClass) {}
  public:

    virtual int Create(bool pManaged = TRUE, ArgList pArgV = NULL
                                          , int pArgC = 0);

    virtual char *GetClass () { return "XolGadgetC"; }
    virtual void HelpEvent () { return; }  // virtual dummy callback function
};
```

8.4.2 Important Data Members

This class defines no data members.

8.4.3 Usable Methods and Operators

XolGadgetC (XolWidgetC *pParent, char *pName,
WidgetClass pClass = xmGadgetClass);
XolGadgetC (Widget pParent, char *pName,
WidgetClass pClass = xmGadgetClass);

Description: The constructors execute only the constructors of the basis class. For further information see *XolWidgetC*.

Parameters:	*pParent*	- Pointer to the parent widget object or the parent *Xt* widget handle
	pName	- Name of the widget object
	pClass	- *OSF/Motif* class of the widget object (by default the *OSF/Motif* class *xmGadgetClass* is used)
Results:	None	

virtual int Create (bool pManaged = TRUE, ArgList pArgV = NULL , int pArgC = 0);

Description: This method overwrites the virtual method *XolWidgetC::CreateVa*. It executes the same method of the basis class. Furthermore, it registers the callback handler as the callback method for the *OSF/Motif* callback resource *XmCR_HELP*.

Parameters: Parameters are the same as for the method *XolWidgetC::CreateVa*

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolWidgetC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolGadgetC*)

virtual void HelpEvent();

Description: This method is called by the callback handler of *this* gadget object. The method is a virtual dummy method and has to be overwritten by derived class to realize real functionality.

Parameters: None

Results: None

8.5 Class XolMenuC

The class *XolMenuC* realizes the basis class for all kind of *menu widget* objects. This class belongs to the set of framework classes rather than to the set of common widget classes, since it does not simply encapsulate any *OSF/Motif* widget class. It uses the *OSF/Motif* widget class *RowColumn* to realize the different kinds of menu widget objects. For this purpose, it encapsulates the whole *OSF/Motif* functionality for menu handling within a single class.
An instanced menu widget object can be a menu bar, a pull down menu, a pop-up menu, and a option menu. Although this class defines all necessary data members and methods, the class is normally not used directly. Therefore, a specialised menu object class is derived from this class for each kind of possible menu. A special menu object is instanced from one of these derived and more specialised menu widget object classes. For more information about the specialised menu widget see section 8.6, section 8.7, section 8.8, and section 8.9. By means of menu item objects instanced from class *XolMenuItemC*, see section 8.10 and menu objects a menu object hierarchy can be established in an easy way.
For a menu object a set of menu item objects can be created automatically during the construction. The menu item objects have to be specified within an array of type *XolMenuItemT*. The last entry of this array have to be empty, i.e., all elements of the last entry are *NULL*.
The class *XolMenuC* is derived from class *XolManagerC*. The inherited methods *XolManagerC::CreateVa* and *XolManagerC::Create* MUST NOT BE USED, because the underlying *OSF/Motif* widget is created automatically by the constructor. Therefore these methods are overwritten by virtual dummy methods.

8.5.1 Class Declaration

```
#include <XOL/Menu.h>

class XolMenuC : public XolManagerC
{
    Widget      mCascade;
    XolMenuT    mMenuType;

  public:

    // Constructors

    XolMenuC(XolWidgetC *pParent, char *pTitle   = "Menu"
                                , char pMnemonic = 0
                                , XolMenuT       pType  = XOL_PD
                                , XolMenuItemT*  pItems = NULL);

    // Dummy methods to avoid using the basis class methods

    virtual int  Create   (Widget)             { return 0; }
    virtual int  CreateVa (bool, ...)          { return 0; }
    virtual int  Create   (bool, ArgList, int) { return 0; }

    // Miscellaneous methods

    void Exposure(XButtonPressedEvent *pEvent); // only for pop-up menus

    virtual char *GetClass() { return "XolMenuC"; }

};
```

8.5.2 Important Data Members

XolMenuT mMenuType;

Description: This data member holds the type of the menu realized by the instanced object.

Values: One of the values of the enumeration type *XolMenuT*.

XOL_MB - Menu bar
XOL_PD - Pull down menu
XOL_PU - Pop up menu
XOL_OM - Option menu

Widget mCascade;

Description: If the menu widget object realizes an option menu, this data member holds the cascaded option menu widget. Otherwise this data member is *NULL*.

Values: *Xt* Widget handle

8.5.3 Usable Methods/Operator

XolMenuC(XolWidgetC *pParent, char *pTitle ="Menu"
, char pMnemonic = 0
, XolMenuT pType = XOL_PD
, XolMenuItemT* pItems = NULL);

Description: This method constructs a menu widget object based on a *XOL* manager widget object. Furthermore, the method creates the underlying *OSF/Motif* widget by using the method *XolWidgetC::Create*.

Parameters:
- *pParent* - Pointer to the parent widget object
- *pTitle* - Name of the menu object (by default the name is "Menu")
- *pMnemonic*- Mnemonic character for the menu item created in the parent menu (only relevant for pull down menu objects, by default there is no mnemonic character)
- *pType* - Menu type (by default the menu type is pull down)
- *pItems* - Menu item objects specification that can be used to create a set of menu item objects for *this* menu object; the last entry has to be empty, i.e., all elements are *NULL* (by default there are no menu item objects)

Results: None

void Exposure(XButtonPressedEvent *pEvent);

Description: This method is relevant for pop-up menu objects. For all other menu object his method leads to a dummy action. The method has to be called by its parent widget object caused by the *OSF/Motif* event *XmCR_EXPOSE*.

Parameters:
- *pEvent* - Event structure that has triggered the callback method of the parent widget object

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolManagerC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolMenuC*)

8.6 Class XolMenuBarC

This class realizes *menu bar* widget objects. It is derived from the class *XolMenuC*. Since the basis class *XolMenuC* defines already the functionality for all kinds of menus this class defines only the more specialised constructor.

8.6.1 Class Declaration

```
#include <XOL/Menu.h>

class XolMenuBarC : public XolMenuC
{
  public:

    XolMenuBarC (XolWidgetC *pParent, char *pTitle = "MenuBar"
                                 , XolMenuItemT* pItems = NULL)
               : XolMenuC(pParent, pTitle, 0, XOL_MB, pItems) {}

    virtual char *GetClass() { return "XolMenuBarC"; }
};
```

8.6.2 Important Data Members

The class defines no further data members.

8.6.3 Usable Methods and Operators

XolMenuBarC (XolWidgetC *pParent, char *pTitle = "MenuBar");

Description: This method constructs a menu bar widget object by using the basis class constructor with the menu object type *XOL_MB*. Furthermore, the method creates the underlying *OSF/Motif* widget by using the method *XolWidgetC::Create*.

Parameters: *pParent* - Pointer to the parent widget object
pTitle - Menu bar name (the name is used only for naming, by default the name is "MenuBar")
pItems - Menu item objects specification that can be used to create a set of menu item objects for *this* menu object; the last entry has to be empty, i.e., all elements are *NULL* (by default there are no menu item objects)

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolMenuBarC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolMenuBarC*)

8.7 Class XolPullDownMenuC

This class realizes pull down menu objects based on the basis class *XolMenuC*. Pull down menu objects are menu widgets that become visible when a menu item object of another menu object on the next higher level in the menu object hierarchy is invoked. A typical pull down menu layout is shown in Fig. 3.

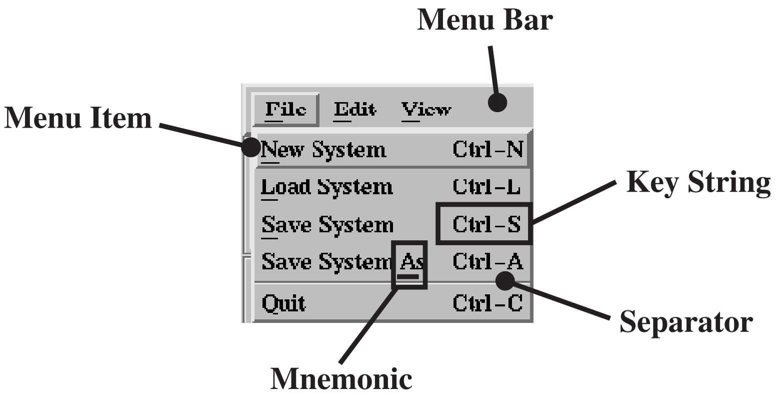

Figure 3: Typical pull down menu with several menu items

A set of pull down menu objects can be connected to a pull down menu hierarchy. For this purpose a new pull down menu is attached to a menu item of the menu on next higher level. To realize this with the classes for menu widget objects, a menu item object of class *XolMenuItemC*, see section 8.10, is created automatically within the parent menu widget object. For this reason the parent of a pull down menu object has to be a menu object or an object of one of the derived menu object classes. If the parent menu object is a menu bar object, then the parent menu object serves as the root of the pull down menu hierarchy. A pull down menu object with name *Help* is handled in a different way. It is registered at the parent widget object as the *help* pull down menu. This realizes the *OSF/Motif* users style that the *help* button is attached at the right margin of the menu bar. The class *XolPullDownMenuC* is derived from the class *XolMenuC*. Since the basis class *XolMenuC* defines already the functionality for all kinds of menus this class defines only a more specialised constructor.

8.7.1 Class Declaration

```
#include <XOL/Menu.h>

class XolPullDownMenuC : public XolMenuC
{
  public:

    XolPullDownMenuC (XolWidgetC *pParent, char *pTitle = "PullDownMenu"
                                         , char pMnemonic = 0
                                         , XolMenuItemT* pItems = NULL)
                    : XolMenuC(pParent, pTitle, pMnemonic, XOL_PD, pItems) {}

    virtual char *GetClass() { return "XolPullDownMenuC"; }
};
```

8.7.2 Important Data Members

The class defines no further data members.

8.7.3 Usable Methods and Operators

XolPullDownMenuC (XolWidgetC *pParent, char *pTitle = "PullDownMenu"
, char pMnemonic = 0)

Description: This method constructs a pull down menu widget object and its underlying *OSF/Motif* widget by using the basis class constructor with the menu object type *XOL_PD*. The method creates the underlying *OSF/Motif* widget by using the method *XolWidgetC::Create*. Furthermore, it creates a menu item for *this* pull down menu object within the parent widget object. Therefore, the parent widget object has to be instanced from one of the menu object classes.

Parameters:
- *pParent* - Pointer to the parent menu widget object
- *pTitle* - Pull down menu name (the name is used also as the string of the menu item object in the parent menu object, by default the name is "PullDownMenu")
- *pMenmonic*- Mnemonic character for the menu item object in the parent menu object (by default there is no mnemonic character)
- *pItems* - Menu item objects specification that can be used to create a set of menu item objects for *this* menu object; the last entry has to be empty, i.e., all elements are *NULL* (by default there are no menu item objects)

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolPullDownMenuC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolPullDownMenuC*)

8.8 Class XolPopupMenuC

This class realizes pop up menu widget objects. Pop-up menu objects are menu widget objects that are laid over their parent widget object by pressing the right mouse button. For this purpose, the parent widget object has to be an instance of a widget class that can receive the *OSF/Motif* event *XmCR_EXPOSE*, e.g., the *OSF/Motif* widget *DrawingArea*. Moreover, The parent widget object has to register an appropriate method as callback method in that the method *Exposure* of *this* pop-up menu object is called.
Pop-up menu objects can contain menu item objects that refers to pull down or option menu objects. Thus pop down menus can also serve as root menu objects for menu object hierarchies. The class *XolPopupMenuC* is derived from the class *XolMenuC*. Since the basis class *XolMenuC* defines already the functionality for all kinds of menus this class defines only a more specialised constructor.

8.8.1 Class Declaration

```
#include <XOL/Menu.h>

class XolPopupMenuC : public XolMenuC
{
  public:

    XolPopupMenuC (XolWidgetC *pParent, char *pTitle = "PopupMenu"
                                  , XolMenuItemT* pItems = NULL)
                : XolMenuC(pParent, pTitle, 0, XOL_PU, pItems) {}

    virtual char *GetClass() { return "XolPopupMenuC"; }
};
```

8.8.2 Important Data Members

The class defines no further data members.

8.8.3 Usable Methods and Operators

XolPopupMenuC (XolWidgetC *pParent, char *pTitle = "PopupMenu")

Description: This method constructs a pop-up menu widget object by using the basis class constructor with the menu object type *XOL_PU*. Furthermore, the method creates the underlying *OSF/Motif* widget by using the method *XolWidgetC::Create*.

Parameters: *pParent* - Pointer to the parent widget object (it has to be a widget object that can handle the *OSF/Motif* event *XmCR_EXPOSE*
pTitle - pop-up menu name (the name is used only for naming, by default the name is "PopupMenu")
pItems - Menu item objects specification that can be used to create a set of menu item objects for *this* menu object; the last entry has to be empty, i.e., all elements are *NULL* (by default there are no menu item objects)

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolPopupMenuC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolPopupMenuC*)

8.9 Class XolOptionMenuC

This class realizes option menu objects. Option menu objects are similar to pull down menu objects. The difference between an option menu object and a pull down menu object is, that the option menu widget is laid over the associated menu item instead it is dropping down from it. The class *XolOptionMenuC* is derived from the class *XolMenuC*. Since the basis class *XolMenuC* defines already the functionality for all kinds of menus this class defines only a more specialised constructor.

8.9.1 Class Declaration

```
#include <XOL/Menu.h>

class XolOptionMenuC : public XolMenuC
{
  public:
```

```
    XolOptionMenuC (XolWidgetC *pParent, char *pTitle = "OptionMenu"
                                       , XolMenuItemT* pItems = NULL)
                : XolMenuC(pParent, pTitle, 0, XOL_OM, pItems) {}

    virtual char *GetClass() { return "XolOptionMenuC"; }
};
```

8.9.2 Important Data Members

The class defines no further data members.

8.9.3 Usable Methods and Operators

XolOptionMenuC (XolWidgetC *pParent, char *pTitle = "OptionMenu")

Description: This method constructs a option menu widget object and its underlying *OSF/Motif* widget by using the basis class constructor with the menu object type *XOL_OM*. The method creates the underlying *OSF/Motif* widget by using the method *XolWidgetC::Create*. Furthermore, it creates a menu item for *this* option menu object within the parent widget object. Therefore, the parent widget object has to be instanced from one of the menu object classes.

Parameters:
pParent - Pointer to the parent menu widget object
pTitle - Option menu name (the name is used also as the string of the menu item object in the parent menu object, by default the name is "OptionMenu")
pItems - Menu item objects specification that can be used to create a set of menu item objects for *this* menu object; the last entry has to be empty, i.e., all elements are *NULL* (by default there are no menu item objects)

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolOptionMenuC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolOptionMenuC*)

8.10 Class XolMenuItemC

This class realizes all different kinds of menu item objects

- push buttons,
- toggle buttons, and
- separators.

Each kind of menu item can be realized as a *OSF/Motif* widget or a *OSF/Motif* gadget. Whereas the push buttons and toggle buttons are active menu items, the separators are passive. They are only be used for grouping active menu items. Active menu items can trigger actions registered as callback function. Furthermore, push buttons can invoke the opening of a menu object on the next lower level in the menu hierarchy. Such menu item objects are created automatically when a pull down or option menu object is instanced. On selecting a toggle button menu item the state of the toggle visible is changed and the action registered as call back function is called.
For active menu item objects a mnemonic character can be specified. A mnemonic character is one character contained in the menu item label that identifies the menu item uniquely within the menu object. The associated menu item object can selected by pressing the appropriate key. The mnemonic character is displayed with underline. Furthermore, accelerator keys can be defined. Accelerator keys are also called hot key, because they allow the user the direct selection of a menu item without stepping through the menu hierarchy. The layout of different kinds of menu items can be seen in Fig. 3.

Every menu item can hold an additional data element of any type casted to *XolPointerT*. These data elements can be used for passing additional information to the callback method called on selecting of a menu item object. For example, in each menu item object an unique number can be stored. In a method that is registered as the callback method for more than one menu item object, the additional information can be used for identification of the menu item object.
Methods that are registered as callback methods for menu item objects have to be declared static and have to be of type *XolCallbackProcT*. On calling the callback method, the pointer to the selected menu item object is given by the callback user data parameter.
The class is derived from the class *XolManagerC*. The inherited methods *XolManagerC::CreateVa* and *XolManagerC::Create* MUST NOT BE USED, because the underlying *OSF/Motif* widget is created automatically by the constructor. Although this class defines all necessary data members and methods to realize all different kind of menu item objects, the class is normally not used directly. A specialised menu object class is derived from this class for each kind of menu item. A special menu item object is instanced from one of the derived and more specialised menu item object classes. For more information about the specialised menu widget see section 8.11... section 8.16.

8.10.1 Class Declaration

```
#include <XOL/Menu.h>

class XolMenuItemC : public XolPrimitiveC
{
    void Create(char  pMnemonic = 0, char* pAccel     = NULL
                                   , char* pAccelText = NULL
                                   , XolCallbackProcT pCallback = NULL
                                   , XolPointerT      pItemData = NULL);
  protected:

    XolMenuItemC (XolMenuC *pParent, char *pLabel   // for internal usage
                                   , char pMnemonic, XolMenuC *pChild);
  public:

    XtPointerT mItemData;

    // Constructors

    XolMenuItemC (XolMenuC *pParent, char* pLabel, WidgetClass pClass
                                   , char  pMnemonic  = 0
                                   , char* pAccel     = NULL
                                   , char* pAccelText = NULL
                                   , XolCallbackProcT pCallback = NULL
                                   , XolPointerT      pItemData = NULL);

    XolMenuItemC (XolMenuC *pParent, char *pName = "Seperator");
    XolMenuItemC (XolMenuC *pParent, XolMenuItemT *pItem);

    // Miscellaneous

    virtual char *GetClass() { return "XolMenuItemC"; }

    friend XolMenuC;
};
```

8.10.2 Important Data Members

XolPointerT mItemData;

Description: This data member hold any type of menu item associated information casted to the type *XolPointerT*. The data member can be used for passing additional information to the callback method called on selecting of a menu item object. For example, in each menu item object an unique number can be stored. In a method that is registered as the callback method for more than one menu item object, the additional information can be used for identification of the menu item object.

Values: Any value casted to the type *XolPointerT*

8.10.3 Usable Methods and Operators

XolMenuItemC (XolMenuC *pParent, char* pLabel, WidgetClass pClass
, char pMnemonic = 0
, char* pAccel = NULL
, char* pAccelText = NULL
, XolCallbackProcT pCallback = NULL
, XolPointerT pItemData = NULL);

Description: This method is the default constructor. It constructs a menu item object of type given by parameter *pClass*. A menu item object can have a mnemonic character as well a accelerator key (hot key). For the menu item object a callback method given by the parameter *pCallback* can be registered that is called on selecting this menu item.

Parameters:
pParent - Pointer to the parent menu widget object to that the menu item object belongs
pLabel - Menu item label that is used for naming as well that is the string displayed within the menu item object
pClass - Menu item class, the following values are recognised
XOL_PBG - xmPushButtonGadgetClass
XOL_PBW - xmPushButtonWidgetClass
XOL_TBG - xmToggleButtonGadgetClass
XOL_TBW - xmToggleButtonWidgetClass
XOL_SPG - xmSeparatorGadgetClass
XOL_SPW - xmSeparatorWidgetClass
pMnemonic- Mnemonic character (by default there is no mnemonic character)
pAccel - Accelerator key, e.g., Ctrl<key>-C
pAccelText- Accelerator text that is displayed at the right side of the menu item
pCallback- callback method that is called when *this* menu item is selected
pItemData- Additional item associated data as discussed before

Results: None

XolMenuItemC (XolMenuC *pParent, char *pName = "Seperator");

Description: This method constructs a simple separator gadget.

Parameters:
pParent - Pointer to the parent menu widget object to that the menu item object belongs
pName - Name of the separator widget, it is not displayed (by default the name is "Separator")

Results: None

XolMenuItemC (XolMenuC *pParent, XolMenuItemT *pItem);

Description: This method constructs a menu item object from the menu item specification given by the parameter *pItem*. The meaning of the elements of the menu item specification is the same as for the default constructor parameters, see above.

Parameters:
pParent - Pointer to the parent menu widget object to that the menu item object belongs
pItem - Menu item specification

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolManagerC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolMenuItemC*)

8.10.4 Example

The following example shows the using of menu bar and menu items. In the example the methods *Quit*, *Open*, and *Style* shall be declared as callback methods. They are of needed type *XolCallbackProcT*. The global application object is already created but not managed. Than a simple menu hierarchy can be defined as shown in the example. There are two temporary variables *Menu1* and *Menu2* for holding pull down menu widget objects. The menus on the first pull down level are created with the menu bar widget object as parent that is managed by the global application object. These pull down menus are the parents for the pull down menu objects on second menu hierarchy level. Push button menu item definitions within the fist level pull down menu *File* use almost the the whole set of possible parameters complete parameters. The definitions of menu items for the second level pull down menu *Line Style* use type casting to cast the different typed function *Style* to the necessary callback method type. Also in these defintions the possibility to define callback user data is used. They define the ordinal number of the menu item as the callback user data parameter. Figure 4 shows the result of the sample source code.

```
void Quit  (Widget, XolWidgetC*  , XolPointerT);
void Open  (Widget, XolWidgetC*  , XolPointerT);

void Style (Widget, XolMenuItemC* Item, XolPointerT Data)
{
  if (Data == 1)
    Item->AnyFunction("Any1");
  else if (Data == 2)
    Item->AnyFunction("Any2");
};

XolPullDownMenuC *Menu1,
                 *Menu2;

Menu1 = new XolPullDownMenuC (sGlobApp->mWorkPlace->mMenuBar, "File", 'F');

  new XolPushItemWC (Menu1, "Open", 'O', "Ctrl<Key>O", "Ctrl-O", Open);
  new XolPushItemWC (Menu1, "Save", 'S');
  new XolSepItemWC  (Menu1);
  new XolPushItemWC (Menu1, "Quit", 'Q', "Ctrl<Key>C", "Ctrl-C", Quit);

Menu1 = new XolPullDownMenuC (sGlobApp->mWorkPlace->mMenuBar, "Lines", 'L');

  Menu2 = new XolPullDownMenuC (Menu1, "Line Weight", 'W');

    new XolPushItemWC (Menu2, "1", '1');
    new XolPushItemWC (Menu2, "2", '2');
    new XolPushItemWC (Menu2, "3", '3');
    new XolPushItemWC (Menu2, "4", '4');

  Menu2 = new XolPullDownMenuC (Menu1, "Line Color", 'C');

    new XolPushItemWC (Menu2, "Cyan"   , 'C', "Alt<Key>C", "Alt-C");
    new XolPushItemWC (Menu2, "Yellow" , 'Y', "Alt<Key>Y", "Alt-Y");
```

```
    new XolPushItemWC (Menu2, "Magenta", 'M', "Alt<Key>M", "Alt-M");

  Menu2 = new XolPullDownMenuC (Menu1, "Line Style", 'S');

    new XolPushItemWC (Menu3, "Single", 'S', "Alt<Key>S", "Alt-S"
                                        , (XolCallbackProcT)Style
                                        , (XolPointerT)1);
    new XolPushItemWC (Menu3, "Double", 'D', "Alt<Key>D","Alt-D"
                                        , (XolCallbackProcT)Style
                                        , (XolPointerT)2);

Menu1 = new XolPullDownMenuC (sGlobApp->mWorkPlace->mMenuBar, "Help", 'H');

sGlobApp->mWorkPlace->mMenuBar->Manage();
```

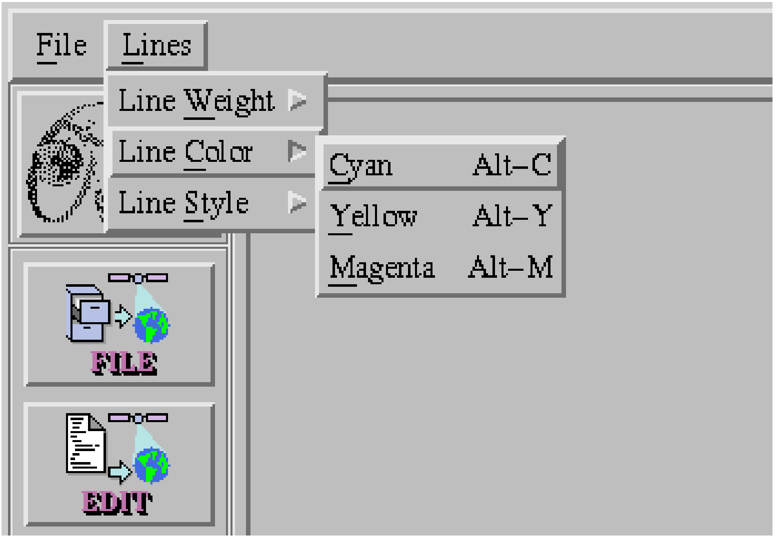

Figure 4: Menu hierarchy example

8.11 Class XolToggleItemGC

This class realizes toggle menu item objects without its own window. The class is derived from the class *XolMenuItemC*. Since the basis class *XolMenuItemC* defines already the functionality for all kinds of menu item objects, this class defines only a more specialised constructor.

8.11.1 Class Declaration

```
#include <XOL/Menu.h>

class XolToggleItemGC : public XolMenuItemC
{
  public:

    XolToggleItemGC (XolMenuC *pParent, char* pLabel
                                      , char  pMnemonic  = 0
                                      , char* pAccel     = NULL
                                      , char* pAccelText = NULL
                                      , XolCallbackProcT pCallback = NULL
                                      , XolPointerT      pItemData = NULL)
```

```
                   : XolMenuItemC(pParent, pLabel, XOL_TBG
                                                  , pMnemonic
                                                  , pAccel
                                                  , pAccelText
                                                  , pCallback
                                                  , pItemData) {}

    virtual char *GetClass() { return "XolToggleItemGC"; }
};
```

8.11.2 Important Data Members

The class defines no further data members.

8.11.3 Usable Methods and Operators

XolToggleItemGC (XolMenuC *pParent, char* pLabel
, char pMnemonic = 0
, char* pAccel = NULL
, char* pAccelText = NULL
, XolCallbackProcT pCallback = NULL
, XolPointerT pItemData = NULL);

Description: This method constructs a toggle menu item.

Parameters: The meaning of parameters is the same as for the default constructor of the basis class *XolMenuItemC*, see section 8.10

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolMenuItemC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolToggleItemGC*)

8.12 Class XolToggleItemWC

This class realizes toggle menu item objects wit its own window. The class is derived from the class *XolMenuItemC*. Since the basis class *XolMenuItemC* defines already the functionality for all kinds of menu item objects, this class defines only a more specialised constructor.

8.12.1 Class Declaration

```
#include <XOL/Menu.h>

class XolToggleItemWC : public XolMenuItemC
{
  public:

    XolToggleItemWC (XolMenuC *pParent, char* pLabel
                                      , char  pMnemonic  = 0
                                      , char* pAccel     = NULL
                                      , char* pAccelText = NULL
                                      , XolCallbackProcT pCallback = NULL
                                      , XolPointerT      pItemData = NULL)
                   : XolMenuItemC(pParent, pLabel, XOL_TBW
                                                  , pMnemonic
```

```
                                                  , pAccel
                                                  , pAccelText
                                                  , pCallback
                                                  , pItemData) {}

    virtual char *GetClass() { return "XolToggleItemWC"; }
};
```

8.12.2 Important Data Members

The class defines no further data members.

8.12.3 Usable Methods and Operators

XolToggleItemWC (XolMenuC *pParent, char* pLabel
, char pMnemonic = 0
, char* pAccel = NULL
, char* pAccelText = NULL
, XolCallbackProcT pCallback = NULL
, XolPointerT pItemData = NULL);

Description: This method constructs a toggle menu item.

Parameters: The meaning of parameters is the same as for the default constructor of the basis class *XolMenuItemC*, see section 8.10

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolMenuItemC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolToggleItemWC*)

8.13 Class XolPushItemGC

This class realizes push menu item objects without its own window. The class is derived from the class *XolMenuItemC*. Since the basis class *XolMenuItemC* defines already the functionality for all kinds of menu item objects, this class defines only a more specialised constructor.

8.13.1 Class Declaration

```
#include <XOL/Menu.h>

class XolPushItemGC : public XolMenuItemC
{
  public:

    XolPushItemGC (XolMenuC *pParent, char* pLabel
                                 , char  pMnemonic  = 0
                                 , char* pAccel     = NULL
                                 , char* pAccelText = NULL
                                 , XolCallbackProcT pCallback = NULL
                                 , XolPointerT      pItemData = NULL)
             : XolMenuItemC(pParent, pLabel, XOL_PBG
                                         , pMnemonic
                                         , pAccel
                                         , pAccelText
```

```
                                              , pCallback
                                              , pItemData) {}

    virtual char *GetClass() { return "XolPushItemGC"; }
};
```

8.13.2 Important Data Members

The class defines no further data members.

8.13.3 Usable Methods and Operators

XolPushItemGC (XolMenuC *pParent, char* pLabel
, char pMnemonic = 0
, char* pAccel = NULL
, char* pAccelText = NULL
, XolCallbackProcT pCallback = NULL
, XolPointerT pItemData = NULL);

Description: This method constructs a push menu item.

Parameters: The meaning of parameters is the same as for the default constructor of the basis class *XolMenuItemC*, see section 8.10

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolMenuItemC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolPushItemGC*)

8.14 Class XolPushItemWC

This class realizes push menu item objects wit its own window. The class is derived from the class *XolMenuItemC*. Since the basis class *XolMenuItemC* defines already the functionality for all kinds of menu item objects, this class defines only a more specialised constructor.

8.14.1 Class Declaration

```
#include <XOL/Menu.h>

class XolPushItemWC : public XolMenuItemC
{
  public:

    XolPushItemWC (XolMenuC *pParent, char* pLabel
                                 , char  pMnemonic  = 0
                                 , char* pAccel     = NULL
                                 , char* pAccelText = NULL
                                 , XolCallbackProcT pCallback = NULL
                                 , XolPointerT      pItemData = NULL)
                 : XolMenuItemC(pParent, pLabel, XOL_PBW
                                              , pMnemonic
                                              , pAccel
                                              , pAccelText
                                              , pCallback
                                              , pItemData) {}
```

```
    virtual char *GetClass() { return "XolPushItemWC"; }
};
```

8.14.2 Important Data Members

The class defines no further data members.

8.14.3 Usable Methods and Operators

XolPushItemWC (XolMenuC *pParent, char* pLabel
, char pMnemonic = 0
, char* pAccel = NULL
, char* pAccelText = NULL
, XolCallbackProcT pCallback = NULL
, XolPointerT pItemData = NULL);

Description: This method constructs a push menu item.

Parameters: The meaning of parameters is the same as for the default constructor of the basis class *XolMenuItemC*, see section 8.10

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolMenuItemC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolPushItemWC*)

8.15 Class XolSepItemGC

This class realizes separator menu item objects without its own window. The class is derived from the class *XolMenuItemC*. Since the basis class *XolMenuItemC* defines already the functionality for all kinds of menu item objects, this class defines only a more specialised constructor.

8.15.1 Class Declaration

```
#include <XOL/Menu.h>

class XolSepItemGC : public XolMenuItemC
{
  public:

    XolSepItemGC (XolMenuC *pParent, char *pName = "Separator")
                : XolMenuItemC(pParent, pName , XOL_SPG) {}

    virtual char *GetClass() { return "XolSepItemGC"; }
};
```

8.15.2 Important Data Members

The class defines no further data members.

8.15.3 Usable Methods and Operators

XolSepItemGC (XolMenuC *pParent, char* pName = "Separator");

Description: This method constructs a separator menu item.

Parameters: The meaning of parameters is the same as for the default constructor of the basis class *XolMenuItemC*, see section 8.10

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolMenuItemC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolSepItemGC*)

8.16 Class XolSepItemWC

This class realizes separator menu item objects wit its own window. The class is derived from the class *XolMenuItemC*. Since the basis class *XolMenuItemC* defines already the functionality for all kinds of menu item objects, this class defines only a more specialised constructor.

8.16.1 Class Declaration

```
#include <XOL/Menu.h>

class XolSepItemWC : public XolMenuItemC
{
  public:

    XolSepItemWC (XolMenuC *pParent, char *pName = "Separator")
                : XolMenuItemC(pParent, pName , XOL_SWG) {}

    virtual char *GetClass() { return "XolSepItemWC"; }
};
```

8.16.2 Important Data Members

The class defines no further data members.

8.16.3 Usable Methods and Operators

XolSepItemWC (XolMenuC *pParent, char* pName = "Separator");

Description: This method constructs a separator menu item.

Parameters: The meaning of parameters is the same as for the default constructor of the basis class *XolMenuItemC*, see section 8.10

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolMenuItemC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolSepItemWC*)

8.17 Class XolWmShellC

Widgets that are not managed by its parent window have to be managed by the *X Windows* manager. They are independent from their parent widgets with respect to their geometry and their representation. They negotiate their geometry and representation with the window manager. For this purpose, they have to exchange data with the window manager according to the ICCC (Inter-Client Communications Convention). Such widgets are called shell widgets, because they provide their child widgets with an own *X Windows* shell. They are invisible with exception of their window manager decorations. Furthermore, shell widgets have only one child, the client area widget. Such a child widget is normally a manager that manage further children. The *Xt* WMShell widget class implements the ICCC-compliant behaviour. WMShells are simple, wire bed-frames that have no special attributes. They can have different decoration that are realized by the window manager.
The class *XolWmShellC* encapsulates the *Xt* widget class *WmShell*. All objects that are instanced from this class or a derived shell widget object class are represented as shell widgets. The class is derived from the *XOL* widget object class *XolWidgetC*. It realizes the functionality for the different kinds of shell widgets *top-level shell widgets*, *transient shell widgets*, *dialog shell widgets*, and *application shell widget*. *Top-level shell widgets* are all widgets that use top-level windows with window manager decorations independently from their parent widgets. *Transient shell widgets* are similar to top-level shell widgets. However transient shell widgets cannot be iconified independently from their parent widgets. This means that a transient shell widget is iconified when the associated application shell widget is iconified. *Dialog shell widgets* are transient shells without any window manager decorations that provide dialog specific widgets with a shell that is displayed over the top of its parent widget. *Application shell widgets* are top-level shell that contains the applications main window widget. An application should have only one widget of this class. For all classes of shell widgets objects excepted the dialog shell widget, the only one child object, the client area object is created automatically. It is an object of the *OSF/Motif* widget class *MainWindow*. All children of this shell widget have to be created with this widget object as its parent. However, the user can set the client area widget object pointer to its own already constructed widget object. In this case, the shell's client area widget is not created automatically.
Objects are not instanced from the class *XolWmShellC* directly, but from the more specialised classes, see section 8.19 .. section 8.20. The *Xt* widget class *ApplicationShell* was not realized as a derivative of this class, because the library implements a higher sophisticated application widget object that belongs to framework classes. However, application shell widgets can be realized also with this class. The class *XolWmShellC* belongs to common widget classes because it encapsulates directly the defined *Xt* and *OSF/Motif* widget classes.
Often, objects instanced from a shell widget class are used as top-level shell widget. Therefore, the class *XolWmShellC* realizes by default objects that are top-level shell widget objects. In this case, its method *Create* creates a shell widget and in addition an *OSF/Motif* main window widget automatically. After using the *Create* method, the method *Manage* has to be used to bring up the Shell window on the display.

8.17.1 Class Declaration

```
#include <XOL/Shell.h>

class XolWmShellC : public XolWidgetC
{
    bool mManaged;

  public:

    XolWidgetC *mClientArea;

    // Constructors

    XolWmShellC (XolWmShellC *pParent, char *pName,
                 WidgetClass pClass = topLevelShellWidgetClass)
               : XolWidgetC(pParent, pName, pClass), mManaged(FALSE)
                                                   , mClientArea(NULL) {}

    XolWmShellC (Widget pParent, char *pName,
                 WidgetClass pClass = topLevelShellWidgetClass)
```

```
            : XolWidgetC(pParent, pName, pClass), mManaged(FALSE)
                                                , mClientArea(NULL) {}

    // Methods that creates the Motif widgets.

    virtual int CreateRel(bool pScroll = FALSE, int xPos  = 30, int yPos  = 30
                                              , int xSize = 0 , int ySize = 0);
    virtual int Create   (bool pScroll = FALSE, int xPos  = 0 , int yPos  = 0
                                              , int xSize = 0 , int ySize = 0);
    virtual void Manage();

    // Methods that handle attributes

    virtual void SetTitle     (char* pTitle);
    virtual void SetIconLabel (char* pLabel);
    virtual void SetIcon      (XolPixmapC* pIcon);
    virtual void SetIcon      (Pixmap      pIcon);
    virtual void SetMinSize   (long dx, long dy);
    virtual void SetMaxSize   (long dx, long dy);

    // Miscellaneous methods

    virtual char *GetClass()   { return "XolWmShellC"; }
};
```

8.17.2 Important Data Members

XolWidgetC *mClientArea;

Description: This data member holds the the client area widget object for the top level shell classes that is normally created automatically by using the method *Create* or *CreateRel*. In case that *this* shell widget object is a dialog shell this data member contains *NULL*.
The data member has to be used as the parent for all *XOL* widget objects that should be managed as a child of this shell.

Values: Main window widget object

8.17.3 Usable Methods and Operators

XolWmShellC (XolWmShellC* pParent, char *pName,
WidgetClass pClass = topLevelShellWidgetClass);
XolWmShellC (Widget pParent, char *pName,
WidgetClass pClass = topLevelShellWidgetClass);

Description: This method constructs a shell widget object by using the constructor of the basis class *XolWidgetC*.

Parameters:
- *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle, normally of the application object
- *pName* - Name of the shell widget that is also displayed as the title by default
- *pClass* - Class of the shell widget, can be
  - *TopLevelSh* - non-transient top-level shell, can be iconified independently from the application shell
  - *TransientSh* - transient top-level shell, cannot be iconified independently from the application shell
  - *ApplicationSh* - application top-level shell
  - *DialogSh* - dialog top-level shell

Results: None

virtual int CreateRel (bool pScroll = FALSE, int xPos = 30, int yPos = 30
, int xSize= 0 , int ySize= 0);
virtual int Create (bool pScroll = FALSE, int xPos = 0 , int yPos = 0
, int xSize= 0 , int ySize= 0);

Description: This methods creates the underlying *OSF/Motif* top-level widget. The top-level widget is created with specified or default size and position. The position can be specified either relative or absolute with respect to its parent widget object.
For shell widget objects that are not dialog shell widget objects the one possible child object, the client area widget (*OSF/Motif* class *MainWindow*) is created automatically by this method, if the client area widget pointer is not set by the user. The pointer to this client area widget object is stored in the data member *mClientArea*. It has to be used as the parent widget object for the creation of all children of *this* shell widget object. To pop up the created shell widget object on display, it has to be managed by using the method *Manage* explicitly.

Parameters: *pScroll* - Scrolling policy for the child main window widget object.
TRUE - the main window is capable to scroll its contents, but only if it is needed dependent on the geometry
FALSE - the main window is not capable to scroll its contents
xPos - shell widget x position (0 - not specified)
yPos - shell widget y position (0 - not specified)
xSize - shell width
0 && (*pScroll==TRUE*) - XOL_DEF_XSIZE
0 && (*pScroll==FALSE*) - automatic x size
ySize - shell height
0 && (*pScroll==TRUE*) - XOL_DEF_YSIZE
0 && (*pScroll==FALSE*) - automatic y size

Results: None

virtual void Manage();

Description: This method overrides the virtual method *XolWidgetC::Manage*. It realizes the popping up of *this* shell widget object on display.

Parameters: None

Results: None

virtual void SetTitle (char* pTitle);

Description: This method sets the title of the shell widget object. The title is displayed in the window manager head line of the shell widget.

Parameters: *pTitle* - Shell title

Results: None

virtual void SetIconLabel (char* pLabel);

Description: This method sets the abbreviation that labels the iconified shell widget object.

Parameters: *pLabel* - Shell icon label

Results: None

virtual void SetIcon (XolPixmapC* pIcon);
virtual void SetIcon (Pixmap pIcon);

Description: These methods set the shell widget icon.

Parameters: *pIcon* - Icon pixmap or pixmap widget object

Results: None

virtual void SetMinSize (long dx, long dy);
virtual void SetMaxSize (long dx, long dy);

Description: These methods set the minimum/maximum size of the shell widget.

Parameters: *dx* - Shell minimum/maximum width
dx - Shell minimum/maximum height

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolWidgetC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolWmShellC*)

8.18 Class XolTopLevelShellC

This class realizes *top-level shell widget* objects. Top-level shell widgets are all widgets that use top-level windows with window manager decorations independently from their parent widgets. The only one child of the top-level shell widget object is created
automatically. It is an object of the *OSF/Motif* widget class *MainWindow*. The class is derived from the class *XolWmShellC*. Since the basis class *XolWmShellC* defines already the functionality for all kinds of shell widget objects, this class defines only a more specialised constructor.

8.18.1 Class Declaration

```
#include <XOL/Shell.h>

class XolTopLevelShellC : public XolWmShellC
{
  public:

    XolTopLevelShellC (XolWmShellC *pParent, char *pName)
                     : XolWmShellC(pParent, pName
                                   , topLevelShellWidgetClass) {}
    XolTopLevelShellC (Widget pParent, char *pName)
                     : XolWmShellC(pParent, pName
                                   , topLevelShellWidgetClass) {}
    virtual char *GetClass() { return "XolTopLevelShellC"; }
};
```

8.18.2 Important Data Members

The class defines no further data members.

8.18.3 Usable Methods and Operators

XolTopLevelShellC (XolWmShellC *pParent, char *pName);
XolTopLevelShellC (Widget pParent, char *pName);

Description: The method constructs a top-level shell widget object.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle, normally of the application object
pName - Name of the shell widget that is also displayed as the title by default

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolWmShellC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolTopLevelShellC*)

8.19 Class XolTransientShellC

This class realizes *transient shell widget* objects. Transient shell widgets are similar to top-level shell widgets. However they widgets cannot be iconified independently from their parent widgets. This means that a transient shell widget is iconified when the associated application shell widget is iconified. The only one child of the transient shell widget object is created automatically. It is an object of the *OSF/Motif* widget class *MainWindow*. The class is derived from the class *XolWmShellC*. Since the basis class *XolWmShellC* defines already the functionality for all kinds of shell widget objects, this class defines only a more specialised constructor.

8.19.1 Class Declaration

```
#include <XOL/Shell.h>

class XolTransientShellC : public XolWmShellC
{
  public:

    XolTransientShellC (XolWmShellC *pParent, char *pName)
                      : XolWmShellC(pParent, pName
                                              , transientShellWidgetClass) {}
    XolTransientShellC (Widget pParent, char *pName)
                      : XolWmShellC(pParent, pName
                                              , transientShellWidgetClass) {}
    virtual char *GetClass() { return "XolTransientShellC"; }
};
```

8.19.2 Important Data Members

The class defines no further data members.

8.19.3 Usable Methods and Operators

XolTransientShellC (XolWmShellC *pParent, char *pName);
XolTransientShellC (Widget pParent, char *pName);

Description: The method constructs a transient shell widget object.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle, normally of the application object
pName - Name of the shell widget that is also displayed as the title by default

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolWmShellC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolTransientShellC*)

8.20 Class XolDialogShellC

This class realizes *dialog shell widget* objects. Dialog shell widgets are dialog shells without any window manager decorations. They provide dialog specific widgets with a shell that is displayed over the top of its parent widget. The only one child of the dialog shell widget object is not created automatically. It has to be set to a dialog specific widget object created by the user. Therefore, th data member contains *NULL* after the creation of *this* dialog shell widget object.
The class is derived from the class *XolWmShellC*. Since the basis class *XolWmShellC* defines already the functionality for all kinds of shell widget objects, this class defines only a more specialised constructor.

8.20.1 Class Declaration

```
#include <XOL/Shell.h>

class XolDialogShellC : public XolWmShellC
{
  public:

    XolDialogShellC (XolWmShellC *pParent, char *pName)
                   : XolWmShellC(pParent, pName
                                 , xmDialogShellWidgetClass) {}
    XolDialogShellC (Widget pParent, char *pName)
                   : XolWmShellC(pParent, pName
                                 , xmDialogShellWidgetClass) {}
    virtual char *GetClass() { return "XolDialogShellC"; }
};
```

8.20.2 Important Data Members

The class defines no further data members.

8.20.3 Usable Methods and Operators

XolDialogShellC (XolWmShellC *pParent, char *pName);
XolDialogShellC (Widget pParent, char *pName);

Description: The method constructs a dialog shell widget object.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle
pName - Name of the dialog shell widget

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolWmShellC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolDialogShellC*)

8.21 Class XolPixmapC

This class realizes the different forms of *X Windows* pixmaps (pixel maps). In the *X Windows* system a pixmap is a chunk of non-visible memory with a given size and colour depth. Normally, the colour depth is determined by the colour depth of a given drawable, i.e. a given widget or the screen itself. To make the pixmap visible, its pixels have to be copied explicitly from the non-visible pixmap memory to the screen memory. This is normally done by the function *Xlib* function *XCopyArea*. Since the *X Windows* pixmaps are drawables, it is possible to draw into the pixmaps directly. However, pixmaps can only be used for screen and display for which they were defined.

Pixmaps are used as widget background fill patterns or simply as an icon. The simplest form of a pixmap is the *X Windows bitmap*. A bitmap is a special kind of pixmaps that uses a colour depth of only one bit. The more high sophisticated form of pixmaps are the coloured pixmaps. They have to be specified in a file according to the *XPM (X Pixel Map)* format [LH94]. The format allows also the masking of the pixmaps. The handling of pixmaps is implemented by a caching mechanism in the *X Windows* system. After the caching of a pixmap only a pixmap handle is used to access it.
Whereas the *Xlib* defines only functionality for handling bitmaps, the class *XolPixmapC* encapsulates functionality for pixmaps that can have a colour depth of more than one bit. For this purpose, it uses the *XPM* library [LH94]. The *XPM* library specifies the format for coloured pixmap definitions and provide the functionality to transform such a pixmap definition to a *X Windows* pixmap. The class *XolPixmapC* encapsulates this functionality and allows the handling of bitmaps and pixmaps within a single class. The class was derived from the class *SstObjectC* [Sch96].

8.21.1 Class Declaration

```
#include <XOL/Pixmap.h>

class XolPixmapC : public SstObjectC
{
    Screen  *mScreen;
    Display *mDisplay;

  public:

    Pixmap mPixmap;
    Pixmap mShapeMask;

    int mWidth;
    int mHeight;

    // Constructors

    XolPixmapC(XolWidgetC* pParent, char *pName);
    XolPixmapC(Widget pParent, char *pName);

    XolPixmapC(XolWidgetC* pParent, int pWidth, int pHeight);
    XolPixmapC(Widget pParent, int pWidth, int pHeight);

    // Destructor

    virtual ~XolPixmapC();

    // Miscellaneous

    virtual char *GetClass() { return "XolPixmapC"; }
};
```

8.21.2 Important Data Members

Screen *mScreen;

Description: This data member holds the *X Windows* screen for which the pixmap was created. For the determination of the pixmaps *X Windows* screen the *Xt* widget handle of the parent widget is used.

Values: Pointer to the *X Windows* screen descriptor

Display *mDisplay;

Description: This data member holds the *X Windows* display for which the pixmap was created. For the determination of the pixmaps *X Windows* display the *Xt* widget handle of the parent widget is used.

Values: Pointer to the *X Windows* display descriptor

Pixmap mPixmap;

Description: This data member hold after the creation of a pixmap its handle.

Values: Pixmap handle

Pixmap mShapeMask;

Description: This data member is a pixmap that holds a shaping mask. The shaping mask can be specified within a file according to the *XPM* file format. This data member is only used for coloured pixmaps that are specified within a file according to the *XPM* specification.

Values: Pixmap handle

int mWidth;
int mHeight;

Description: These data members hold the width/height of this pixmap object that is created with specific width/height. Otherwise these data members contain 0.

Values: Width/Height value

8.21.3 Usable Methods and Operators

XolPixmapC (XolWidgetC* pParent, char *pName);
XolPixmapC (Widget pParent, char *pName);

Description: These methods construct a pixmap object from its specification given in bitmap or *XPM* format.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle; The *pParent* widget object has not to be really a the parent of *this* pixmap, since the pixmap is handled independently from a widget. The parent widget object is only used for the determination of the according *X Windows* screen and display. But, the parent widget object MUST BE CREATED already, before the pixmap object can be constructed!

pName - Name of the file that specifies the pixmap in bitmap or *XPM* format.

Results: None

XolPixmapC (XolWidgetC* pParent, int pWidth, int pHeight);
XolPixmapC (Widget pParent, int pWidth, int pHeight);

Description: These methods construct an empty pixmap object with given width and height. Such pixmap objects can be used for drawing directly into the pixmap by using graphic primitives.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle; The *pParent* widget object has not to be really a the parent of *this* pixmap, since the pixmap is handled independently from a widget. The parent widget object is only used for the determination of the according *X Windows* screen and display. But, the parent widget object MUST BE CREATED already, before the pixmap object can be constructed!

pWidth - Width of the pixmap in pixel
pHeight - Height of the pixmap in pixel

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *SstObjectC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolPixmapC*)

8.22 Class XolLabelC

This class encapsulates the *OSF/Motif* label widget class. Labels provide the ability to display static, uneditable text and pixmaps. A label is useful for displaying a message, title, or description. It can display either text or a pixmap. The type of displayed information, i.e. text or pixmap, can be changed during the label objects lifetime. The *OSF/Motif* class label is the superclass for all kinds of buttons.
The class *XolLabelC* is derived from class *XolPrimitiveC* described in section 8.3. It is the superclass for more specialised widget object classes, see section 8.23 and 8.24.

8.22.1 Class Declaration

```
#include <XOL/Label.h>

class XolLabelC : public XolPrimitiveC
{
    XolPixmapC *mPixmap;
    byte        mType;

  public:

    // Constructors

    XolLabelC(XolWidgetC* pParent, char *pName, byte pType = XmSTRING
                              , WidgetClass pClass = xmLabelWidgetClass);
    XolLabelC(Widget pParent, char *pName, byte pType = XmSTRING
                         , WidgetClass pClass = xmLabelWidgetClass);

    // Methods that create the underlying Motif widget (Text/Pixmap)

    virtual int Create (bool pManaged = TRUE, char *pText = NULL
                                         , int xPos = -1, int yPos = -1
                                         , int xSize= -1, int ySize= -1);
    virtual int Create (bool pManaged, XolPixmapC *pPixmap
                                  , int xPos = -1, int yPos = -1
                                  , int xSize= -1, int ySize= -1);
    // Miscellaneous methods

    void Set(XolPixmapC* pText);
    void Set(char*       pPixmap);

    void ReAlign(byte pAlign = 0);

    virtual char* GetClass () { return "XolLabelC"; }
};
```

8.22.2 Important Data Members

XolPixmapC *mPixmap;

Description: This data member holds in the case of a pixmap label the reference to the pixmap object. Otherwise this data member is *NULL*.

Values: Pointer to a pixmap object

byte mType;

Description: This data member holds the current type of the label widget object.

Values: XmSTRING | XmPIXMAP

8.22.3 Usable Methods and Operators

XolLabelC(XolWidgetC* pParent, char *pName, byte pType = XmSTRING
, WidgetClass pClass = xmLabelWidgetClass);
XolLabelC(Widget pParent, char *pName, byte pType = XmSTRING
, WidgetClass pClass = xmLabelWidgetClass);

Description: These methods construct a label widget object. The label widget object can be used for displaying text or pixmaps. The type of the label widget object have to be specified during its construction.

Parameters:
pParent - Pointer to the parent widget object or the parent *Xt* widget handle;
pType - Type of label widget object
pClass - Class of label widget object (normally the class is *LabelWidget*, but also derived *OSF/Motif* widget classes can be used)

Results: None

virtual int Create (bool pManaged = TRUE, char *pText = NULL
, int xPos = -1, int yPos = -1
, int xSize = -1, int ySize= -1);
virtual int Create (bool pManaged, XolPixmapC *pPixmap
, int xPos = -1, int yPos = -1
, int xSize = -1, int ySize = -1);

Description: With these methods the underlying *OSF/Motif* widget is created. If the type of the label widget object is *XmPIXMAP* the parameter *pText* is interpreted as the pixmap filename. Otherwise the parameter is interpreted as normal text. The contents of the label is aligned by default in its centre.

Parameters:
pManaged - Determines whether the created widget should be managed automatically by its parent during the creation
TRUE - the widget is managed automatically (default)
FALSE - the widget is not managed automatically
pText - Text of the text label widget object or the pixmap file name (NULL -> name of the label widget object name is used as label text)
pPixmap - Pointer to the pixmap object that is displayed in the label
xPos - Label x position within its parent widget object (-1 -> not specified)
yPos - Label y position within its parent widget object (-1 -> not specified)
xSize - Label x size (-1 -> not specified)
ySize - Label y size (-1 -> not specified)

Results: None

void Set (XolPixmapC* pText);
void Set (char* pPixmap);

Description: With these methods the type of the label as well the displayed contents can be changed

Parameters:
pText - Type of *this* label object is changed to text and the given text *pText* is displayed in it
pPixmap - Type of *this* label object is changed to pixmap and the given pixmap *pPixmap* is displayed in it

Results: None

void ReAlign(byte pAlign = 0);

Description: This method realigns the pixmap or string of the label widget object.

Parameters: *pAlign* - Determines the new alignment
XmALIGNMENT_BEGINNING - the pixmap or text is aligned at the left side of the label

XmALIGNMENT_CENTER	- the pixmap or text is aligned in the centre of the label
XmALIGNMENT_END	- the pixmap or text is aligned at the right side of the label

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolPrimitiveC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolLabelC*)

8.23 Class XolPushButtonC

This class encapsulates the *OSF/Motif* push button widget class. A push button is a basic control that performs some action when the user activates it. It performs some action determined by the application. When a push button is armed, or ready to activated, it changes its appearance. A push button widget can display text or pixmap. The type of displayed information, i.e. text or pixmap, can be change during the push button objects lifetime.
The class *XolPushButtonC* is derived from class *XolLabelC* described in section 8.22. To handle possible events, several virtual dummy methods are defined. These methods are called by the callback handler that is registered automatically for the *OSF/Motif* events *XmCR_ACTIVATE*, *XmCR_ARM*, and *XmCR_DISARM*. To handle these events by own callback methods, either the dummy methods have to be overwritten within a derived class or own callback methods have to be registered for the events. To handle these events by own callback methods, either the dummy methods have to be overwritten within a derived class or own callback methods have to be registered for the events.

8.23.1 Class Declaration

```
#include <XOL/PushB.h>

class XolPushButtonC : public XolLabelC
{
    static void CallbackHandler(Widget, XolPushButtonC*, XolPointerT);

  public:

    // Constructors

    XolPushButtonC (XolWidgetC* pParent, char *pName, byte pType = XmSTRING)
                 : XolLabelC(pParent, pName, pType
                                          , xmPushButtonWidgetClass) {}

    XolPushButtonC (Widget pParent, char *pName, byte pType = XmSTRING)
                 : XolLabelC(pParent, pName, pType
                                          , xmPushButtonWidgetClass) {}

    // Create methods

    virtual int Create(bool pManaged =TRUE, char *pText = NULL
                                        , int xPos = -1, int yPos = -1
                                        , int xSize= -1, int ySize= -1);
    virtual int Create(bool pManaged, XolPixmapC *pPixmap
                                 , int xPos = -1, int yPos = -1
                                 , int xSize= -1, int ySize= -1);
```

```
    // Dummy callback methods

    virtual void ArmEvent      ()    { return; }  // mouse comes into button
    virtual void DisarmEvent   ()    { return; }  // mouse goes out from button
    virtual void ActivateEvent (int) { return; }  // multi click

    // Miscellaneous methods

    virtual char *GetClass ()   { return "XolPushButtonC"; }
};
```

8.23.2 Important Data Members

The class defines no further data members.

8.23.3 Usable Methods and Operators

XolPushButtonC (XolWidgetC* pParent, char *pName, byte pType = XmSTRING);
XolPushButtonC (Widget pParent, char *pName, byte pType = XmSTRING);

Description: These methods construct a push button widget object. A push button widget object has a label that can be text or a pixmap. The type of the label has to be specified.

Parameters:
pParent - Pointer to the parent widget object or the parent *Xt* widget handle;
pName - Name of push button widget object
pType - Type of the push button widget object label

Results: None

virtual int Create (bool pManaged = TRUE, char *pText = NULL
, int xPos = -1, int yPos = -1
, int xSize = -1, int ySize= -1);
virtual int Create (bool pManaged, XolPixmapC *pPixmap
, int xPos = -1, int yPos = -1
, int xSize = -1, int ySize = -1);

Description: These methods create the underlying *OSF/Motif* widget is created. If the type of the label is *XmPIXMAP* the parameter *pText* is interpreted as the pixmap filename. Otherwise the parameter is interpreted as normal text.

Parameters:
pManaged - Determines whether the created widget should be managed automatically by its parent during the creation
TRUE - the widget is managed automatically (default)
FALSE - the widget is not managed automatically
pText - Text of the text label widget object (NULL -> name of the label widget object name is used as label text)
pPixmap - Pointer to the pixmap object that is displayed in the label
xPos - Label x position within its parent widget object (-1 -> not specified)
yPos - Label y position within its parent widget object (-1 -> not specified)
xSize - Label x size (-1 -> not specified)
ySize - Label y size (-1 -> not specified)

Results: None

virtual void ActivateEvent (int);
virtual void ArmEvent ();
virtual void DisarmEvent ();

Description: These methods are dummy methods that are called by the callback handler of *this* push button widget object. For each possible callback event defined by the *OSF/Motif* widget class *PushButton*, one dummy method is defined. To handle the callbacks by own callback method, either these methods have to be overwritten by a derived class or the own callback methods have to be registered for the callback events by using the method *XolWidgetC::AddCallback*.

Parameters: The parameter of the method *ActivateEvent* delivers the number of multi-clicks that caused the callback.

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolLabelC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolPushButtonC*)

8.24 Class XolToggleButtonC

This class encapsulates the *OSF/Motif* push button widget class. A toggle button is like a push button a basic control that performs some action when the user activates it. A toggle button widget can display text or pixmap. The type of displayed information, i.e. text or pixmap, can be change during the toggle button objects lifetime. Toggle buttons have one of two states. They are either on or off. They can appear in menus or in row column working areas, including radio boxes and check boxes. In a radio box only one toggle button at a time can be on whereas in a check box more than one toggle button can be active. Toggle buttons have indicators with distinctive shapes to distinguish whether or not more than one button at a time can be set. These indicators are optional since the state changes can be indicate also by the push button text or pixmap. The behaviour of toggle buttons is controlled by parent widget. Dependent on the type of the parent widget object the toggle button is defined as a part of a radio box or a check box.
The class *XolToggleButtonC* is derived from class *XolLabelC* described in section 8.22. To handle possible events, several virtual dummy methods are defined. These methods are called by the callback handler that is registered automatically for the *OSF/Motif* events *XmCR_VALUE_CHANGED*, *XmCR_ARM*, and *XmCR_DISARM*. To handle these events by own callback methods, either the callback methods have to be overwritten within a derived class or own callback methods have to be registered for the events.

8.24.1 Class Declaration

```
#include <XOL/ToggleB.h>

class XolToggleButtonC : public XolLabelC
{
    static void CallbackHandler(Widget, XolToggleButtonC*, XolPointerT);

  public:

    // Constructors

    XolToggleButtonC (XolWidgetC* pParent, char *pName, byte pType = XmSTRING)
                    : XolLabelC(pParent, pName, pType
                                               , xmToggleButtonWidgetClass) {}

    XolToggleButtonC (Widget pParent, char *pName, byte pType = XmSTRING)
                    : XolLabelC(pParent, pName, pType
                                               , xmToggleButtonWidgetClass) {}

    // Create methods
```

```
    virtual int Create(bool pManaged =TRUE, char *pText = NULL
                                       , int xPos = -1, int yPos = -1
                                       , int xSize= -1, int ySize= -1);
    virtual int Create(bool pManaged, XolPixmapC *pPixmap
                                 , int xPos = -1, int yPos = -1
                                 , int xSize= -1, int ySize= -1);

    // Dummy callback methods

    virtual void ArmEvent          ()     { return; }  // mouse comes in button
    virtual void DisarmEvent       ()     { return; }  // mouse goes out button
    virtual void StateChangedEvent (bool) { return; }  // new state

    // Miscellaneous methods

    bool GetState();
    void SetState(bool pState);

    virtual char *GetClass()   { return "XOL_ToggleButtonC"; }
};
```

8.24.2 Important Data Members

The class defines no further data members.

8.24.3 Usable Methods and Operators

XolToggleButtonC (XolWidgetC* pParent, char *pName, byte pType = XmSTRING);
XolToggleButtonC (Widget pParent, char *pName, byte pType = XmSTRING);

Description: These methods construct a toggle button widget object. A toggle button widget object has a label that can be text or a pixmap. The type of the label has to be specified.

Parameters:
- *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle;
- *pName* - Name of toggle button widget object
- *pType* - Type of the push button widget object label

Results: None

virtual int Create (bool pManaged = TRUE, char *pText = NULL
, int xPos = -1, int yPos = -1
, int xSize = -1, int ySize= -1);
virtual int Create (bool pManaged, XolPixmapC *pPixmap
, int xPos = -1, int yPos = -1
, int xSize = -1, int ySize = -1);

Description: These methods create the underlying *OSF/Motif* widget is created. If the type of the label is *XmPIXMAP* the parameter *pText* is interpreted as the pixmap filename. Otherwise the parameter is interpreted as normal text.

Parameters:
- *pManaged* - Determines whether the created widget should be managed automatically by its parent during the creation
  - *TRUE* - the widget is managed automatically (default)
  - *FALSE* - the widget is not managed automatically
- *pText* - Text of the text label widget object (NULL -> name of the label widget object name is used as label text)
- *pPixmap* - Pointer to the pixmap object that is displayed in the label
- *xPos* - Label x position within its parent widget object (-1 -> not specified)
- *yPos* - Label y position within its parent widget object (-1 -> not specified)
- *xSize* - Label x size (-1 -> not specified)

	ySize	- Label y size (-1 -> not specified)
Results:	None	

virtual void StateChangedEvent(int);
virtual void ArmEvent ();
virtual void DisarmEvent ();

Description: These methods are dummy methods that are called by the callback handler of *this* toggle button widget object. For each possible callback event defined by the *OSF/Motif* widget class *ToggleButton*, one dummy method is defined. To handle the callbacks by own callback method, either these methods have to be overwritten by a derived class or the own callback methods have to be registered for the callback events by using the method *XolWidgetC::AddCallback*.

Parameters: The parameter of the method *ActivateEvent* delivers the number of multi-clicks that caused the callback.

Results: None

bool GetState();
void SetState(bool pState);

Description: These methods get/set the state of the toggle button widget object.

Parameters: *pManaged* - Determines whether the toggle button is active or inactive

TRUE - the toggle button is active
FALSE - the toggle button is inactive

Results: State of toggle button

TRUE - the toggle button is active
FALSE - the toggle button is inactive

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolLabelC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolToggleButtonC*)

8.25 Class XolTextWidgetC

The class *XolTextWidgetC* realizes scrolled text widget objects as well one line text field widget objects. These text widget objects can be editable or not. The scrolled text widget object consist of two widget objects, the text widget object encapsulated by a scrolled widget object. The text widget object that provides the text operations. The scroll widget object is used for managing the text widget object. While the scroll widget is encapsulated by the object itself, the text widget object is held in the data element *mText*. The encapsulated text widget is created automatically when *this* scrolled widget object is created by using the method *Create*. Before the scrolled widget object can be created the parent *Xt* widget handle has to be defined already, i.e., the create method of the parent widget object has to be executed already. The one line text widget object consist of only one widget object. In this case the data member *mText* is referencing *this* widget object.
The class *XolTextWidgetC* is derived from class *XolManagerC* described in section 8.2. To handle possible events, several virtual dummy methods are defined. These methods are called by the callback handler. The callback handle is registered automatically for the *OSF/Motif* events *XmCR_VALUE_CHANGED*, *XmCR_MODIFYING_TEXT_VALUE*, and *XmCR_MOVING_INSERT_CURSOR*. To handle these events by own callback methods, either the dummy methods have to be overwritten within a derived class or own callback methods have to be registered for these events. However, some of the callback methods have really functionality and maybe have to be called from out the own callback methods.

8.25.1 Class Declaration

```
#include <XOL/TextW.h>

class XolTextWidgetC : public XolManagerC
{
    static void CallbackHandler(Widget, XolTextWidgetC*, XolPointerT);

  public:

    XolPrimitiveC* mText;     // managed text widget
    bool           mChanged;  // changing state of the text widget object

    // Constructors

    XolTextWidgetC (Widget pParent, char *pName);
    XolTextWidgetC (XolWidgetC *pParent, char *pName);

    // Creation methods

    virtual int Create(bool pManaged = TRUE, bool pEditable = TRUE
                                       , int  pRows = 25
                                       , int  pCols = 80);

    // Miscellaneous methods

    void printf(char *pFormat, ...);

    void  SetText (char *pText);
    char* GetText ();

    bool GetCursorPos (short &x, short &y);
    bool SetCursorPos (short  x, short  y);

    virtual char *GetClass()   { return "XolTextWidgetC"; }

    // Dummy callback methods

    virtual void TextChanged  (XEvent*)        { return; }
    virtual void CursorMoved  (short, short)   { return; }
    virtual void CursorMoved  (XmTextPosition) { return; }

    // Callback methods

    virtual void MotionVerify (XmTextVerifyCallbackStruct *pData);
    virtual void ModifyVerify (XmTextVerifyCallbackStruct *pData);
};
```

8.25.2 Important Data Members

XolPrimitiveC* mText;

Description:	In case of scrolled text widget objects this data member holds a pointer to the encapsulated text widget object. All text operations of the scrolled text widget object have to executed on the text widget object referenced by this data member. In case of one line text field widget objects the data member holds a reference to *this* widget object itself.
Values:	Pointer to text widget object

bool mChanged;

Description: This data member indicates whether the contents of the text widget object was changed. For this purpose the data member is set within the callback handler during the processing of the *OSF/Motif* event *XmCR_VALUE_CHANGED*.

Values: TRUE | FALSE

8.25.3 Usable Methods and Operators

XolTextWidgetC (XolWidgetC *pParent, char *pName);
XolTextWidgetC (Widget pParent, char *pName);

Description: These methods construct a scrolled text widget object. After the construction the encapsulated text widget object is not existing.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle;
pName - Name of the scrolled text widget object

Results: None

virtual int Create(bool pManaged = TRUE, bool pEditable = TRUE
, int pRows = 25
, int pCols = 80);

Description: With this method the scrolled text widget object or the one line text widget object is created, i.e., the underlying *OSF/Motif* widgets are created. The type of the text widget object is defined by the number of line given by the parameter *pRows*.
In case of scrolled text widget objects the encapsulated text widget object is constructed and the underlying *OSF/Motif* widget is created automatically. The reference to the text widget object is stored in the data member *mText*. The name of the text widget object is set to the name of *this* encapsulating scrolled widget object. The name of *this* scrolled widget object is changed to the name itself and an appended string *SW*.
In case of one line text widget object an underlying *OSF/Motif* text widget with one line is created. The resulting widget handle is stored in the data member *XolWidgetC::mXtWidget* of *this* widget object. Additionally, the *OSF/Motif* widget class of *this* widget object is changed to class *TextField*. The data member *mText* is set to to *this* widget object itself.
Before this method can be called, the *Xt* widget handle of the parent widget object have to be defined, i.e., the method *Create* have to be already executed for the parent widget object.

Parameters: *pManaged* - Determines whether the created widget should be managed automatically by its parent during the creation
TRUE - the widget is managed automatically (default)
FALSE - the widget is not managed automatically
pEditable - Determines whether the text widget object is editable
TRUE - the text widget object is editable
FALSE - the text widget object is not editable
pRows - Number or rows (1 -> *this* widget object is a one line text widget object)
pCols - Number of columns

Results: None

void printf(char *pFormat, ...);

Description: This methods print out the given text to the text widget object at the current insertion cursor position.

Parameters: *pFormat* - Format string (the same as for the standard *C* function
... - List of variables and constants that should be print out

Results: None

char* GetText ();
void SetText (char *pText);

Description: These methods set/get the contents of the text widget object.

Parameters: *pText* - Text in case of the method *SetText*

Results: Text in case of the method *GetText*

bool GetCursorPos (short &x, short &y);
bool SetCursorPos (short x, short y);

Description: These methods get/set the current insertion cursor position

Parameters: *x* - x position of the cursor
y - y position of the cursor

Results: None

virtual void CursorMoved (XmTextPosition);
virtual void CursorMoved (short, short);
virtual void TextChanged (XEvent*);

Description: These methods are dummy methods that are called by the callback handler when the contents of the text widget object was changed or the insertion cursor was moved. These changings are indicated by the Motif events *XmCR_VALUE_CHANGED* and *XmCR_MOVING_INSERT_CURSOR*, respectively. To handle the callbacks by own callback method, either these methods have to be overwritten by a derived class or the own callback methods have to be registered for the callback events by using the method *XolWidgetC::AddCallback*.

Parameters: The parameter of the method *TextChanged* delivers the whole *Xlib* event structure for handling by own methods. The parameters of the methods *CursorMoved* deliver the new insertion cursor position given in the *OSF/Motif* positioning format and the line/column format, respectively.

Results: None

virtual void MotionVerify (XmTextVerifyCallbackStruct *pData);
virtual void ModifyVerify (XmTextVerifyCallbackStruct *pData);

Description: These methods are default callback methods for the *OSF/Motif* events *XmCR_MODIFYING_TEXT_VALUE* and *XmCR_MOVING_INSERT_CURSOR*. By producing these events, *OSF/Motif* allows the user to verify whether cursor motion or text modifying should be executed or not, before it does the changes. To handle these callbacks by own callback methods, either these methods have to be overwritten within a derived class or the own callback methods have to be registered for the callback events by using the method *XolWidgetC::AddCallback*.

Parameters: *pData* - Pointer to the callback data structure of type *XmTextVerifyCallbackStruct*

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolManagerC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolTextWidgetC*)

8.26 Class XolSeparatorC

This class encapsulates the *OSF/Motif* widget class *Separator*. A separator widget separates controls or group of controls, i.e., widgets of the *OSF/Motif* class *Primitive* or a derived class. A separator widget usually appears as a line. The line can have horizontal or vertical orientation. It supports several styles of line drawing.
The class is derived from the widget object class *XolPrimitiveC* described in section 8.3. The parent widget object have to from *OSF/Motif* widget class *RowColumn* or a derived *OSF/Motif* widget class.

8.26.1 Class Declaration

```
#include <XOL/Sep.h>

class XolSeparatorC : public XolPrimitiveC
{

  public:

    // Constructor

    XolSeparatorC (XolWidgetC* pParent, char *pName)
                 : XolPrimitiveC(pParent, pName, xmSeparatorWidgetClass) {}
    XolSeparatorC(Widget pParent, char *pName)
                 : XolPrimitiveC(pParent, pName, xmSeparatorWidgetClass) {}

    // Create methods

    virtual int Create(bool pManaged = TRUE, byte pOrient = XmNO_ORIENTATION);

    // Miscellaneous methods

    virtual void SetOrientation (byte pOrient = XmNO_ORIENTATION);
    virtual byte GetOrientation ();

    virtual char *GetClass()   { return "XolSeparatorC"; }
};
```

8.26.2 Important Data Members

The class defines no further data members.

8.26.3 Usable Methods and Operators

XolSeparatorC (XolWidgetC* pParent, char *pName);
XolSeparatorC (Widget pParent, char *pName);

Description: These methods constructs a separator widget object.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle;
pName - Name of the separator widget object

Results: None

virtual int Create(bool pManaged= TRUE,
byte pOrient = XmNO_ORIENTATION);

Description: This method creates the underlying *OSF/Motif* separator widget with the orientation given by the parameter *pOrient*. If this parameter delivers *XmNO_ORIENTATION*, the separator takes the orientation of its row column parent widget.

Parameters: *pManaged* - Determines whether the created widget should be managed automatically by its parent during the creation

TRUE - the widget is managed automatically (default)
FALSE - the widget is not managed automatically

pOrient - Separator orientation

XmHORIZONTAL - horizontal orientation
XmVERTICAL - vertical orientation
XmNO_ORIENTATION - orientation is taken from the row column parent widget object

Results: None

virtual void SetOrientation (byte pOrient = XmNO_ORIENTATION);

Description: This method set the orientation of *this* separator widget object. If the orientation parameter *pOrient* delivers *XmNO_ORIENTATION*, the separator orientation is toggled.

Parameters: *pOrient* - Separator orientation

XmHORIZONTAL	-	horizontal orientation
XmVERTICAL	-	vertical orientation
XmNO_ORIENTATION	-	orientation is toggled

Results: None

virtual byte GetOrientation ();

Description: This method gets the orientation of *this* separator widget object.

Parameters: None

Results: *XmHORIZONTAL* | *XmVERTICAL*

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolPrimitiveC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolSeparatorC*)

8.27 Class XolRadioBoxC

This class realizes radio box widget objects based on the *OSF/Motif* widget class *RowColumn*. Radio boxes are collection of toggle buttons, that are also called radio buttons. These toggle buttons can have diamond shaped active indicators. The specific property of a radio box is that only one toggle button can be active at a time. Radio boxes are usually implemented as working areas.
The class uses the *OSF/Motif* widget class *RowColumn* as well the *OSF/Motif* functionality for realizing radio boxes to realize an easy to use radio box widget object. The class is derived from the class *XolManagerC* described in section 8.2. Child toggle button widget object are controlled by radio box widget objects as radio button automatically.

8.27.1 Class Declaration

```
#include <XOL/RadioBox.h>

class XolRadioBoxC : public XolManagerC
{

  public:

    // Constructors

    XolRadioBoxC (XolWidgetC *pParent, char *pName)
              : XolManagerC(pParent, pName, xmRowColumnWidgetClass) {}

    XolRadioBoxC (Widget pParent, char *pName)
              : XolManagerC(pParent, pName, xmRowColumnWidgetClass) {}

    // Create methods

    virtual int Create(bool pManaged = TRUE, byte  pDirect = XmVERTICAL
```

```
                                        , short pColNum= 1);

    // Miscellaneous methods

    virtual char *GetClass()   { return "XolRadioBoxC"; }
};
```

8.27.2 Important Data Members

The class defines no further data members.

8.27.3 Usable Methods and Operators

XolRadioBoxC (XolWidgetC *pParent, char *pName);
XolRadioBoxC (Widget pParent, char *pName);

Description: These methods constructs a radio box widget object.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle;
pName - Name of the radio box widget object

Results: None

virtual int Create(bool pManaged = TRUE, byte Direct = XmVERTICAL
, short pColNum = 1);

Description: This method creates the underlying *OSF/Motif* row column widget with the orientation given by the parameter *pDirect*.

Parameters: *pManaged* - Determines whether the created widget should be managed automatically by its parent during the creation
TRUE - the widget is managed automatically (default)
FALSE - the widget is not managed automatically
pDirect - Radio box orientation (by default XmVERTICAL)
XmHORIZONTAL- radio buttons are arranged horizontally
XmVERTICAL - radio buttons are arranged vertically
pColNum - Number of radio button columns if the orientation is vertical or number of radio button rows if the orientation is horizontal

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolManagerC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolRadioBoxC*)

8.28 Class XolCheckBoxC

This class realizes check box widget objects based on the *OSF/Motif* widget class *RowColumn*. Check boxes are collection of toggle buttons, that are also called check buttons. These toggle buttons can have square shaped active indicators. The specific property of a check box is that more than one toggle button can be active at a time. Check boxes are usually implemented as working areas.
The class uses the *OSF/Motif* widget class *RowColumn* as well the *OSF/Motif* functionality for realizing check boxes to realize an easy to use check box widget object. The class is derived from the class *XolManagerC* described in section 8.2. Child toggle button widget object are controlled by check box widget objects as check button automatically.

8.28.1 Class Declaration

```
#include <XOL/CheckBox.h>

class XolCheckBoxC : public XolManagerC
{

  public:

    // Constructors

    XolCheckBoxC (XolWidgetC *pParent, char *pName)
                : XolManagerC(pParent, pName, xmRowColumnWidgetClass) {}

    XolCheckBoxC (Widget pParent, char *pName)
                : XolManagerC(pParent, pName, xmRowColumnWidgetClass) {}

    // Create methods

    virtual int Create(bool pManaged = TRUE, byte  pDirect = XmVERTICAL
                                            , short pColNum= 1);

    // Miscellaneous methods

    virtual char *GetClass()   { return "XolCheckBoxC"; }
};
```

8.28.2 Important Data Members

The class defines no further data members.

8.28.3 Usable Methods and Operators

XolCheckBoxC (XolWidgetC *pParent, char *pName);
XolCheckBoxC (Widget pParent, char *pName);

Description: These methods constructs a check box widget object.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle;
pName - Name of the check box widget object

Results: None

virtual int Create(bool pManaged = TRUE, byte Direct = XmVERTICAL
, short pColNum = 1);

Description: This method creates the underlying *OSF/Motif* row column widget with the orientation given by the parameter *pDirect*.

Parameters: *pManaged* - Determines whether the created widget should be managed automatically by its parent during the creation

TRUE - the widget is managed automatically (default)
FALSE - the widget is not managed automatically

pDirect - Check box orientation (by default XmVERTICAL)

XmHORIZONTAL- check buttons are arranged horizontally
XmVERTICAL - check buttons are arranged vertically

pColNum - Number of check button columns if the orientation is vertical or number of check button rows if the orientation is horizontal

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolManagerC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolCheckBoxC*)

8.29 Class XolMessageBoxC

This class encapsulates the *OSF/Motif* widget class message box. A message box simply provide some kind of messages to the user, and typically include buttons that allow the user to respond to the message. By default, a message box has a pixmap label, a string label, and three buttons: OK, Cancel, and Help. *OSF/Motif* provides different kinds of message boxes for error messages, information messages, question messages, warning messages, and working messages. For each of these different types of messages, the message box uses a different pixmap label. Also the arrangement and the labels of the buttons are dependent on the type of message. *OSF/Motif* divide message boxes and message dialogs. Message dialogs are message boxes embedded in a shell widget. The class *XolMessageBoxC* encapsulates the *OSF/Motif* message boxes as well message dialogs and allow the using of all type of messages. It is derived from class *XolManagerC* described in section 8.2. To handle possible events defined for message boxes, several virtual dummy callback methods are defined. These methods are called by the callback handler. The callback handler is automatically registered for the *OSF/Motif* events *XmCR_OK*, *XmCR_CANCEL*, and *XmCR_HELP*. To handle these events by own callback methods, either the dummy methods have to be overwritten within a derived class or own callback methods have to be registered for these events. The pixmap label of a message box widget object can be determined by a given pixmap object.

8.29.1 Class Declaration

```
#include <XOL/MsgBox.h>

class XolMessageBoxC : public XolManagerC
{
    static void CallbackHandler(Widget, XolMessageBoxC*, XolPointerT);

    XolPixmapC* mPixmap;
    bool        mPixCrtd;
    byte        mType;

  public:

    // Constructors

    XolMessageBoxC (XolWidgetC *pParent, char *pName
                                       , byte pDialogType = XOL_MESSAGE);

    XolMessageBoxC (Widget pParent, char *pName
                                  , byte pDialogType = XOL_MESSAGE);

    // Destructor

    virtual ~XolMessageBoxC();

    // Create methods

    virtual int Create (bool pManaged, bool  pOwnShell
                                     , char* pMessage
                                     , char* pPixmap = NULL);
    virtual int Create (bool pManaged, bool  pOwnShell
```

```
                                , char* pMessage = NULL
                                , XolPixmapC* pPixmap =NULL);

    // Dummy callback methods

    virtual void HelpEvent   () { return; }
    virtual void OkEvent     () { delete this; return; }
    virtual void CancelEvent () { delete this; return; }

    // Miscellaneous methods

    virtual void Append (char* pMessage, char *pCharSet = NULL);
    virtual void New    (char* pMessage, char *pCharSet = NULL);

    virtual void SetOkString     (char* pLabel));
    virtual void SetCancelString (char* pLabel));
    virtual void SetHelpString   (char* pLabel));

    virtual char *GetClass () { return "XolMessageBoxC"; }
};
```

8.29.2 Important Data Members

XolPixmapC* mPixmap;

Description: The data member is the pointer to the pixmap object that is used for specifying the pixmap label of *this* message box object.

Values: Pointer to pixmap object

byte mType;

Description: This data member holds the type of the messages displayed in *this* message widget object

Values:

XOL_ERROR	- error message box
XOL_INFO	- info message box
XOL_MESSAGE	- standard message box
XOL_QUESTION	- question message box
XOL_WARNING	- warning message box
XOL_WORKING	- working message box

8.29.3 Usable Methods and Operators

XolMessageBoxC (XolWidgetC *pParent, char *pName
, byte pDialogType = XOL_MESSAGE);
XolMessageBoxC (Widget pParent, char *pName
, byte pDialogType = XOL_MESSAGE);

Description: These methods construct a message box/dialog object for displaying messages of type given by the parameter *pDialogType*.

Parameters:

pParent	- Pointer to the parent widget object or the parent *Xt* widget handle;
pName	- Name of the radio box widget object
pDialogType	- Message Box/Dialog type

XOL_ERROR	- error message box
XOL_INFO	- info message box
XOL_MESSAGE	- standard message box (default)
XOL_QUESTION	- question message box
XOL_WARNING	- warning message box
XOL_WORKING	- working message box

Results: None

virtual ~XolMessageBoxC();

Description: This method destroys the message box widget object.

Parameters: None

Results: None

virtual int Create (bool pManaged, bool pOwnShell
, char* pMessage
, char* pPixmap = NULL);
virtual int Create (bool pManaged, bool pOwnShell
, char* pMessage = NULL
, XolPixmapC* pPixmap =NULL);

Description: These methods create the underlying *OSF/Motif* message box or message dialog. Message dialogs are message boxes embedded in an own shell. The parameter *pOwnShell* determines whether *this* widget object is managing a *OSF/Motif* message box or *OSF/Motif* message dialog.
If a pixmap object is specified by the parameter *pPixmap*, this pixmap object is used as the pixmap label of the message box / dialog. If a name of a pixmap file is given, see section 8.21, a pixmap object is created automatically.

Parameters: *pManaged* - Determines whether the created widget should be managed automatically by its parent during the creation

TRUE - the widget is managed automatically (default)
FALSE - the widget is not managed automatically

pOwnShell - Determines whether a *OSF/Motif* message box or *OSF/Motif* message dialog is created

TRUE - Message dialog widget is created
FALSE - Message box widget is created

pMessage - Message string
pPixmap - Pixmap object or pixmap file name

Results: None

virtual void HelpEvent ();
virtual void OkEvent ();
virtual void CancelEvent ();

Description: These methods are dummy methods that are called by the callback handler of *this* message box widget object when one of the three buttons is pressed. To handle the callbacks by own callback method, either these methods have to be overwritten by a derived class or the own callback methods have to be registered for the callback events by using the method *XolWidgetC::AddCallback*. By default the callback methods for the *OSF/Motif* events *XmCR_OK* and *XmCR_CANCEL* destroy *this* message box widget object.

Parameters: None

Results: None

virtual void Append (char* pMessage, char *pCharSet = NULL);
virtual void New (char* pMessage, char *pCharSet = NULL);

Description: These methods appends the given message *pMessage* to the existing message and replace the existing message with the given message, respectively. The message can be processed with a specific character set given by *pCharSet*. If this parameter is omitted the default character set is used.

Parameters: *pMessage* - Message that is appended to or replacing the existing message
pCharSet - Character set (NULL -> the default character set is used)

Results: None

virtual void SetOkString (char* pLabel);
virtual void SetCancelString (char* pLabel);
virtual void SetHelpString (char* pLabel);

Description: With these methods the default label text of the message box buttons can be changed

Parameters: *pLabel* - Button label text

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolManagerC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolMessageBoxC*)

8.30 Class XolMainWinC

This class belongs to the group of framework classes. It realizes very complex main window widget objects by using a number of underlying *OSF/Motif* widgets. For this purpose, the class itself encapsulates the *OSF/Motif* widget class *FormWidget*. With the form widget the *OSF/Motif* widget class *MainWindow* with additional functionality is imitated The *OSF/Motif* widget class *MainWindow* is a subclass of *ScrolledWindow* that provides support for a menu bar and specialised areas. In addition to the viewport and scroll bar components, also called working area, the *OSF/Motif* main window widget allows optional menu!bar, command window, and a message!window. The *OSF/Motif* main window widget lays out these components in a manner compliant with the *OSF/Motif* style guide [Fet94]. The menu bar, if present, spans the top of the main window widget horizontally. By default, the command window, if present, spans the main window horizontally just below the menu bar. The working area, is below the command window. The message window, if present, is below the working area. The command window can also be located below the working area and above the message window. The later arrangement of the command window is used by the class *XolMainWinC*.
Often an application shows an icon!bar that contains push buttons labelled by icon pixmaps which are associated with the most important actions of the application. Furthermore, often an application has an area in which an application logo is displayed. Therefore, in addition to the *OSF/Motif* main window specialised areas, the class *XolMainWinC* allows an optional icon!area and an optional logo area. All these widgets for specialised areas described above are put together within the main window widget object. The application can extend a main window widget object by its own widget object for specialised tasks. A typical main window widget object using the *XOL* class library is shown in Fig. 5.
Several widget objects for specialised areas are created automatically during the managing of a main window widget object dependent on the configuration of optional areas. Several widget objects can be replaced by applications widget objects to manipulate the default behaviour of the main window widget object. Each main window widget area is embedded within an automatically created frame widget object. The command window, the working area, and the message window are embedded additionally within a paned window. That is, that all of these windows can be resized by the user in an easy way. Thus, the class *XolMainWinC* realizes very complex but also very flexible main window widget objects.
There is a rule, in which order the methods of the main window widget object have to be used.

1. First, the main window widget object have to be constructed with a derivative of a shell widget object as parent.

2. The the main window widget object has to be created by using the method *Create*. The parameters of this method specify the optional main window areas. Furthermore, needed widget objects like frames widget objects around the main window areas or the paned window widget object are created dependent on the specified configuration.

3. After this, application defined area widget objects can be created and assigned for each main window area object by using the widget object referenced by the data element *Paned* as the parent widget object parent.

4. Further application defined areas can be created as children of the main window widget object.
5. After all these area widget object creations, the method *Manage* have to be called to create non-existing main window widget object areas and to exposure the main window widget object.
6. Now, all application widget object like icon push buttons that require an automatic created widget object as the parent can be created.

If any used main window area is not associated with a application defined widget object, a default widget object is created automatically for this main window area.
The class is derived from the class *XolManagerC* described in section 8.2.

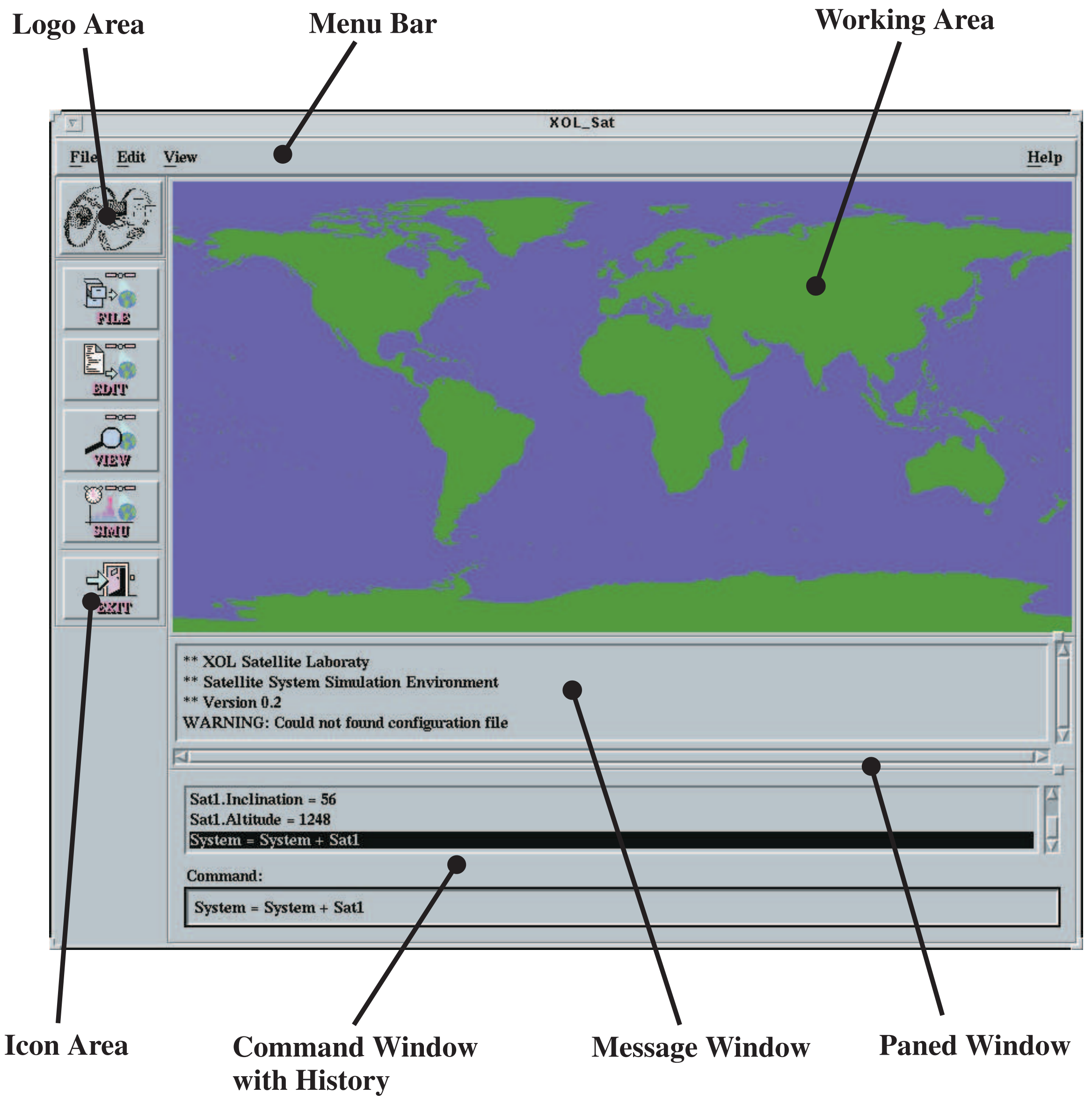

Figure 5: Typical main window widget object using the *XOL* class library

8.30.1 Class Declaration

```
#include <XOL/MainW.h>

class XolMainWinC : public XolManagerC
{
```

```
    bool mCmdAreaCrtd ,
         mMsgAreaCrtd ,
         mWorkAreaCrtd,
         mIconAreaCrtd,
         mWorkAreaUsed,
         mCmdAreaUsed ,
         mMsgAreaUsed ,
         mLogoAreaUsed;
    byte mIconAreaUsed;
    bool mManaged;

  public:

    XolWidgetC     *mWorkArea,
                   *mCmdArea;
    XolIconBarC    *mIconArea;
    XolTextWidgetC *mMsgArea ;
    XolLabelC      *mLogoArea;
    XolWidgetC     *mLogoFrame,
                   *mIconFrame,
                   *mPanedFrame,
                   *mPaned;
    XolMenuBarC    *mMenuBar;

    // Constructors

    XolMainWinC (XolWidgetC *pParent, char *pName);
    XolMainWinC (Widget pParent, char *pName);

    // Destructor

    virtual ~XolMainWinC();

    // Create Methods

    virtual int Create (bool pMenuUsed = TRUE, bool pWorkUsed = TRUE
                                             , bool pCmdUsed  = TRUE
                                             , bool pMsgUsed  = TRUE
                                             , bool pLogoUsed = FALSE
                                             , byte pIconUsed = XmVERTICAL);
    virtual void Manage ();

    // Miscellaneous methods

    XolWidgetC   *Paned     () { return mPaned;     }
    XolWidgetC   *LogoFrame () { return mLogoFrame; }
    XolWidgetC   *IconFrame () { return mIconFrame; }
    XolWidgetC   *MenuBar   () { return mMenuBar;   }

    virtual char *GetClass  () { return "XolMainWinC"; }
};
```

8.30.2 Important Data Members

XolWidgetC *mWorkArea;

Description: This data member holds the pointer to the main window viewport widget object, the so called working area widget object. The working area widget object is optional. It can be any *XOL* widget object or any application defined widget object. The working area widget object must be created and assigned to this data member after the execution of the method *XolMainWinC::Create*, but before the execution of the method *XolMainWinC::Manage*. The parent widget object of the working area widget object must be the widget object referenced by the data member *Paned*. When the working area using option is activated and no widget object is assigned to this data member when the method *Manage* is executed, a text widget object is automatically created as working area widget object.

Values: Pointer to any *XOL* or derived widget object

XolWidgetC *mCmdArea;

Description: This data member holds the pointer to the command window widget object. The command window is optional. This data member contains *NULL* when the command window using option was not activated. Normally, the command window is a text widget object, that is created automatically by the method *Manage*, when the command window using option was activated. It is possible to use any application defined widget object as command window. In this case the application defined command window widget object must be created and assigned to this data member after the execution of the method *Create* of the main window widget object, but before the execution of the method *Manage*. The parent widget object of the application defined command window widget object must be the widget object referenced by *Paned*.

Values: Pointer to the command window widget object

XolTextWidgetC *mMsgArea;

Description: This data member holds the pointer to the message!window widget object. The message window is optional. This data member contains *NULL* when the message window using option was not activated. Normally, the message window is a text widget object. When the message window using option is activated, it is created automatically by the method *Manage*. It is possible to use any application defined widget object as message window. In this case the application defined message window widget object must be created and assigned to this data member after the execution of the method *Create* of the main window widget object, but before the execution of the method *Manage*. The parent widget object of the application defined message window widget object must be the widget object referenced by *Paned*.

Values: Pointer to the message window widget object

XolWidgetC *mIconArea;

Description: This data member holds the pointer to the icon!area widget object. The icon area is optional. This data member contains *NULL* when the icon area using option was not activated. Normally, the icon area is an icon bar widget object from class *XolIconBarC* described in section 8.35. By default the icon bar widget object has one row/column arranged horizontally or vertically. When the icon area using option was activated, the icon bar widget object is created automatically by the method *Create*. Therefore, it is not possible to construct the icon buttons that should be arranged within the icon area widget object before the execution of the method *XolMainWinC::Create*. After the execution of this method the icon area widget object has to be used as the parent widget object for the creation of icon buttons widget object.

Values: Pointer to the icon area widget object

XolLabelC *mLogoArea;

Description: This data member holds the pointer to the logo area widget object. By default this data member is *NULL*. The logo area widget object is not created automatically. It has to be created and assigned by the application. It can be any *XOL* widget object or any application defined widget object. The widget object referenced by data element *mLogoFrame* has to be used as the parent widget object of the logo area widget object. Normally, push button widget object labelled by a pixmap object is used as the logo area widget object. The data member can be set anytime after the execution of the method *XolMainWinC::Create*.

Values: Pointer to the logo area widget object

*XolWidgetC *mLogoFrame;*

Description: This data element holds the pointer to the logo area!frame widget object. When the logo area using option is activated, the logo area frame widget object is created automatically during the execution of the method *Create*. Otherwise this data element is *NULL* and cannot be used. This data element has to be used as the parent for the creation of the logo area widget object.

Values: Pointer to the logo area frame widget object

*XolWidgetC *mIconFrame;*

Description: This data element holds the pointer to the icon!area frame widget object. When the icon area using option is activated, the icon area frame widget object is created automatically during the execution of the method *Create*. Otherwise this data element is *NULL* and cannot be used. This data element has to be used as the parent for the creation of the icon area widget object.

Values: Pointer to the icon area frame widget object

*XolWidgetC *mPanedFrame;*

Description: This data member holds the pointer to the frame widget object for the paned window widget object that encapsulates at least one of the main widget object areas command window, working area, and message area. If at least one of these areas are activated this data member is created automatically during the execution of the method *Create*. Otherwise this data member contains *NULL* and the data member cannot be used by the application.

Values: Pointer to the frame widget object of the paned window

*XolWidgetC *mPaned;*

Description: This data member holds the pointer to the paned window widget object. It encapsulates at least one of the main widget object areas command window, working area, and message area. When at least one of these areas are activated this data member is created automatically during the execution of the method *Create*. Otherwise this data member contains *NULL* and the data member cannot be used by the application. This data member has to be used as the parent widget object for constructing one of the main window widget object areas command window, working area, and message window.

Values: Pointer to the paned window widget object.

*XolMenuBarC *mMenuBar;*

Description: This data member holds the pointer to the menu bar widget object. The menu bar widget object is created automatically during the execution of the method *Create*. This data member has to be used as the parent widget object for the construction of the menu button objects.

Values: Pointer to the menu bar widget object

8.30.3 Usable Methods and Operators

*XolMainWinC (XolWidgetC *pParent, char *pName);*
*XolMainWinC (Widget pParent, char *pName);*

Description: These methods construct a main window widget object.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle (normally, the parent is a shell widget)
pName - Name of the main window widget object

Results: None

virtual ~XolMainWinC();

Description: This method destroys the main window widget object. All automatic created widget objects are also destroyed automatically.

Parameters: None

Results: None

virtual int Create (bool pMenuUsed = TRUE, bool pWorkUsed = TRUE
, bool pCmdUsed = TRUE
, bool pMsgUsed = TRUE
, bool pLogoUsed = FALSE
, byte pIconUsed = XmVERTICAL);

Description: This method constructs and creates several needed main window area widget objects and it underlying *OSF/Motif* widgets automatically:

- *mMenuBar* - Menu bar widget object:
  If the parameter *pMenuUsed* is *TRUE*, this widget object is created automatically.
- *mPaned* - Paned window widget object:
  If one of the parameters *pWorkUsed*, *pCmdUsed*, and *pMsgUsed* is *TRUE*, this widget object is created automatically. It embeds at least one of the optional areas command window, working area, and message window
- *mPanedFrame* - Frame widget object for the paned widget object:
  If the paned widget object is automatically created, see above, this widget object is also created automatically.
- *mLogoFrame* - Frame widget object for the logo area:
  If the parameter *pLogoUsed* is *TRUE*, this widget object is created automatically.
- *mIconFrame* - Frame widget object for the icon area:
  If the parameter *pIconUsed* is different from *XmNO_ORIENTATION*, this widget object is created automatically.

The parameter of this method specify which optional main window widget area objects are used.
While for other widget objects it is possible to manage the underlying *OSF/Motif* widgets during the widget object creation, the underlying *OSF/Motif* widgets used by the main window widget object have to be managed explicitely by using the method *Manage*. the widget

Parameters:

pMenuUsed - Determines whether the main window widget object has a menu bar
- *TRUE* - Menu bar is used at the top of the main window widget (default)
- *FALSE* - Menu bar is not used

pWorkUsed - Determines whether the main window widget object has a working area
- *TRUE* - Working area is used (default)
- *FALSE* - Working area is not used

pCmdUsed - Determines whether the main window widget object has a command window
- *TRUE* - Command window is used (default)
- *FALSE* - Command window is not used

pMsgUsed - Determines whether the main window widget object has a message window
- *TRUE* - Message window is used (default)
- *FALSE* - Message Window is not used

pLogoUsed - Determines whether the main window widget object has a logo area
- *TRUE* - Logo area is used
- *FALSE* - Logo area is not used

pIconUsed - Determines whether the main window widget object has a icon area
- *XmVERTICAL* - Vertical icon area is used at the left side of the main window widget

XmHORIZONTAL	- Horizontal icon area is used below the menu bar
XmNO_ORIENTATION	- Icon area is not used

Results: None

virtual void Manage ();

Description: This method manages the underlying *OSF/Motif* widget objects. Furthermore, during the first execution it creates needed main window area widget objects automatically that are not set by the application at this time:

- *mIconArea* - Icon area widget object:
  If the parameter *pIconUsed* of the method *Create* was not *XmNO_ORIENTATION*, and this widget object is not existing yet, it is created automatically with the specified by the parameter *pIconUsed* of the method *Create*.
- *mLogoArea* - Logo area widget object:
  If the parameter *pLogoUsed* of the method *Create* was *TRUE*, and this widget object is not existing yet, it is created automatically.
- *mMsgArea* - Message window area widget object:
  If the parameter *pMsgUsed* of the method *Create* was *TRUE*, and this widget object is not existing yet, it is created automatically. In this case, the created message window is a text widget object with 3 lines and 80 columns.
- *mCmdArea* - Command window area widget object:
  If the parameter *pCmdUsed* of the method *Create* was *TRUE*, and this widget object is not existing yet, a command window widget object is created automatically.
- *mWorkArea* - Working area widget object:
  If the parameter *pWorkUsed* of the method *Create* was *TRUE*, and this widget object is not existing yet, it is created automatically. In this case, the created message window is a text widget object with 25 lines and 80 columns.

Parameters: None

Results: None

XolWidgetC *Paned ();
XolWidgetC *LogoFrame ();
XolWidgetC *IconFrame ();
XolWidgetC *MenuBar ();

Description: These methods deliver the pointers to several main window area widget objects

Parameters: None

Results: Pointer to the widget object

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolManagerC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolMainWinC*)

8.31 Class XolApplicationC

The class *XolApplicationC* encapsulates the common programming model of each *OSF/Motif* application. It handles the data elements needed for further application handling after the creation of the application object. The class belongs to the group of framework classes. It does not encapsulate a *OSF/Motif* widget directly. Rather it uses widget objects and *OSF/Motif* functionality to encapsulate the *OSF/Motif* application programming model within an object.
The *OSF/Motif* programming model is the same as the *Xt* programming model [You90]:

1. **Initialise the Xt Intrinsics.** This step establishes a connection to the X server, allocates resources, and initialises the *Xt Intrinsics* layer.
2. **Create widgets.** Every program creates one or more widgets to construct the user interface.
3. **Register callbacks and event handlers.** Callbacks and event handlers are application-defined functions that respond to user actions and events that occur within each widget.
4. **Realize all widgets.** Realizing a widget creates the X window used by the widget.
5. **Enter the event loop.** X window application are completely event-driven, and are designed to loop indefinitely responding to events. The event loop retrieves events from the X event queue and dispatches the event to the appropriate event handler or callback function.

The *XOL* programming model is quite similar:

1. **Construct the application object.** During the construction the data members of the application object are initialised. Since the application object class is derived from class *XolWmShellC* the constructor of this class is executed. Thus with the construction of the application object a shell widget object is created implicitly. The class of the shell widget is of course applicationShellWidget. Furthermore, the application object has no parent widget object. The constructed application object should be stored in the global variable *sGlobApp*.
2. **Execute method Create.** The method *Create* initialises the *Xt Intrinsics* layer. Furthermore, it creates the underlying *Xt* application shell widget with given size at a given position.
3. **Create widget objects and register its callback methods.** This step can be done in two different ways:
   (a) After the execution of the method *Create* all widgets are created and its callback methods are registered before the method *Run* is called.
   (b) The application object class is derived to an own application object class that overwrites the virtual method *Init*. The method *Init* is called as the first step of the method *Run*. Within the method *Init* all widget objects can be created and its callback methods can be registered.
4. **Run the application.** The application is executed by calling the method *Run*. This method calls as the first step the virtual method *Init*. By default, the method *Init* has no functionality. However within a derived class the method can be overwritten with real functionality. By means of this mechanism, the method *Init* can realize whole or at least additional initialisations. If no main window widget object was created in preceding initialisations, the method *Run* creates a default main window widget object automatically. Following, the method *Run* realize the underlying *Xt* application shell widget, i.e., the underlying application shell widget as well its children are managed. As the final step, the method *Run* calls the infinite event response loop.

Thus, the *OSF/Motif* programming model is encapsulated the within methods of the application object. The application object MUST BE CREATED only once. The created application object should be assigned to the global variable *sGlobApp* to enable the global access to applications context information. The class *XolApplicationC* is derived from class *XolManagerC* described in section 8.2. Thus an application object has the whole functionality of manager widget objects.

8.31.1 Class Declaration

```
#include <XOL/App.h>

class XolApplicationC : public XolWmShellC
{
    XtAppContext mContext;   // holds the application context
    String* mFallbackTable;  // holds the application fallback resources
    int     mFallbackCount;  // holds the number of application fallback res.
    char**  mArgV;           // holds the command line argument vector
    int     mArgC;
```

```
  public:

    XolMainWinC *mWorkPlace; // application main window widget object
    Display     *mDisplay;   // holds the main application display
    char        *mClass;     // holds the applications class name

    // Constructors

    XolApplicationC(char *pName, char *pClass, int pArgC, char **pArgV);

    // Create methods

    void AddFallbackRes(char *pResStr);

    virtual int  Create(bool pScroll = FALSE, int xPos = 0, int yPos = 0
                                         , int xSize= 0, int ySize= 0);

    virtual void Manage() { return; }  // Manage have to be dummy function
                                       // for the application object
    virtual void Init()   { return; }  // Init dummy method called by method
                                       // Run
    // Miscellaneous methods

    virtual void  Run ();
    virtual void  SetDefColors (char *pBackground);
    virtual char* GetClass ()  { return "XolApplicationC"; }

    XtAppContext GetAppContext() { return mContext; }
};
```

8.31.2 Important Data Members

XtAppContext mContext;

Description: This data member holds the application context that is created when the *Xt Intrinsics* layer is initialised.

Values: Application context

String* mFallbackTable;

Description: This data member is the table of fallback resources of the application. Fallback resources are used to specify the value of *X Windows* and *OSF/Motif* resources when neither an application specific nor an user specific resource definition exists. The contents of fallback resource table is determined by the standard fallback table given by the global variable *sFallbacks* and the fallback resources added by using the method *AddFallbackRes*.

Values: Fallback resource strings

int mFallbackCount;

Description: This data member holds the number of entries within in the fallback resource table

Values: Integer value

XolMainWinC *mWorkPlace;

Description: This data member holds the pointer to the application main window widget object. This widget object is a normally a main window widget object that is created by the application and assigned to this data member. The parent of the main window widget object must be the data widget object referenced by the data member *mClientArea* inherited from class *XolWmShellC*. If no main window widget object is created and assigned to this data member when the method *Run* is called, then the method *Run* creates a default main window widget object.

Values: Pointer to a main window widget object

Display *mDisplay;

Description: This data element holds the reference to the applications display descriptor.

Values: Pointer to the display descriptor

char *mClass;

Description: This data element holds the application class name that is specified as a parameter of the object constructor.

Values: String

8.31.3 Usable Methods and Operators

XolApplicationC (char *pName, char *pClass, int pArgV, char **pArgV);

Description: The method constructs a application object. During the construction all data members are initialised. Although the application object class is derived from class *XolWmShellC* the application object has no parent widget object. The constructor of the basis class instances a shell widget object. The class of the shell widget object is applicationShellWidget. The constructed application object should be stored in the global variable *sGlobApp*. The application object has to be constructed before any widget object can be created.

Parameters:
- *pName* - Name of the application object
- *pClass* - Class name of the application, that determines the name of the application default file and the default title of the main window widget (NULL -> the name of application object is also used as class name
- *pArgC* - Number of command line arguments stored in parameter *pArgV*
- *pArgV* - Command line argument vector

Results: None

virtual int Create (bool pScroll = FALSE, int xPos = 0, int yPos = 0
, int xSize = 0, int ySize = 0);

Description: This method initialises the *Xt Intrinsics* layer and creates the underlying *Xt* application shell widget with a given size at a given position. The method has to be executed before any widget object can be created. If additional fallback resources should be specified, the method *AddFallbackRes* has to be used before using the method *Create*. After the execution of method *Create*, it is impossible to add further fallback resources to the application fallback resource table.

Parameters:
- *pScroll* - Scrolling policy for the embedded main window widget object.
  - *TRUE* - the main window widget object is capable to scroll its contents, but only if it is needed dependent on the geometry
  - *FALSE* - the main window widget object is not capable to scroll its contents
- *xPos* - application shell widget x position (0 - not specified)
- *yPos* - application shell widget y position (0 - not specified)
- *xSize* - application shell width
  - 0 && (*pScroll==TRUE*) - XOL_DEF_XSIZE
  - 0 && (*pScroll==FALSE*)- automatic x size

	ySize	- application shell height 0 && (*pScroll*==*TRUE*) - XOL_DEF_YSIZE 0 && (*pScroll*==*FALSE*)- automatic y size
Results:	None	

virtual void Manage();

Description: The application object cannot managed explicitely. It is realized/ managed automatically during the execution of the method *Run*. Therefore, the virtual method *Manage* of the basis class is overwritten with a dummy.

Parameters: None

Results: None

virtual void Init();

Description: This method is a virtual dummy method that have no functionality. The method is called as the first step by the method *Run*. It can be overwritten within a derived application class to realize real functionality. By means of this mechanism, the method *Init* can realize whole or at least additional initialisations. The original purpose of this method is the creation of widget objects as well the registration of its callback methods.

Parameters: None

Results: None

virtual void Run ();

Description: The application is executed by calling the method *Run* of the application object. This method calls as the first step the virtual method *Init*. Then, if no main window widget object was created in preceding initialisations, the method *Run* creates a default main window widget object automatically. Following, this method realizes the underlying *OSF/Motif* application shell widget. That is, the application shell widget as well its children are managed. As the final step, the method *Run* calls the infinite event response loop. Since, the method returns when the application is ended, the method can be considered as infinite Thus, all necessary initialisation have to be done before the execution of this method.

Parameters: None

Results: None

void AddFallbackRes(char *pResStr);

Description: The method adds a string as a fallback resource to the fallback resource table of the application object. The method can only be executed before method *Create* is called.

Parameters: *pResStr* - Resource string

Results: None

virtual void SetDefColors (char *pBackground);

Description: With this method the default colours held in global variables are set. These colours are used for all widget objects that are created after the setting. The colour set is calculated by means of an *OSF/Motif* library function based on the background colour given by *pBackColor*. See the *OSF/Motif* manual for more information.

Parameters: *pBackground*- Background colour name

Results: None

XtAppContext GetAppContext();

Description: This method delivers the application context stored in the data member *mContext*.

Parameters: None

Results: Application context

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolWmShellC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolApplicationC*)

8.31.4 Example

The examples below show the two ways to use the application object by an application. In first example the global application object is constructed and created. After this the applications main window widget object is constructed and created. Than the main window area objects that shall not be created automatically are constructed, e.g. the logo area or the icon area. Last, the main window widget object is managed and the the application object is started by using the method *Run*.

```
int main(int argc, char *argv[])
{
    XolIconButtonC   *Icon;

    sGlobApp = new XolApplicationC("Example","Example", argc, argv);
    sGlobApp->AddFallbackRes(MyFonts);
    sGlobApp->AddFallbackRes(MyColorList);
    sGlobApp->Create();
    sGlobApp->mWorkPlace = new XolMainWinC(sGlobApp->mClientArea,"Main");
    sGlobApp->mWorkPlace->Create(TRUE, TRUE, TRUE, FALSE, TRUE, XmVERTICAL);

    sGlobApp->mWorkPlace->mLogoArea = new XolPushButtonC(WorkPlace->mLogoFrame,
                                                         "Logo", XmPIXMAP);
    sGlobApp->mWorkPlace->mLogoArea->Create(TRUE,"logo.xpm");
    sGlobApp->mWorkPlace->mLogoArea->AddCallback(XmNactivateCallback,
                                                (XolCallbackProcT)Function,
                                                 WorkPlace->mLogoArea);

    Icon = new XolIconButtonC(sGlobApp->mWorkPlace->mIconArea,
                              "EditIcon","SAT_SysEdit48.xpm");
    Icon->Create(TRUE);
    Icon->AddCallback(XmNactivateCallback, Function, (XolPointerT)sGlobApp);

    ....

    sGlobApp->mWorkPlace->Manage();
    sGlobApp->Run();
}
```

The second example shows the alternative way to construct and create the application object. In this way the application object class is derived and the virutal dummy method *XolApplicationC::Init* is overwritten. The whole initialisation is than realised in this method.

```
class MyApplicationC : public XolApplicationC
{
  public:

    MyApplicationC (char *pName, char *pClass, int pArgC, char **pArgV)
                  : XolApplicationC (pName, pClass, pArgC, pArgV);
```

```
    void Init();
};

void MyApplicationC:: Init()
{
  XolIconButtonC   *Icon;

  mWorkPlace = new XolMainWinC(mClientArea,"Main");
  mWorkPlace->Create(TRUE, TRUE, TRUE, FALSE, TRUE, XmVERTICAL);

  mWorkPlace->mLogoArea = new XolPushButtonC(WorkPlace->mLogoFrame,
                                          "Logo", XmPIXMAP);
  mWorkPlace->mLogoArea->Create(TRUE,"logo.xpm");
  mWorkPlace->mLogoArea->AddCallback(XmNactivateCallback,
                                    (XolCallbackProcT)Function,
                                    WorkPlace->mLogoArea);

  Icon = new XolIconButtonC(mWorkPlace->mIconArea,"EditIcon"
                                                 ,"SAT_SysEdit48.xpm");
  Icon->Create(TRUE);
  Icon->AddCallback(XmNactivateCallback, Function, (XolPointerT)this);

  ....
}

int main(int argc, char *argv[])
{
    sGlobApp = new MyApplicationC("Example","Example", argc, argv);
    sGlobApp->AddFallbackRes(MyFonts);
    sGlobApp->AddFallbackRes(MyColorList);
    sGlobApp->Create();
    sGlobApp->Run();
}
```

8.32 Class XolScaleC

This class belongs to the set of specialised *XOL* widget object classes. It encapsulates the *OSF/Motif* widget class *Scale*. A scale widget displays a slider somewhat similar in appearance and functionality to a Scrollbar. The slider represents a value within a range and allows optionally the user to supply a new value by dragging the slider button. The *OSF/Motif* scale widget class defines two new callback events *XmCR_DRAG* and *XmCR_VALUE_CHANGED*. A scale widget has a resource that controls the orientation of the scale. The orientation can be vertical as well horizontal. The scale widget can have a label. Furthermore it allows the displaying of the value in numeric format around the slider. The value as well the range of the value displayed by a scale widget can be changed by the application.
The class *XolScaleC* is derived from widget object class *XolManagerC* described in section 8.2. To handle possible callback events *XmCR_DRAG* and *XmCR_VALUE_CHANGED*, this class defines several virtual dummy methods that are called by the callback handler. The callback handler is registered automatically for these *OSF/Motif* events. To handle these events by own callback methods, either the dummy methods have to be overwritten within a derived class or own callback methods have to be registered for the events. The class defines several convenient methods to set and get the value as well its range either in floating point format or in integer format with a given decimal point position.

8.32.1 Class Declaration

```
#include <XOL/Scale.h>

class XolScaleC : public XolManagerC
```

```
{
    static void CallbackHandler(Widget, XolScaleC*, XolPointerT);

  public:

    // Constructors

    XolScaleC (XolWidgetC* pParent, char *pName)
              : XolManagerC(pParent, pName, xmScaleWidgetClass) {}

    XolScaleC (Widget pParent, char *pName)
              : XolManagerC(pParent, pName, xmScaleWidgetClass) {}

    // Create methods

    virtual int Create (bool pManaged = TRUE, byte  pOrient = XmVERTICAL
                                            , int   pMin    = 0
                                            , int   pMax    = 10
                                            , short pDecimal= 0
                                            , bool  pValInd = TRUE);

    virtual int Create (bool pManaged, byte pOrient, float pMin, float pMax);

    // Methods that handle the widget object attributes

    virtual void SetLabel (char *pLabel);

    virtual void SetParam (int  pMin, int  pMax, short  pDecimal);
    virtual void GetParam (int& pMin, int& pMax, short& pDecimal);

    virtual void SetParam (float  pMin, float  pMax);
    virtual void GetParam (float& pMin, float& pMax);

    virtual void SetValue (int  pVal);
    virtual void GetValue (int& pVal);

    virtual void SetValue (float  pVal);
    virtual void GetValue (float& pVal);

    // Dummy callback methods

    virtual void ChangedEvent (int)   { return; }
    virtual void ChangedEvent (float) { return; }
    virtual void DragEvent    (int)   { return; }
    virtual void DragEvent    (float) { return; }

    // Miscellaneous methods

    virtual char *GetClass()   { return "XolScaleC"; }
};
```

8.32.2 Important Data Members

The class defines no further data members.

8.32.3 Usable Methods and Operators

XolScaleC (XolWidgetC* pParent, char *pName)
XolScaleC (Widget pParent, char *pName)

Description: These methods construct a scale widget object. They have no further functionality than calling the constructors of the basis class. After its construction the underlying *OSF/Motif* scale widget has to be created by using the method *Create*.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle
pName - Name of the scale widget object

Results: None

virtual int Create (bool pManaged = TRUE, byte pOrient = XmVERTICAL
, int pMin = 0
, int pMax = 10
, short pDecimal = 0
, bool pValInd = TRUE);
virtual int Create (bool pManaged, byte pOrient, float pMin
, float pMax
, bool pValInd = TRUE);

Description: These methods create the underlying *OSF/Motif* scale widget according to the given parameter. They register the callback handle as callback methods for the possible *OSF/Motif* events *XmCR_DRAG* and *XmCR_VALUE_CHANGED*.

Parameters: *pManaged* - Determines whether the created widget should be managed automatically by its parent during the creation
TRUE - the widget is managed automatically (default)
FALSE - the widget is not managed automatically
pOrient - Determines the orientation of the scale
XmVERTICAL - the scale is oriented vertically (default)
XmHORIZONTAL- the scale is oriented horizontally
pMin - Minimum value given as floating point or integer value
pMax - Minimum value given as floating point or integer value
pDecimal - Position of the decimal point that is used by the widget object to calculate the minimum and maximum value if these values are given as integer values
pValInd - Determines whether or not the scale is labelled additionally by the value
TRUE - the scale is labelled by value
FALSE - the scale is not labelled by value

Results: None

virtual void SetLabel (char *pLabel);

Description: This method set the text of the scale widget object label. By default, the label text is the widget object name.

Parameters: *pLabel* - Label text

Results: None

virtual void GetParam (int
virtual void GetParam (float& pMin, float& pMax);
virtual void SetParam (int pMin, int pMax, short pDecimal);
virtual void SetParam (float pMin, float pMax);

Description: These methods get/set the range of the scale value. The range values can be given as integer or floating value parameter. However, in case of integer value parameter, the decimal point position, .i.e., the decimal precision has to be given additionally.

Parameters: *pMin* - Minimum value taken/given as floating point or integer value parameter

	pMax	- Minimum value taken/given as floating point or integer value parameter
	pDecimal	- Position of the decimal point that is used by the widget object to calculate the minimum and maximum value if these values are given as integer values
Results:	None	

virtual void GetValue (int& pVal);
virtual void GetValue (float& pVal);
virtual void SetValue (int pVal);
virtual void SetValue (float pVal);

Description: These methods get/set the value of the scale widget object. The values can be given as integer or floating value parameter. However, in case of integer value parameter the current decimal precision has to be considered.

Parameters: *pVal* - Value taken/given as floating point or integer value parameter

Results: None

virtual void ChangedEvent (int);
virtual void ChangedEvent (float);
virtual void DragEvent (int);
virtual void DragEvent (float);

Description: These methods are dummy methods that are called by the callback handler of *this* scale widget object. To handle the callbacks by own callback method, either these methods have to be overwritten by a derived class or the own callback methods have to be registered for the callback events by using the method *XolWidgetC::AddCallback*.

Parameters: The parameter delivers the new scale object value as integer or floating point value. Both the callback method with integer value parameter and the callback method with floating point value parameter is called by the callback handler for each occurred event.

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolManagerC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolScaleC*)

8.33 Class XolScaleBoxC

The class *XolScaleBoxC* realizes a more complex scale widget box by using several widget objects. Therefore, it belongs to the set of framework classes.

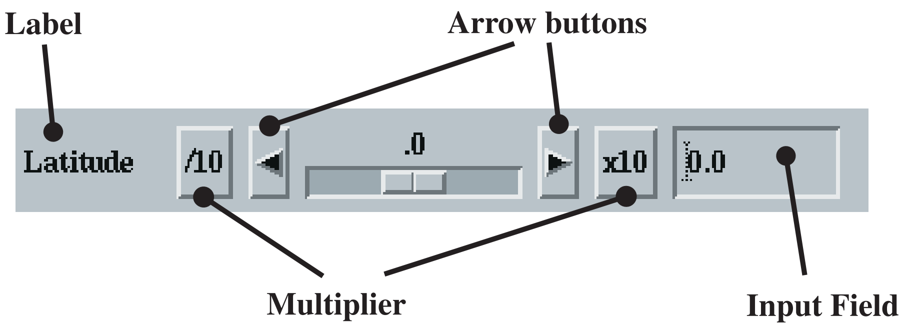

Figure 6: *XOL* scale box widget object

The scale box widget object can decorate an underlying scale widget object by several additional widget objects, Fig. 6, including

- a text field widget object that allows the user to put in the value directly,
- arrow button widget objects that allow the change of the value stepwise
- multiplier button widget objects that allow the change of the values range, and
- a label widget object that allow the labelling of the scale widget object.

Within the text field widget object object the user can input the value directly. The position of the slide is changed automatically. With arrow buttons the user can change the value stepwise with step size that is equal to the decimal precision. The slider is moved automatically. Using the multiplier buttons the user can change the decimal precision or the range value of the slider or both.
The scale widget box itself is a row column widget in which several child widget objects are arranged that decorate the scale widget. The class is derived from *XolManagerC* described in section 8.2. Due to the similarity to the class *XolScaleC*, the class *XolScaleBoxC* defines almost the same set of methods as the class *XolScaleC*. To handle possible callback events generated by widget objects within the scale box widget object, the class registers a callback handle for all possible callback events. The callback events are handled by the scale box widget object. If the value of the scale is changed, the callback handler calls additionally a callback method defined by the dummy method. To handle value changes by own callback methods, the dummy methods have to be overwritten within a derived class or the own callback methods have to be registered for the callback events.
As class *XolScaleC*, the class *XolScaleBoxC* defines several convenient methods to set and get the value as well its range either in floating point format or in integer format with a given decimal point position.

8.33.1 Class Declaration

```
#include <XOL/Scale.h>

#define XOL_MOD_PREC    1
#define XOL_MOD_MINMAX  2
#define XOL_MOD_BOTH    3

class XolScaleBoxC : public XolManagerC
{
    XolPushButtonC  *mMultiMin;
    XolPushButtonC  *mMultiMax;
    XolWidgetC      *mArrowMin;
    XolWidgetC      *mArrowMax;
    XolScaleC       *mScale;
    XolLabelC       *mLabel;
    XolTextWidgetC  *mInput;

    bool  mSetMode;
    byte  mMode;

    static void CallbackHandler(Widget, XolScaleBoxC*, XolPointerT);

  public:

    // Constructors

    XolScaleBoxC (XolWidgetC* pParent, char *pName);
    XolScaleBoxC (Widget pParent, char *pName);

    // Create methods

    virtual int Create (bool pManaged = TRUE, bool  pArrows = TRUE
                                            , bool  pMultipliers = TRUE
                                            , byte  pMultiMode   = XOL_MOD_BOTH
                                            , byte  pInputField  = 5
```

```
                                            , byte  pOrient = XmVERTICAL
                                            , int   pMin = 0
                                            , int   pMax = 10
                                            , short pDecimal = 0);

    virtual int Create (bool pManaged, byte pOrient, float pMin, float pMax);

    // Methods that handle the widget object attributes

    virtual void SetLabel (char *pLabel);

    virtual void SetParam (int  pMin, int  pMax, short  pDecimal);
    virtual void GetParam (int& pMin, int& pMax, short& pDecimal);

    virtual void SetParam (float  pMin, float  pMax);
    virtual void GetParam (float& pMin, float& pMax);

    virtual void SetValue (int  pVal);
    virtual void GetValue (int& pVal);

    virtual void SetValue (float  pVal);
    virtual void GetValue (float& pVal);

    // Dummy callback methods

    virtual void ValueChanged (int)   { return; }
    virtual void ValueChanged (float) { return; }

    // Miscellaneous methods

    virtual char *GetClass()   { return "XolScaleBoxC"; }

    void CallbackMethod(Widget, XolPointerT);   // !!! must not be used
};
```

8.33.2 Important Data Members

XolPushButtonC *mMultiMin;
XolPushButtonC *mMultiMax;

Description: These data members hold the pointers to the push button widget objects that serve as multiplier buttons. If multiplier buttons are activated the method *Create* constructs and create the push button widget objects automatically. Otherwise these data members are *NULL*.

Values: Pointer to push button widget object

XolWidgetC *mArrowMin;
XolWidgetC *mArrowMax;

Description: These data members hold the pointers to the arrow button widget objects that serve as multiplier buttons. If arrow buttons are activated the method *Create* constructs and create the widget objects automatically. Otherwise these data members are *NULL*.

Values: Pointer to widget object

XolScaleC *mScale;

Description: This data member holds the pointer to the scale widget object. The scale widget object is created automatically by the method *Create*.

Values: Pointer to scale widget object

XolLabelC *mLabel;

Description: This data member holds the pointer to the label widget object. If the label is activated, the label widget object is created automatically by the method *Create*. Otherwise the data member is *NULL*.

Values: Pointer to label widget object

XolTextWidgetC *mInput;

Description: This data member holds the pointer to the text field widget object. If the input field is activated, the text field widget object is created automatically by the method *Create*. Otherwise the data member is *NULL*.

Values: Pointer to text field widget object

8.33.3 Usable Methods and Operators

XolScaleBoxC (XolWidgetC* pParent, char *pName);
XolScaleBoxC (Widget pParent, char *pName);

Description: These methods construct the scale box widget object. They execute the basis class constructor and initialise all data member.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle
pName - Name of the scale widget object

Results: None

virtual int Create (bool pManaged = TRUE, bool pArrows = TRUE
, bool pMultipliers = TRUE
, byte pMultiMode = XOL_MOD_BOTH
, byte pInputField = 5
, byte pOrient = XmVERTICAL
, int pMin = 0
, int pMax = 10
, short pDecimal = 0);
virtual int Create (bool pManaged, byte pOrient, float pMin, float pMax);

Description: These methods create the underlying *OSF/Motif* column widget. They constructs as well create the scale widget object as well the decorating widget objects according to the given parameters. Furthermore, they register the callback handler as callback method for relevant events, so that the value changing can be handled by the scale box widget object.

Parameters:

pManaged - Determines whether the created widget should be managed automatically by its parent during the creation

- *TRUE* - the widget is managed automatically (default)
- *FALSE* - the widget is not managed automatically

pMultipliers - Determines whether or not the scale is decorated by multiplier buttons

- *TRUE* - the scale is decorated by value
- *FALSE* - the scale is not decorated by value

pMultiMode - Defines the multiplying mode for multipliers when they are activated

- *XOL_MOD_PREC* - Using a multiplier button leads only to the change of the decimal precision.
- *XOL_MOD_MINMAX* - Using a multiplier button leads only to the change of either the minimum or the maximum value.
- *XOL_MOD_BOTH* - Using a multiplier button leads to the change of both the decimal precision and either the minimum or the maximum value.

	pInputField	- Determines whether or not the scale is decorated by text field widget object. Furthermore, the value determines the length of the text field widget object. If this parameter 0, the scale is not decorated by an input field.
	pOrient	- Determines the orientation of the scale box *XmVERTICAL* - the scale box is oriented vertically (default) *XmHORIZONTAL* - the scale box is oriented horizontally
	pOrient	- Determines the orientation of the scale
Results:	None	

virtual void SetLabel (char *pLabel);

Description: This method set the text of the scale widget object label. By default, the label text is the widget object name.

Parameters: *pLabel* - Label text

Results: None

virtual void GetParam (int
virtual void GetParam (float& pMin, float& pMax);
virtual void SetParam (int pMin, int pMax, short pDecimal);
virtual void SetParam (float pMin, float pMax);

Description: These methods get/set the range of the scale value. The range values can be given as integer or floating value parameter. However, in case of integer value parameter, the decimal point position, .i.e., the decimal precision has to be given additionally.

Parameters:	*pMin*	- Minimum value taken/given as floating point or integer value parameter
	pMax	- Minimum value taken/given as floating point or integer value parameter
	pDecimal	- Position of the decimal point that is used by the widget object to calculate the minimum and maximum value if these values are given as integer values

Results: None

virtual void GetValue (int& pVal);
virtual void GetValue (float& pVal);
virtual void SetValue (int pVal);
virtual void SetValue (float pVal);

Description: These methods get/set the value of the scale widget object. The values can be given as integer or floating value parameter. However, in case of integer value parameter the current decimal precision has to be considered.

Parameters: *pVal* - Value taken/given as floating point or integer value parameter

Results: None

virtual void ChangedEvent (int);
virtual void ChangedEvent (float);

Description: These methods are dummy methods that are called by the callback handler of *this* scale widget object. To handle the callbacks by own callback method, either these methods have to be overwritten by a derived class or the own callback methods have to be registered for the callback events by using the method *XolWidgetC::AddCallback*. In later case the callback methods are registered in the callback method list and are therefore executed behind the execution of the callback handle. Since the whole value changing is handled by the scale box widget object, own callback methods can use the methods of the scale box widget object to get the changed values.

Parameters: The parameter delivers the new scale value as integer or floating point value. Both the callback method with integer value parameter and the callback method with floating point value parameter are called by the callback handler for each occurred event.

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolManagerC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolScaleBoxC*)

8.34 Class XolIconButtonC

Icon button widget objects are push button widget objects that can only be labelled by pixmaps. Furthermore they are arranged within an icon bar widget object, see section 8.35. Their parent is always a icon bar widget object. After their construction they are created automatically by their parent icon bar widget object. Icon button widget objects can be added to its parent icon bar widget object either before the icon bar widget object is created or after the icon bar widget object is created. In first case, the icon button widget object is created automatically by the icon bar widget object during its creation. In second case the icon button widget object has to be created explicitely because the icon car widget object can create only icon button widget objects that are known during its creation. The class *XolIconButtonC* is derived from class *XolPushButtonC* described in section 8.23.

8.34.1 Class Declaration

```
#include <XOL/IconW.h>

class XolIconButtonC : protected XolPushButtonC
{
    XolPixmapC *mPixmap;
    bool  mCreated;
    char* mFile;                 // PixMap file name

  public:

    XolIconButtonC (XolIconBarC *pParent, char *pName, char *pFile);
    ~XolIconButtonC ();

    Create (bool pManaged = TRUE);

    virtual char *GetClass ()   { return "XolIconButtonC"; }

    friend XolIconBarC;
};
```

8.34.2 Important Data Members

The class defines only data members that are used internally.

8.34.3 Usable Methods and Operators

XolIconButtonC (XolIconBarC *pParent, char *pName, char *pFile);

Description: This method constructs an icon button widget object. The label pixmap is not created during the construction.

Parameters: *pParent* - Pointer to the parent icon button widget object
pName - Name of the icon button widget object
pFile - Name of the label pixmap file

Results: None

~XolIconButtonC ();

Description: This method destroys the label pixmap.

Parameters: None

Results: None

int Create (bool pManaged = TRUE);

Description: This method constructs the label pixmap object and creates the underlying *OSF/Motif* push button widget. The method is executed by the parent icon bar widget object during its creation. For icon button widget objects that are constructed after the creation of the parent icon bar widget object this method has to be called explicitely.

Parameters: *pManaged* - Determines whether the created widget should be managed automatically by its parent during the creation

TRUE - the widget is managed automatically (default)
FALSE - the widget is not managed automatically

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolPushButtonC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolIconButtonC*)

8.35 Class XolIconBarC

The class *XolIconBarC* realizes icon bar widget objects that manage a number of icon button widget objects described in section 8.34. The icon bar can be oriented horizontally and vertically. The number of rows or columns can be specified. Icon button widget objects that shall be managed by a icon bar widget object have to use this object as its parent. Icon button widget object that are constructed before the method *Create* of its parent icon bar widget object is called, are created automatically during the creation of the icon bar widget object. Icon button widget objects that are constructed after their parent icon bar construction have to be created explicitely. The icon bar widget object can also contain separator widget objects, section 8.26.
The class *XolIconBarC* is derived from the class *XolManagerC* described in section 8.2.

8.35.1 Class Declaration

```
#include <XOL/IconW.h>

class XolIconBarC : public XolManagerC
{
  public:

    XolIconBarC (XolWidgetC *pParent, char *pName)
               : XolManagerC(pParent, pName, xmRowColumnWidgetClass) {}

    XolIconBarC (Widget pParent, char *pName)
               : XolManagerC(pParent, pName, xmRowColumnWidgetClass) {}

    int Create(bool pManaged);

    void Append(char *pName, char *pFile, XolCallbackProcT pProc
                                        , XolPointerT pData);

    void SetOrientation (int pOri = XmVERTICAL); // set Orientation
    void SetRowColNum   (int pNum = 1);          // set Number of columns
```

```
    void ReAlign(byte pAlign = 0);

    virtual char *GetClass()   { return "XolIconBarC"; }
};
```

8.35.2 Important Data Members

The class defines no further data members.

8.35.3 Usable Methods and Operators

XolIconBarC (XolWidgetC *pParent, char *pName);
XolIconBarC (Widget pParent, char *pName);

Description: These methods constructs the icon bar widget object.

Parameters: *pParent* - Pointer to the parent icon bar widget object
pName - Name of the icon bar widget object

Results: None

int Create(bool pManaged);

Description: This method creates the underlying *OSF/Motif* row column widget with vertical orientation and one column. It calls the method *Create* for all already constructed icon button and separator widget objects with *this* widget object as parent. The parameter *pManaged* is used as the parameter for calling the creation methods of the icon button and separator widget objects.

Parameters: *pManaged* - Determines whether the created widget should be managed automatically by its parent during the creation
TRUE - the widget is managed automatically (default)
FALSE - the widget is not managed automatically

Results: None

void Append(char *pName, char *pFile, XolCallbackProcT pProc
, XolPointerT pData);

Description: This method constructs and creates a new icon button widget object within *this* icon bar object. The icon bar object has to be created already, before this method can be used.

Parameters: *pName* - Name of the icon button widget object
pFile - Name of the label pixmap file
pProc - Callback method
pData - Callback user data

Results: None

void SetOrientation (int pOri = XmVERTICAL);
void SetRowColNum(int pNum= 1);

Description: These methods set the orientation or the number of rows/columns.

Parameters: *pOri* - Orientation of the icon bar widget object
XmVERTICAL - vertical orientation (default)
XmHORIZONTAL - horizontal orientation
pNum - If the icon bar widget object is vertical oriented, the parameter *pNum* specifies the number of columns. Otherwise it specifies the number of rows.

Results: None

void ReAlign(byte pAlign = 0);

Description: This method realigns the label pixmaps of the icon button widget object which have *this* icon bar widget object as parent.

Parameters: *pAlign* - Determines the new alignment

XmALIGNMENT_BEGINNING	- the pixmap is aligned at the left side of the label
XmALIGNMENT_CENTER	- the pixmap is aligned in the centre of the label
XmALIGNMENT_END	- the pixmap is aligned at the right side of the label

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolManagerC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolIconBarC*)

8.36 Class XolIconShellC

The class *XolIconShellC* is derived from class *XolTransientShellC* described in section 8.19. It realizes a transient shell widget object that can be opened and closed repeatedly. This is reached by managing and unmanaging the widget tree bellow the shell widget object. An icon shell widget manages an icon bar widget object within its client area that can be created automatically. To use an own icon bar widget object within the client area, the application has to construct and create the widget object instanced from a widget object class that is derived from the class *XolIconBarC*. This own icon bar widget object has to be constructed with *this* icon shell widget object as the parent before the *this* icon shell widget object is created. The pointer to the own icon bar widget object has to be stored in the data member *mClientArea* inherited from class *XolTransientShellC*, see section 8.19.

8.36.1 Class Declaration

```
#include <XOL/IconW.h>

class XolIconShellC : public XolTransientShellC
{
    bool mOpened;

  public:

    XolIconBarC *mIconBar;

    XolIconShellC (XolWmShellC *pParent, char *pName)
                 : XolTransientShellC(pParent, pName), mOpened(FALSE)
                                                    , mIconBar(NULL) {}

    XolIconShellC (Widget pParent, char *pName)
                 : XolTransientShellC(pParent, pName), mOpened(FALSE)
                                                    , mIconBar(NULL) {}

    virtual ~XolIconShellC();

    int Create    (int xPos = 30, int yPos = 30);
    int CreateRel (int xPos =  0, int yPos =  0);

    void Append(char *pName, char *pFile, XolCallbackProcT pProc
```

```
                                          , XolPointerT pData);

    static  void OpenCloseCB(Widget, XolIconShellC*, XolPointerT);
    virtual void OpenClose  ();
    virtual void Open       ();
    virtual void Close      ();

    virtual char *GetClass()   { return "XolIconShellC"; }
};
```

8.36.2 Important Data Members

bool mOpened;

Description: This data member indicates the visibility of the shell widget.

Values: *TRUE* - the shell widget and its children are managed
FALSE - the shell widget and its children are unmanaged

XolIconBarC *mIconBar;

Description: This data member holds the icon bar client widget of the icon shell wuidget object. This data member can be set by the application to its own icon bar widget object. If this data member is not set by the application when the method *Create* is called, a icon bar widget object is created automatically and can be used by the application for the definition of icon button widget objects.

Values: Pointer to the the icon shell's icon bar widget object

8.36.3 Usable Methods and Operators

XolIconShellC (XolWmShellC *pParent, char *pName);
XolIconShellC (Widget pParent, char *pName);

Description: These methods construct a icon shell widget object. After the construction a *Create* method of the basis class has to be called before the icon shell widget can be made visible. However, is an own icon bar widget object shall be used, the data member *mIconBar* has to be set by the application.

Parameters: *pParent* - Pointer to the parent widget object or the parent *Xt* widget handle, normally the application object
pName - Name of the shell widget that is also displayed as the title by default

Results: None

int CreateRel (int xPos = 0, int yPos = 0);
int Create (int xPos = 30, int yPos = 30);

Description: These methods execute the *Create* methods of the transient shell basis class. Furthermore, they create an icon bar widget object if the application has not set the data element *mIconBar* to its own icon bar widget object. After the execution the application can use the automatic icon bar widget object to construct and create icon button widget objects.

Parameters: *xPos* - shell widget x position (0 | 30 - not specified)
yPos - shell widget y position (0 | 30 - not specified)

Results: None

void Append(char *pName, char *pFile, XolCallbackProcT pProc
, XolPointerT pData);

Description: This method constructs and creates a new icon button widget object within the icon bar object. The icon bar object has to be created already, before this method can be used.

Parameters:	*pName* - Name of the icon button widget object	
	pFile - Name of the label pixmap file	
	pProc - Callback method	
	pData - Callback user data	
Results:	None	

static void OpenCloseCB (Widget, XolIconShellC*, XolPointerT);

Description: This method executes the method *OpenClose* to toggle the visibility (opened/close) of the icon shell widget object. It is a callback method and serves as the callback interface for application's callback actions and the visibility of the icon shell widget.
The method should register as the callback method for any widget object that should change the visibility of the shell widget. As user data parameter for registering *this* widget object has to be used.

Parameters: Not specified

Results: None

virtual void OpenClose ();

Description: This method toggles the visibility of the icon shell widget object.

Parameters: None

Results: None

virtual void Close ();
virtual void Open ();

Description: These methods make the icon shell widget and its children invisible/visible.

Parameters: None

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolTransientShellC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolIconShellC*)

8.36.4 Example

The example below shows the construction as well the creation of an icon shell widget object that contains an automatically created icon bar widget object. Several icon button objects are appended to the icon shell widget. Before the icon shell widget object is constructed the global application object referenced by the global variable *sGlobApp* has already to be constructed and created. In the example a class for file handling is defined. This class defines for each file handling action a static callback method that calls a special method for further processing. In the example the special methods only print out their function. The callback method corresponds to the needed callback method type with the exception that the second parameter is a pointer to an object that is not of class *XolWidgetC* or any derived class. Than, for each callback method an icon button widget object is appended to the icon shell widget object. The two possible ways to create icon buttons are shown. The first three icon button widget object are constructed and created automatically by using the method *XolIconShellC::Append*. Furthermore the callback method is registered automatically by this method. The last two icon buttons are constructed and created explicitely. Also the callback methods have to be registered explicitely. The static callback methods have to be casted to the type *XolCallbackProcT*. The file handler object is registered as callback user data. Thus the callback methods can determine the *this* pointer. Additionally, a separator widget object is constructed and created within the icon shell widget. The last icon button creation shows how the new icon shell widget object is connected to any other widget object that can produce an event. The resulting icon widget shell is shown in Fig. 7.

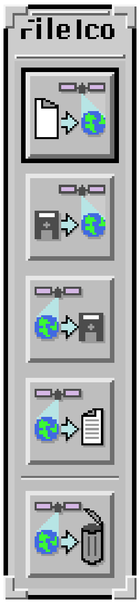

Figure 7: Icon shell widget object example

```
class MyFileHandlerC
{
  public:

    static void NewCB    (Widget, MyFileHandlerC *This, XolPointerT)
                                                        { This->New(); }

    static void LoadCB   (Widget, MyFileHandlerC *This, XolPointerT)
                                                        { This->Load(); }

    static void SaveCB   (Widget, MyFileHandlerC *This, XolPointerT)
                                                        { This->Save(); }

    static void SaveAsCB (Widget, MyFileHandlerC *This, XolPointerT)
                                                        { This->SaveAs(); }

    static void DelCB    (Widget, MyFileHandlerC *This, XolPointerT)
                                                        { This->Del(); }

    void New   (void) { printf("\nFile Handler: New");    }
    void Load  (void) { printf("\nFile Handler: Load");   }
    void Save  (void) { printf("\nFile Handler: Save");   }
    void SaveAs(void) { printf("\nFile Handler: SaveAs"); }
    void Del   (void) { printf("\nFile Handler: Del");    }
};
...

XolIconButtonC   *Icon;
XolIconShellC    *FileIconShell;
XolSeparatorC    *Sep;
MyFileHandlerC   *FileHandler;
...
```

```
FileHandler = new MyFileHandlerC();

FileIconShell = new XolIconShellC(sGlobApp, "FileIconShell");
FileIconShell->Create();

FileIconShell->Append("NewSystem"   , "SAT_NewSys32.xpm"
                                    , (XolCallbackProcT)FileHandler->NewCB
                                    , (XolPointerT)FileHandler);

FileIconShell->Append("LoadSystem"  , "SAT_LoadSys32.xpm"
                                    , (XolCallbackProcT)FileHandler->LoadCB
                                    , (XolPointerT)FileHandler);

FileIconShell->Append("SaveSystem"  , "SAT_SaveSys32.xpm"
                                    , (XolCallbackProcT)FileHandler->Save
                                    , (XolPointerT)FileHandler);

Icon = new XolIconButtonC(FileIconShell->mIconBar, "SaveSystemAs"
                                                 , "SAT_SaveSysAs32.xpm");
Icon->Create(TRUE);
Icon->AddCallback(XmNactivateCallback,(XolCallbackProcT)FileHandler->SaveAs
                                     ,(XolPointerT)FileHandler);

Sep = new XolSeparatorC(FileIconShell->mIconBar, "Separator");
Sep->Create(FALSE);

Icon = new XolIconButtonC(FileIconShell->mIconBar, "DelSystem"
                                                 , "SAT_DelSys32.xpm");
Icon->Create(TRUE);
Icon->AddCallback(XmNactivateCallback, (XolCallbackProcT)FileHandler->Del
                                     , (XolPointerT)FileHandler);

Icon = new XolIconButtonC(sGlobApp->mWorkPlace->mIconArea,
                          "FileIcon", "SAT_File48.xpm");
Icon->Create(TRUE);
Icon->AddCallback(XmNactivateCallback,
                  (XolCallbackProcT)FileIconShell->OpenCloseCB,
                  FileIconShell);
...
```

8.37 Class XolFileSelectC

The class *XolFileSelectC* encapsulates the *OSF/Motif* widget class FileSelectionBox. It realizes file selection dialogs as well file selection boxes. The difference is that in the file selection dialog the file selection box is embedded in its own shell whereas the file selection box can be embedded in any other manager widget. The file selection box allows the user to select a file that match a pattern within a specified directory.
The class *XolFileSelectC* is derived from class *XolManagerC* described in section 8.2. It registers a callback handler for the additional callback events *XmCR_OK* and *XmCR_CANCEL*. The callback event *XmCR_HELP* is handled by the callback handler of the basis class *XolManagerC*. When the callback event *XmCR_OK* is occurring, the callback manager determines the file name of the selected file and calls the dummy method. To handle the events by own callback methods the dummy methods have to be overwritten within a derived class or the own callback methods have to be registered for these events by using the method *XolWidgetC::AddCallback*.

8.37.1 Class Declaration

```
#include <XOL/FileSelB.h>
```

```
class XolFileSelectC : public XolManagerC
{
    static void CallbackHandler(Widget, XolFileSelectC*, XolPointerT);

  public:

    XolFileSelectC (XolWidgetC* pParent, char *pName)
                  : XolManagerC(pParent, pName
                                     , xmFileSelectionBoxWidgetClass) {}
    XolFileSelectC (Widget pParent, char *pName)
                  : XolManagerC(pParent, pName
                                     , xmFileSelectionBoxWidgetClass) {}

    virtual int Create(bool pManaged = TRUE, bool  pOwnShell= TRUE
                                         , char* pPath    = "./"
                                         , char* pPattern = NULL
                                         , int   xPos  = -1, int yPos  = -1
                                         , int   xSize = -1, int ySize = -1);

    virtual void CancelEvent ()      { return; }
    virtual void OkEvent     (char*) { return; }

    virtual char *GetClass()  { return "XolFileSelectC"; }
};
```

8.37.2 Important Data Members

The class defines no further data members.

8.37.3 Usable Methods and Operators

XolFileSelectC (XolWidgetC* pParent, char *pName)
XolFileSelectC (Widget pParent, char *pName)

Description: These methods construct the file selection box/dialog widget object.

Parameters: *pParent* - Pointer to the parent widget object
pName - Name of the file selection box/dialog widget object

Results: None

virtual int Create (bool pManaged = TRUE, bool pOwnShell= TRUE
, char* pPath = "./"
, char* pPattern = NULL
, int xPos = -1, int yPos = -1
, int xSize = -1, int ySize= -1);

Description: These methods create the underlying *OSF/Motif* file selection box or file selection dialog. File selection dialogs are file selection boxes embedded in an own shell. The parameter *pOwnShell* determines whether *this* widget object is managing a *OSF/Motif* file selection box or *OSF/Motif* file selection dialog.

Parameters: *pManaged* - Determines whether the created widget should be managed automatically by its parent during the creation
TRUE - the widget is managed automatically (default)
FALSE - the widget is not managed automatically

pOwnShell - Determines whether a *OSF/Motif* file selection box or *OSF/Motif* file selection dialog is created
TRUE - File selection dialog widget is created
FALSE - File selection box widget is created

	pPath	- Path of initial directory
	pPattern	- Pattern for file name pattern
	xPos	- file selection widget x position (by default -1 - not specified)
	yPos	- file selection widget y position (by default -1 - not specified)
	xSize	- file selection widget width (by default -1 - not specified)
	ySize	- file selection widget height (by default -1 - not specified)
Results:	None	

virtual void CancelEvent ();
virtual void OkEvent (char*);

Description: These methods are dummy method that are called by the callback handler.

Parameters: The parameter of method *OkEvent* delivers the file name of the selected file otherwise *NULL*.

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolManagerC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolFileSelectC*)

8.37.4 Example

The example below shows an own file selection box class derived from the file selection box class that overwrites the dummy methods for real functionality. A file selection box object is constructed as child of the global application object referenced by the global variable *sGlobApp*. After the construction, the file selection box object is created and automatic managing within an own shell. It shows all files that match the file name mask "*.xpm" within the current directory. After its creation the file selection box object reacts to the events *XmCR_HELP*, *XmCR_CANCEL*, and *XmCR_OK*. The resulting file selection box widget is shown in Fig. 8.

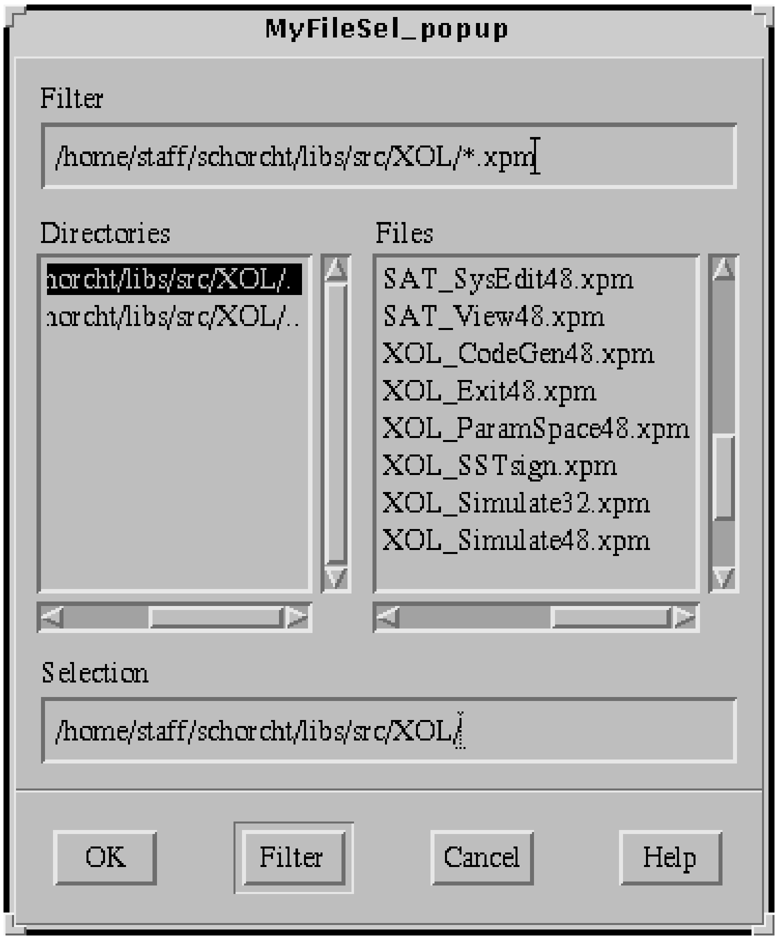

Figure 8: File selection box example

```
...

class MyFileSelBox : public XolFileSelectC
{
  public:

    MyFileSelBox(XolWidgetC *Parent): XolFileSelectC(Parent, "MyFileSel") {}

    virtual int Create() { return XolFileSelectC::Create(); }

    virtual void HelpEvent()  {
                                printf(NULL, "Help pressed \n");
                                return;
                              }

    virtual void CancelEvent() {
                                 printf(NULL, "Cancel pressed \n");
                                 UnManage();
                                 return;
                               }

    virtual void OkEvent(char *FileName) {
                                           printf(NULL, "%s selected \n"
                                                  , FileName);
```

```
                                                        UnManage();
                                                        return;
                                                     }
};

...

MyFileBox = new MyFileSelBox(sGlobApp);
MyFileBox->Create(TRUE, TRUE, "./", "*.xpm");
```

8.38 Class XolEditorWidgetC

The class *XolEditorWidgetC* belongs to the group of framework classes. It realizes a complete text editor widget by using several *XOL* widget objects. The editor widget, Fig. 9 consists of a scrolled text widget, two labelled text fields for displaying the current row and column, and a control radio box for the selection of the input mode. The labelled text fields for displaying positioning information are embedded in a frame. To arrange the different widgets a form widget is created automatically as parent.
The class *XolEditorWidgetC* is derived from class *XolTextWidgetC*. Thus, text editor widget objects provide the same functionality text widget objects. The dummy method *XolTextWidgetC::CursorMoved* was overwritten to handle the the row and column display.

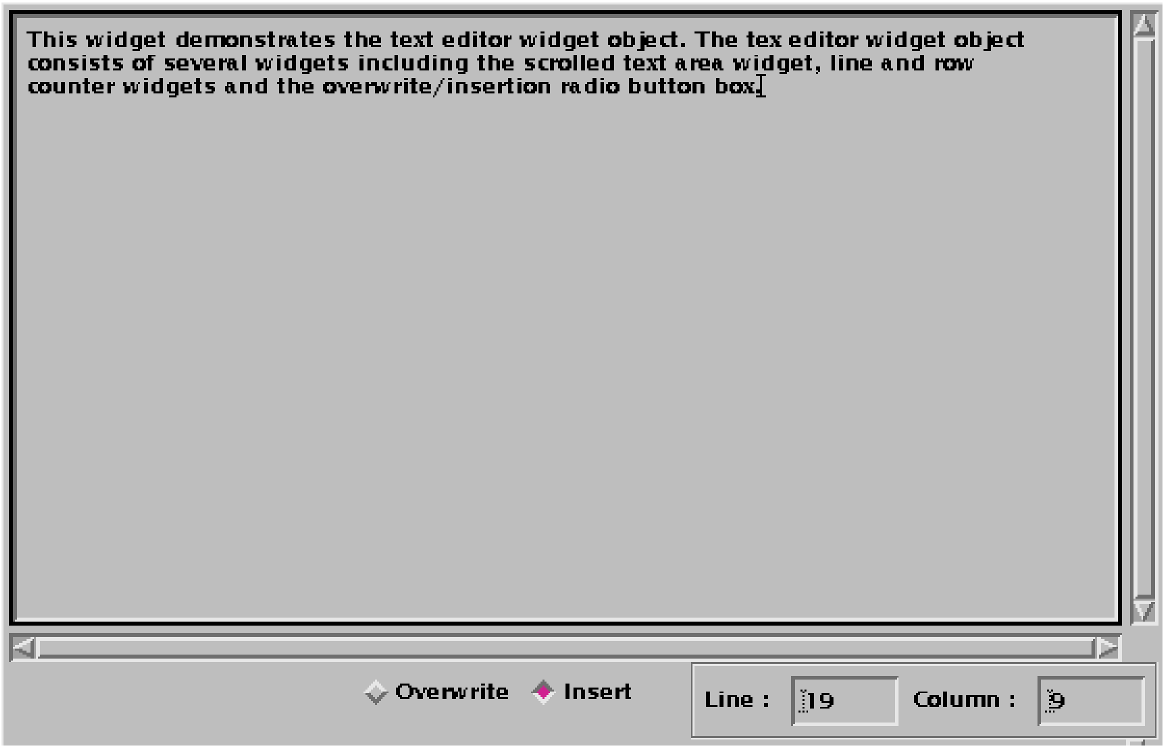

Figure 9: Text editor widget

8.38.1 Class Declaration

```
#include <XOL/EditW.h>

class XolEditorWidgetC : public XolTextWidgetC
{
    bool mInsertMode;
```

```
  public:

    XolWidgetC       *mEditForm,
                     *mLineNumForm;
    XolWidgetC       *mLineNumFrame;
    XolRadioBoxC     *mRadioBox;
    XolTextWidgetC   *mXOut,
                     *mYOut;
    XolLabelC        *mXLabelText,
                     *mYLabelText;

    XolEditorWidgetC (XolWidgetC *pParent, char *pName);
    ~XolEditorWidgetC();

    static void SetInsertMode (Widget, XolEditorWidgetC *, XolPointerT);

    virtual void CursorMoved (short x, short y);

    virtual char *GetClass ()  { return "XolEditorWidgetC"; }
};
```

8.38.2 Important Data Members

bool mInsertMode;

Description: This data member holds the insertion mode.

Values: *TRUE* - Insert mode is activated
FALSE - Overwrite mode is activated

XolWidgetC *mEditForm;

Description: This data member holds the pointer to the parent form widget object that is used to arrange the text widget object and its decorations. The parent form widget object is constructed and created automatically.

Values: Pointer to the form widget object

XolWidgetC *mLineNumFrame;

Description: This data member holds the pointer to the frame widget object that embeds the position displaying widget objects parent. The frame widget object is constructed and created automatically.

Values: Pointer to the frame widget object

XolWidgetC *mLineNumForm;

Description: This data member holds the pointer to the form widget object that is used to arrange the position display widget objects. The form widget object is constructed and created automatically.

Values: Pointer to the form widget object

XolTextWidgetC *mXOut;
XolTextWidgetC *mYOut;

Description: These data member holds the pointer to the text widget objects that are used for displaying the current position of the cursor. The widget objects are constructed and created automatically.

Values: Pointer to text widget object

XolLabelC *mXLabelText;
XolLabelC *mYLabelText;

Description: These data member holds the pointer to the label widget objects that are used as labels for position display widget objects. The widget objects are constructed and created automatically.

Values: Pointer to label widget object

XolRadioBoxC *mRadioBox;

Description: This data member holds the pointer to the radio box widget object that is used for managing toggle buttons for controlling the insertion mode. The parent radio box widget object is constructed and created automatically.

Values: Pointer to the radio box widget object

8.38.3 Usable Methods and Operators

XolEditorWidgetC (XolWidgetC *pParent, char *pName);

Description: This method constructs a text editor widget object with all decorations. First it constructs the parent form widget object and creates the the underlying *OSF/Motif* form widget. Than it constructs and creates the scrolled text widget object as well the decoration widget objects, including the labelled row and column display, the embedding frame and the insertion mode control radio box. All widget objects are constructed as well created by this method. Therefore, the parent of *this* text editor widget object has to be already created before the text editor widget object can be constructed. The Execution of the create method is not necessary.

Parameters: *pParent* - Pointer to the parent widget object
pName - Name of the file selection box/dialog widget object

Results: None

~XolEditorWidgetC ();

Description: This method destructs the test editor widget object by destroying all widget objects.

Parameters: None

Results: None

static void SetInsertMode (Widget, XolEditorWidgetC *, XolPointerT);

Description: This method is the callback method that is executed when a toggle button of the insertion mode control radio box is used.

Parameters: Callback method parameters

Results: None

virtual void CursorMoved (short x, short y);

Description: This method overwrites the callback method *CursorMoved* of the basis class to get and process the position information of the cursor.

Parameters: *x* - new x position
y - new y position

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolTextWidgetC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolEditorWidgetC*)

8.39 Class XolDrawObjC

The class *XolDrawObjC* defines an abstract basis class for the definition of all kinds of two-dimensional drawable graphic objects. A drawable graphic object is an geometrical object that can be drawn within a widget object. A drawable graphic object has an abstract method *Draw* that is called by the application to invoke the drawable object to draw itself. To define a class for real drawable graphic objects this class has to be derived. Almost it is enough to overwrite the abstract method *Draw* with real functionality.
Under *X Windows* all drawing operations need a graphics context. Therefore, each drawable object has a reference to such a graphics context. The graphics contexts specifies the attributes of the drawing operation. All drawable objects can share the same graphics context. In this case, the application has to set the values in the graphics context before each drawing operation. Since the graphics context is stored on the X server side, the graphics context has to be transfered over the X protocol from the client side to the server side before each drawing operation. This has the disadvantage that the network traffic is increasing and the performance of application is decreasing. Therefore, it is possible to create a number of graphics contexts at the server side that are shared only by few drawable objects that need almost the same drawing attributes. However, since the resources at X server side are almost limited only a limited number of graphics contexts are available. The class *XolDrawObjC* allow either own or shared graphics contexts.
Since normally not only one drawable graphic object exists, a drawable list class *XolDrawListC* for handling a number of drawable objects were developed, section 8.40. Each drawable object has to be put in such a drawable list after its creation. A drawable list is itself a drawable object that provide the method *Draw*. Thus it is possible to build up drawable object hierarchies (drawable object tree) where the nodes are drawable lists and the leafs are the drawable objects. A specialised drawable list object is the drawable root object instanced from class *XolDrawRootC*, section 8.41. This is the root of the whole drawable object tree. It has a reference to the drawing area widget object, the widget object on which the drawing operations are executed. For performance reasons each drawable object holds a reference to the drawable root object , section 8.41, and to the drawing area widget object, section 8.42. Additionally, each drawable object holds a reference to its parent. All these data members are set by the methods of the class *XolDrawListC* that have to be used to build up the drawable object hierarchy. The a sample drawable object hierarchy is depicted in Fig. 10.
A drawable object can be visible or not at a time. If a drawable object is not visible it is managed but it is not drawn.
The normal using scheme of a drawable object is

1. constructing the drawable object
2. appending it to an drawing object list
3. setting various parameters if necessary
4. making it visible

The class *XolDrawObjC* is derived from the class *SstObjectC* [Sch96].

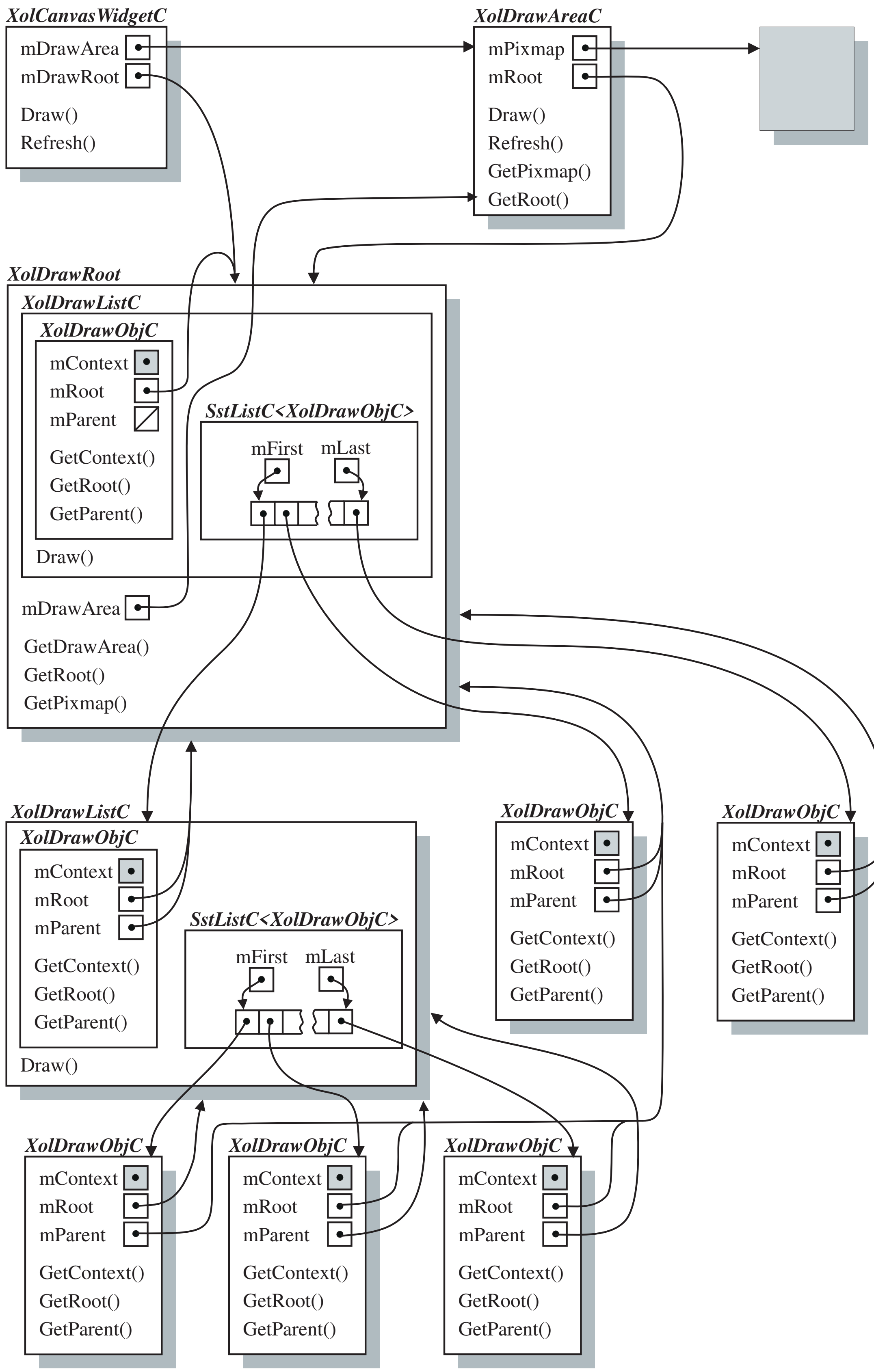

Figure 10: Sample drawable object tree

8.39.1 Class Declaration

```
#include <XOL/DrawObj.h>

class XolDrawObjC : public SstObjectC
{
    XolDrawContC* mContext;
    XolDrawListC* mParent;
    XolDrawRootC* mRoot;
    bool  mVisible;
    bool  mOwnGC;
    bool  mInvers;
    char* mFore;
    char* mBack;

    void  Remove ();
    void  Append (XolDrawListC* pOwner);
    void  Redraw (bool pRefresh = TRUE);

  public:

    XolDrawObjC  (bool pOwnGC = FALSE);
    ~XolDrawObjC ();

    void Visible(bool pVisible = TRUE, bool pRedraw = TRUE);

    virtual XolDrawListC *GetParent  () { return mParent;  }
    virtual XolDrawContC *GetContext () { return mContext; }
    virtual XolDrawRootC *GetRoot    () { return mRoot;    }

    virtual void SetParent (XolDrawListC *) { return; } // not yet
    virtual void SetRoot   (XolDrawRootC *) { return; } // not yet

    void CreateContext (XolWidgetC   *pWidget);
    void SetContext    (XolDrawContC *pContext);

    void SetGCValues(int pMask, XGCValues *pValues, bool pRedraw = TRUE);

    void SetColors (char *pFore, char *pBack, bool pRedraw = TRUE);
    void SetLine   (int pWidth, int pStyle, int pCap, int pJoin); // not yet
    void SetFill   (int pStyle, int pRule, bool pRedraw = TRUE);

    void InversIt  (bool pRedraw = TRUE);
    void SetInvers (bool pInvers = TRUE, bool pRedraw = TRUE);

    virtual void Draw() = 0;

    virtual char *GetClass()   { return "XolDrawObjC"; }

    friend class XolDrawListC;
    friend class XolDrawRootC;
};
```

8.39.2 Important Data Members

The class defines only data members that are used internally.

8.39.3 Usable Methods and Operators

XolDrawObjC (bool pOwnGC = FALSE);

Description: This method constructs a drawable object that can have either its own graphics context ore the context of another object. This is determined by the parameter *pOwnGC*. In case the object uses its own graphics context, the graphics context can be created by the application before graphics object is appended to any drawable list. Otherwise the graphs context is created automatically. In case the graphics context is shared with other drawable objects, the graphics context is set to the graphics context of the drawable list to that the drawable object is appended. By default the drawable object is not visible.

Parameters: *pOwnGC* - Determines whether an own graphics context is used or not

TRUE - drawable object uses its own graphics context
FALSE - drawable object shares the graphics context (default)

Results: None

void Visible (bool pVisible = TRUE, bool pRedraw = TRUE);

Description: This method changes the state of the visibility of the drawable object.

Parameters: *pVisible* - Determines whether an drawable is visible or not

TRUE - drawable object is visible
FALSE - drawable object is not visible

pRedraw - Determines whether the object is redrawn after the changing of visibility

TRUE - drawable object is redrawn
FALSE - drawable object is not redrawn

Results: None

virtual XolDrawContC *GetContext ();
virtual XolDrawListC *GetParent ();
virtual XolDrawRootC *GetRoot ();

Description: These methods provides the access to the important data members. By means of these methods the drawable objects anywhere in the drawable object tree have the access to the needed drawing area widget object and the drawable root object.

Parameters: None

Results:
- the graphics context that is associated with *this* drawable object
- the drawable list to which *this* drawable object belongs
- the drawable root object

void CreateContext (XolWidgetC *pWidget);

Description: With this method an own graphics context can be created explicitely for *this* drawable object. The graphics context is created independent from whether *this* drawable object should have originally its own graphics context or not. For the creation of a graphics context an already created widget is needed.

Parameters: *pWidget* - Widget that determines the display parameters for the graphics context (normally the drawing area widget is used)

Results: None

void SetContext (XolDrawContC *pContext);

Description: With this method the graphics context can be explicitely to an already existing graphics context. The graphics context is created independent from whether *this* drawable object should have originally its own graphics context or not. For the creation of a graphics context an already created widget is needed.

Parameters: *pContext* - Graphics context shared with other drawable objects

Results: None

void SetGCValues (int pMask, XGCValues *pValues, bool pRedraw = TRUE);

Description: This method sets attributes of the graphics context associated with *this* drawable object. Since the graphics context can be shared, the execution of this method have influence to future operations on other drawable objects.

Parameters:
pMask - Determines which graphics context attributes are changed [Nye92c]
pValues - Data structure that contains the new values for changing attributes
pRedraw - Determines whether the whole drawable object tree is redrawn after the changing of the graphics context attributes.
TRUE - drawable object tree is redrawn
FALSE - drawable object tree is not redrawn

Results: None

void SetColors (char *pFore, char *pBack, bool pRedraw = TRUE);
void SetFill (int pStyle, int pRule, bool pRedraw = TRUE);

Description: These methods set often changed attributes of the graphics context in a more convenient way. The methods should only be used after the appending of *this* object to a specified drawable object list.

Parameters:
pFore - Foreground colour name (see file rgb.txt)
pBack - Background colour name (see file rgb.txt)
pStyle - Determines the style of lines [Nye92c]
- LineSolid (default)
- LineOnOffDash
- LineDoubleDash

pRule - Determines the filling rule for polygons [Nye92c]
- EvenOddRule (default)
- WindingRule

pRedraw - Determines whether the whole drawable object tree is redrawn after the changing of the graphics context attributes.
TRUE - drawable object tree is redrawn
FALSE - drawable object tree is not redrawn

Results: None

void Inversion (bool pRedraw = TRUE);

Description: This method invert *this* drawable object by exchanging the fore- and background colour for redrawing operations. The method can be used to realize the inverse displaying of the object if it is selected.

Parameters:
pRedraw - Determines whether the whole drawable object tree is redrawn after the changing of the graphics context attributes.
TRUE - drawable object tree is redrawn
FALSE - drawable object tree is not redrawn

Results: None

void SetInvers (bool pInvers = TRUE, bool pRedraw = TRUE);

Description: This method set the inversion mode of *this* drawable object by exchanging the fore- and background colour for redrawing operations. The method can be used to realize the inverse displaying of the object if it is selected.

Parameters:
pInvers - Determines whether *this* drawable object is drawn inverse or not
TRUE - drawable object is drawn inverse
FALSE - drawable object is drawn not inverse

pRedraw - Determines whether the whole drawable object tree is redrawn after the changing of the graphics context attributes.

TRUE - drawable object tree is redrawn
FALSE - drawable object tree is not redrawn

Results: None

virtual void Draw () = 0;

Description: This methods is abstract. It defines the method interface for the draw method that is overwritten by all specialised derived drawable object classes. The whole drawable object tree is drawn by calling this method of the drawable root object. The drawable root object calls this method for all drawable objects contained in its list. These drawable objects call this method for all drawable object in their lists and so on. Thus, the whole drawable object tree is drawn beginning with the drawable root object.

Parameters: None

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *SstObjectC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolDrawObjC*)

8.39.4 Example

The example below shows how the drawable object class *XolDrawObjC* has to be derived for realisation of own drawable objects. In the example a rectangle is realised as drawable. The most important task for realising own drawable objects is to overwrite the virtual dummy method *Draw* with real functionality. This method uses functions of the *Xlib* to draw a rectangle drawable object within the drawing buffer referenced by the drawable root object. Since the class is derived from class *XolDrawObjC* it can use its methods to get an access to the drawing buffer. Furthermore it inherits all data members of the class *XolDrawObjC*. Thus it have an access to the relevant graphics context.

```
class MyRectangleC : public XolDrawObjC
{
    int  x, y, dx, dy;
    bool mFilled;

  public:

    MyRectangleC (int px, int py, int pdx, int pdy, bool pFilled = FALSE
                                                  , bool pOwnGC  = FALSE)
                : XolDrawObjC(pOwnGC), x(px), y(py), dx(pdx), dy(pdy)
                                                   , mFilled(pFilled) {}

    virtual void Draw();
    virtual char *GetClass()   { return "XolRectangleC"; }
};

void XolRectangleC::Draw()
{
  XolDrawRootC *tRoot   = GetRoot();
  XolDrawContC *tContext= GetContext();
```

```
  if (mFilled)
    XFillRectangle(tContext->mDisplay, tRoot->GetPixmap(), tContext->mGC
                                  , x, y, dx, dy);
  else
    XDrawRectangle(tContext->mDisplay, tRoot->GetPixmap(), tContext->mGC
                                  , x, y, dx, dy);
};
```

8.40 Class XolDrawListC

The class *XolDrawListC* realizes drawable object lists. A drawable object list is a specialised drawable object that is capable to manage a number of another drawable objects, so-called child drawable objects. It overwrites the abstract method *Draw* to call this method for all child drawable objects. The drawable object list itself is not visible. Its only one purpose is the managing of its child drawable objects. Each drawable object has to be put in such a drawable list after its creation. Thus it is possible to build up drawable object hierarchies, so-called drawable object tree where the nodes are drawable object lists and the leafs are the drawable objects. A specialised drawable list object is the drawable root object instanced from class *XolDrawRootC*, section 8.41. The drawing order within a drawable object list is from the first to the last element, that is the last element in the list is the top object which is seen.
The class *XolDrawListC* is derived from class *XolDrawObjC* as well from class *SstListC* [Sch96].

8.40.1 Class Declaration

```
#include <XOL/DrawObj.h>

class XolDrawListC : public XolDrawObjC, public SstListC<XolDrawObjC>
{
  public:

    XolDrawListC() : SstListC<XolDrawObjC> () {}

    void Remove   (XolDrawObjC *pObj, bool pRedraw = TRUE);
    void AddFore  (XolDrawObjC *pObj, bool pVisible= FALSE
                                    , bool pRedraw = TRUE);
    void AddBack  (XolDrawObjC *pObj, bool pVisible= FALSE
                                    , bool pRedraw = TRUE);
    void MoveFore (XolDrawObjC *pObj, int  pNum = 1, bool pRedraw = TRUE);
    void MoveBack (XolDrawObjC *pObj, int  pNum = 1, bool pRedraw = TRUE);

    virtual void Draw();
};
```

8.40.2 Important Data Members

The class defines no further data members.

8.40.3 Usable Methods and Operators

XolDrawListC ();

Description: This method constructs a drawable object list by using the basis class constructor. It have nor really functionality.

Parameters: None

Results: None

void Remove (XolDrawObjC *pObj, bool pRedraw = TRUE);

Description: This method removes the drawable object from the *this* list.

Parameters: *pRedraw* - Determines whether the whole drawable object tree is redrawn after the removing of the drawable object

TRUE - drawable object tree is redrawn
FALSE - drawable object tree is not redrawn

Results: None

void AddFore (XolDrawObjC *pObj, bool pVisible = FALSE
, bool pRedraw= TRUE);
void AddBack (XolDrawObjC *pObj, bool pVisible= FALSE
, bool pRedraw = TRUE);

Description: These methods add the given drawable object at the front or the rear of *this* drawable object list.

Parameters: *pObj* - Pointer to the drawable object
pVisible - Determines whether the drawable is immediately visible

TRUE - drawable object is visible
FALSE - drawable object is not visible

pRedraw - Determines whether the whole drawable object tree is redrawn after the execution of this method

TRUE - drawable object tree is redrawn
FALSE - drawable object tree is not redrawn

Results: None

void MoveFore (XolDrawObjC *pObj, int pNum = 1, bool pRedraw = TRUE);
void MoveBack (XolDrawObjC *pObj, int pNum = 1, bool pRedraw = TRUE);

Description: These methods change the position of a given object within *this* drawable object list.

Parameters: *pObj* - Pointer to the drawable object
pNum - Number of position in the direction of foreground or background
pRedraw - Determines whether the whole drawable object tree is redrawn after the execution of this method

TRUE - drawable object tree is redrawn
FALSE - drawable object tree is not redrawn

Results: None

virtual void Draw ();

Description: This method overwrites the abstract method *Draw* of the basis class. The method calls this method for all child drawable object.

Parameters: None

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolDrawObjC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolDrawListC*)

8.41 Class XolDrawRootC

The class *XolDrawRootC* realizes the root object for the drawing object tree. The root object contains a reference to the drawing area widget object that is used as the drawing area. The drawing area widget object has to be from class *XolDrawAreaC*. It is a specialisation of the drawing object list class. The root object is responsible for the background filling of the associated drawing area widget object. Therefore, the full restoration of the drawing area is reached by using the method *Draw* of the root object. This method fills the background of the drawing area widget object and calls the method *Draw* of its children.

8.41.1 Class Declaration

```
#include <XOL/DrawObj.h>

class XolDrawRootC : public XolDrawListC
{
    XolDrawAreaC *mDrawArea;

  public:

    XolDrawRootC(XolDrawAreaC *pDrawArea);

    XolDrawAreaC*  GetDrawArea()  { return mDrawArea; }
    XolDrawRootC*  GetRoot    ()  { return this;      }
    Pixmap         GetPixmap  ();

    virtual void Draw();

    virtual char *GetClass()   { return "XolDrawRootC"; }
};
```

8.41.2 Important Data Members

XolDrawAreaC *mDrawArea;

Description: This data member holds the pointer to the drawing area widget object that is used for displaying the drawable objects.

Values: Pointer to drawing area widget object class

8.41.3 Usable Methods and Operators

XolDrawRootC (XolDrawAreaC *pDrawArea);

Description: This method constructs the drawable root object. The drawable root object has always a graphics context. It is created automatically during the construction.

Parameters: *pDrawArea* - Pointer to the drawing area widget object that is used as the drawing area.

Results: None

XolDrawAreaC* GetDrawArea ();

Description: This method delivers the pointer to the drawing area widget object. Since, all drawable objects have a reference to *this* drawable root object, all drawable object have also access to the drawing area widget object over *this* drawable root object.

Parameters: None

Results: Pointer to the drawing area widget object

XolDrawRootC* GetRoot ();

Description: This method delivers *this* . It overwrites the method *GetRoot* of the basis class.

Parameters: None

Results: Pointer to *this* object

Pixmap GetPixmap ();

Description: This method returns the buffer on which the graphic operations are executed. Normally, each graphics operation is executed on a buffer.

Parameters: None

Results: Pointer to the buffer

virtual void Draw ();

Description: The method overwrites this method of the basis class. It fills the drawing area widget with background and foreground colours. Than it calls the method *Draw* for all objects, that belongs to the drawable root object

Parameters: None

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolDrawListC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolDrawRootC*)

8.42 Class XolDrawAreaC

The class *XolDrawAreaC* realizes drawing area widget object. A drawing area widget is used as canvas for drawing operations. A drawing area widget object holds a reference to the drawable object tree that is associated with it. To draw the whole drawable object tree, the method *Draw* of the drawing area widget object is used. To avoid that the drawing operation is traceable by the user as well to avoid that the complete drawable object tree has to be redrawn when parts or the whole of the drawing area widget object will be exposed, the drawing operations are normally done on a buffer in the background. Therefore, the drawing area widget object holds such a buffer on which the drawing operation can be done.
The class *XolDrawAreaC* registers automatically a callback handler for *OSF/Motif* callback events *XmCR_EXPOSE*, *XmCR_RESIZE*, and *XmCR_INPUT*, that indicate the necessity of redrawing or refreshing the contents of the drawing area widget object. The class *XolDrawAreaC* defines for each of the mentioned events a virtual dummy method that is called by the callback handler. However, the callback method *Expose* provide really functionality. It refreshes the drawing area widget by copying the contents from the drawing area buffer. To handle these events with own callback methods, the virtual dummy methods have to be overwritten within a derived class or the callback methods have to be registered for the callback events using the method *XolWidgetC::AddCallback*.

8.42.1 Class Declaration

```
#include <XOL/DrawObj.h>

class XolDrawAreaC : public XolWidgetC
{
    XolDrawRootC *mRoot;       // Pointer to the graphical object tree root
    XolPixmapC   *mPixmap;     // Pixmap for double buffering

    static void CallbackHandler(Widget, XolDrawAreaC*, XolPointerT);

  public:

    // Constructors/Destructors

    XolDrawAreaC (XolWidgetC *pParent, char *pName);
    XolDrawAreaC (Widget pParent, char *pName);

    ~XolDrawAreaC ();
```

```
    // Creation methods

    virtual int Create (bool pManaged = TRUE, int  xSize = 800
                                            , int  ySize = 600);
    // Drawing methods

    virtual void Refresh(int xPos = 0, int yPos = 0, int xSize = 0
                                                   , int ySize = 0);
    virtual void Draw();

    // Methods that handle data members

    XolDrawRootC*  GetRoot   () { return mRoot; }
    Pixmap         GetPixmap () { return mPixmap->mPixmap; }

    void SetRoot   (XolDrawRootC *) { return; }   // not yet implemented
    void SetPixmap (XolPixmapC   *) { return; }   // not yet implemented

    void SetColors(char *pFore, char *pBack);

    // Callback methods

    virtual void Exposure (XEvent *) ;               // xmNexposeCallback
    virtual void Input    (XEvent *) { return; }     // xmNinputCallback
    virtual void Resize   ()         { return; }     // xmNresizeCallback

    // Miscellaneous methods

    virtual char *GetClass()  { return "XolDrawAreaC"; }
};
```

8.42.2 Important Data Members

XolDrawRootC *mRoot;

Description: This data member holds the pointer to the drawable root object of the drawable object tree. The drawable root object is created automatically during the creation of the drawing area widget object.

Values: Pointer to the drawable root object

XolPixmapC *mPixmap;

Description: This data member holds the pointer to the drawing buffer of the drawing area widget. This buffer has to be used for all drawing operations. The drawing buffer is created automatically during the creation of the drawing area widget.

Values: Pointer to the drawing buffer

8.42.3 Usable Methods and Operators

XolDrawAreaC (XolWidgetC *pParent, char *pName);
XolDrawAreaC (Widget pParent, char *pName);

Description: These methods constructs the drawing area widget object. The parent should be from any scrolled windows widget class like *XolCanvasWidgetC*, section 8.43

Parameters: *pParent* - Pointer to the parent widget object
pName - Name of the drawing area widget object

Results: None

virtual int Create (bool pManaged = TRUE, int xSize = 800 , int ySize = 600);

Description: This method creates the underlying *OSF/Motif* drawing area widget, the drawing area background buffer and the drawable root object associated to *this* drawing area widget object. Furthermore, it registers the callback handler for the *OSF/Motif* callback events *XmCR_EXPOSE*, *XmCR_RESIZE*, and *XmCR_INPUT*.

Parameters:
- *pManaged* - Determines whether the created widget should be managed automatically by its parent during the creation
  - *TRUE* - the widget is managed automatically (default)
  - *FALSE* - the widget is not managed automatically
- *xSize* - Width of the drawing area widget as well the drawing background buffer
- *ySize* - Height of the drawing area widget as well the drawing background buffer

Results: None

virtual void Draw ();

Description: This method calls the method *Draw* of the drawable root object associated to *this* drawing area widget object to draw the complete drawable object tree.

Parameters: None

Results: None

virtual void Refresh (int xPos = 0, int yPos = 0, int xSize = 0 , int ySize = 0);

Description: This method copies the area specified by parameters from the drawing background buffer to the drawing area widget.

Parameters:
- *xPos* - x position of the copied area
- *yPos* - y position of the copied area
- *xSize* - Width of the copied area (0 - the remaining measured from *xPos*)
- *ySize* - Height of the copied area (0 - the remaining measured from *yPos*)

Results: None

XolDrawRootC* GetRoot ();

Description: This method delivers the pointer to the drawable root object.

Parameters: None

Results: Pointer to drawable root object

Pixmap GetPixmap ();

Description: This method delivers the pointer to the drawing background buffer.

Parameters: None

Results: Pointer to drawable root object

void SetColors (char *pFore, char *pBack);

Description: This method sets foreground and the background colour of the drawing area widget object. Since the background is filled by the drawable root object and the root object determines the colours for all drawable objects without own graphics context, the method set really the colours of the drawable root object. The drawable object tree is not redrawn.

Parameters:
- *pFore* - Foreground colour name (see file rgb.txt)
- *pBack* - Background colour name (see file rgb.txt)

Results: None

virtual void Exposure (XEvent *);
virtual void Input (XEvent *);
virtual void Resize ();

Description: These methods are dummy methods that are called by the callback handler of *this* drawing area. The callback method *Expose* provide really functionality. It refreshes the drawing area widget by copying the contents from the drawing area buffer. To handle the callbacks by own callback methods, either these methods have to be overwritten within a derived class or the own callback methods have to be registered for the callback events by using the method *XolWidgetC::AddCallback*.

Parameters: The parameter if any delivers the coordinates for the event

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolWidgetC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolDrawAreaC*)

8.43 Class XolCanvasWidgetC

The class *XolCanvasWidgetC* realizes a canvas widget object. A canvas widget object is a widget that is used for drawing operations to display graphical objects. It consist of a scrolled window widget and an embedded drawing area widget. The scrolled window widget is used to manage a viewing area as a part of a bigger drawing area. The class provides the application with the interface to the embedded drawing area widget object and the drawable object tree. To invoke the drawing of the drawable object tree or the refreshing of parts of the drawing area widget objects, the methods of this class have to be used. The class *XolCanvasWidgetC* is derived from class *XolManagerC* described in section 8.2.

8.43.1 Class Declaration

```
#include <XOL/CanvasW.h>

class XolCanvasWidgetC : public XolManagerC
{

  public:

    XolDrawAreaC *mDrawArea;    // Drawing Area widget object
    XolDrawRootC *mDrawRoot;    // Drawable root object

    XolCanvasWidgetC (XolWidgetC *pParent, char *Name);
    XolCanvasWidgetC (Widget pParent, char *Name);

    virtual ~XolCanvasWidgetC();

    virtual int Create (bool pManaged = TRUE, int  pDrawXSize = 800
                                            , int  pDrawYSize = 600
                                            , int  pViewXPos  = 0
                                            , int  pViewYPos  = 0
                                            , int  pViewXSize = 0
                                            , int  pViewYSize = 0);

    void SetColors (char *pFore, char *pBack);
```

```
    virtual void Draw    ();
    virtual void Refresh (int xPos = 0, int yPos = 0, int xSize = 0
                                                    , int ySize = 0);

    virtual char *GetClass ()  { return "XolCanvasWidgetC"; }

};
```

8.43.2 Important Data Members

XolDrawAreaC *mDrawArea;

Description: This data member holds the pointer to the embedded drawing area widget object. The drawing area widget object is constructed and created automatically during the creation of the canvas widget object.

Values: Pointer to the drawing area widget object

XolDrawRootC *mDrawRoot;

Description: This data member holds for faster access a reference to the drawable root object. The drawable root object is automatically created during the creation of the drawing area widget object.

Values: Pointer to the drawable root object

8.43.3 Usable Methods and Operators

XolCanvasWidgetC (XolWidgetC *pParent, char *Name);
XolCanvasWidgetC (Widget pParent, char *Name);

Description: These methods construct a canvas widget object.

Parameters: *pParent* - Pointer to the parent widget object
pName - Name of the canvas widget object

Results: None

virtual ~XolCanvasWidgetC ();

Description: This method destructs the canvas area widget object. It destroys the embedded drawing area object.

Parameters: None

Results: None

virtual int Create (bool pManaged = TRUE, int pDrawXSize = 800
, int pDrawYSize = 600
, int pViewXPos = 0
, int pViewYPos = 0
, int pViewXSize = 0
, int pViewYSize = 0);

Description: This method creates the underlying *OSF/Motif* scrolled window widget as the viewing area and the embedded drawing area widget object.

Parameters:
pManaged - Determines whether the created widget should be managed automatically by its parent during the creation
TRUE - the widget is managed automatically (default)
FALSE - the widget is not managed automatically
pDrawXSize - Width of the drawing area
pDrawYSize - Height of the drawing area
pViewXPos - x position of the viewing area within the drawing area

	pViewYPos	- y position of the viewing area within the drawing area
	pViewXSize	- Width of the viewing area (0 - the remaining measured from *pViewXPos*)
	pViewYSize	- Height of the viewing area (0 - the remaining measured from *pViewYPos*)
Results:	None	

virtual void Draw ();

Description: This method calls the method *Draw* of the embedded drawing area object and thus of the drawable root object to draw the complete drawable object tree.

Parameters: None

Results: None

virtual void Refresh (int xPos = 0, int yPos = 0, int xSize = 0 , int ySize = 0);

Description: This method calls the method *Refresh* of the embedded drawing area widget object to copy the area specified by parameters from the drawing background buffer to the drawing area widget.

Parameters:
xPos - x position of the copied area
yPos - y position of the copied area
xSize - Width of the copied area (0 - the remaining measured from *xPos*)
ySize - Height of the copied area (0 - the remaining measured from *yPos*)

Results: None

void SetColors (char *pFore, char *pBack);

Description: This method sets foreground and the background colour of the embedded drawing area widget object by using the method *SetColors* of the drawing area widget object.

Parameters:
pFore - Foreground colour name (see file rgb.txt)
pBack - Background colour name (see file rgb.txt)

Results: None

virtual char *GetClass ();

Description: This method overwrites the virtual method of the base class *XolWidgetC*. It delivers the class identifier.

Parameters: None

Results: Class name (by default *XolCanvasWidgetC*)

8.43.4 Example

Te example below shows the using of a canvas widget object for drawing a drawable object tree. The canvas widget is used wihtin an own top-level shell widget object. It constructs and creates a drawing area widget object child and thus a drawable root object automatically. Furthermore, it used together with three other widget objects instanced from the scale box object class *XolScaleBoxC* described in section 8.33. These widget objects are arranged within a form widget. Since the form widget is a common *OSF/Motif* widget class the class *XolManagerC* is used to realise the form widget object. The class *SatObserverViewC* is a class from the satellite communcition simulation library developed in our department [not yet documented] ;-) It defines objects that realize different kind of satellite observer view. The creation of a satellite observer view object build up a drawable object tree that represents the observer view. It is derived from class *XolDrawListC* and has therefore the method *Draw*that carries out the drawing of the view. By using the method *AddFore* of the canvas associated drawable root objet, the satellite observer view object is added to it and also drawed. Because the satellite view object is getting the graphics context when it is added to the drawable root object, the foreground and background colours can only be set after the satellite view object is added to the drawable root object. The result of the example is shown in Fig. 11.

```
XolWmShellC      *Shell;
XolManagerC      *Form;
XolCanvasWidgetC *Canvas;
XolScaleBoxC     *Elevation;
XolScaleBoxC     *Longitude;
XolScaleBoxC     *Latitude;
SatObserverViewC *View;

...


Shell = new XolWmShellC(sGlobApp, "Observer");
Shell->Create(FALSE);
Shell->SetTitle("Observer View");

Form = new XolManagerC(Shell->mClientArea, "Form", xmFormWidgetClass);
Form->Create();

Canvas = new XolCanvasWidgetC(Form,"Canvas");
Canvas->Create(TRUE, Size, Size);
Canvas->SetColors("white","black");
Canvas->Set(XmNleftAttachment  ,XmATTACH_FORM,
            XmNrightAttachment ,XmATTACH_FORM,
            XmNtopAttachment   ,XmATTACH_FORM,
            NULL);

Longitude = new XolScaleBoxC(Form,"Longitude");
Longitude->Create(TRUE, TRUE, TRUE, XOL_MOD_PREC, 7
                                  , XmHORIZONTAL, -1800, 1800, 1);
Longitude->SetValue((float)0.0);
Longitude->Set(XmNtopAttachment ,XmATTACH_WIDGET,
               XmNtopWidget     ,Canvas->XtWidget,
               NULL);

Latitude = new XolScaleBoxC(Form,"Latitude   ");
Latitude->Create(TRUE, TRUE, TRUE, XOL_MOD_PREC, 7
                                  , XmHORIZONTAL, -900, 900, 1);
Latitude->SetValue((float)0.0);
Latitude->Set(XmNtopAttachment ,XmATTACH_WIDGET,
              XmNtopWidget     ,Longitude->XtWidget,
              NULL);

Elevation = new XolScaleBoxC(Form,"Elevation ");
Elevation->Create(TRUE, TRUE, TRUE, XOL_MOD_PREC, 7
                                  , XmHORIZONTAL, 0, 900, 1);
Elevation->SetValue((float)0.0);
Elevation->Set(XmNtopAttachment ,XmATTACH_WIDGET,
               XmNtopWidget     ,Latitude->XtWidget,
               NULL);

View = new SatObserverViewC();
View->SetSize(Size);
Canvas->DrawRoot->AddFore(View,TRUE);
View->SetColors("LightBlue","blue3");

Shell->Manage();
```

...

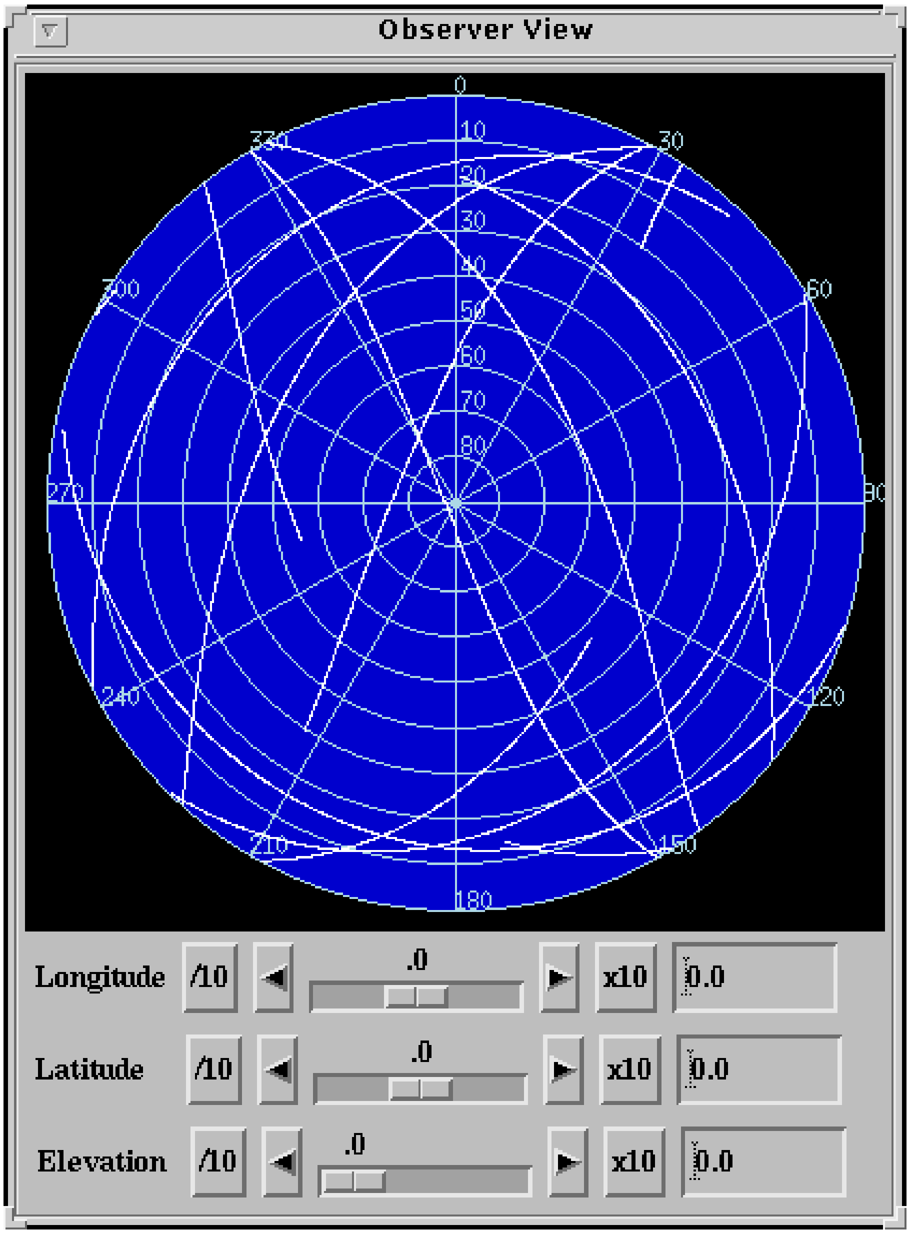

Figure 11: Canvas widget object example

9 Example

In the following a more complex example is shown without any comments. It is simply the result of merging the most examples through this document. It is written for test purposes and has no real functionality. For a more realistic application, but also a very complex application, that uses the *XOL* extensively you can have a look at the satellite communications simulation toolkit [not yet] developed in our department ;-)

```
#include <stdlib.h>
#include <stdio.h>

#include <Xm/RowColumn.h>
```

```
#include <Xm/MainW.h>
#include <Xm/DrawingA.h>
#include <Xm/MwmUtil.h>
#include <Xm/ScrolledW.h>
#include <Xm/Frame.h>

#include <XOL/XOL.h>

// ----------------------------------------------------------------------------

#define MyFonts \
"*fontList: -*-times-medium-r-*-*-14-*=XolDEFAULT_FONT,\
-*-times-bold-i-*-*-18-*-*-*-*-*-*-*=XOL_INFO_BIG_ITALIC,\
-*-times-bold-r-*-*-18-*-*-*-*-*-*-*=XOL_INFO_BIG,\
-*-times-medium-i-*-*-14-*-*-*-*-*-*-*=XOL_INFO_NORMAL_ITALIC,\
-*-times-medium-r-*-*-14-*-*-*-*-*-*-*=XOL_INFO_NORMAL"

#define MyColorList "*selectColor: violet red"

#define SorryText \
"Sorry, this feature is not yet implementd. \
\nTry it later again!"

#define InfoText1 "SDA - System level Design Automation"
#define InfoText2 "\nTool Kit Version 1.0"
#define InfoText3 "\n\nTechnical University of Ilmenau"
#define InfoText4 "\nDepartment of Computer Science and Automation"
#define InfoText5 "\n\ne-mail: sst@theoinf.tu-ilmenau.de"

// ----------------------------------------------------------------------------

class MyFileSelectC : public XolFileSelectC
{
  public:

    MyFileSelectC (XolWidgetC *Parent): XolFileSelectC(Parent,"MyFileSel") {}

    virtual int Create() { XolFileSelectC::Create(); return 0; }

    virtual void HelpEvent()  {
                               printf(NULL, "Help pressed \n");
                               return;
                             }

    virtual void CancelEvent() {
                                printf(NULL, "Cancel pressed \n");
                                UnManage();
                                return;
                               }

    virtual void OkEvent(char *FileName) {
                                          printf(NULL, "%s selected \n"
                                                     , FileName);
                                          UnManage();
                                          return;
                                         }
};
```

```
class MyFileHandlerC
{
  public:

    static void NewCB    (Widget, MyFileHandlerC *This, XolPointerT)
                                                        { This->New(); }

    static void LoadCB   (Widget, MyFileHandlerC *This, XolPointerT)
                                                        { This->Load(); }

    static void SaveCB   (Widget, MyFileHandlerC *This, XolPointerT)
                                                        { This->Save(); }

    static void SaveAsCB (Widget, MyFileHandlerC *This, XolPointerT)
                                                        { This->SaveAs(); }

    static void DelCB    (Widget, MyFileHandlerC *This, XolPointerT)
                                                        { This->Del(); }

    void New   (void) { printf("\nFile Handler: New");   fflush(stdout); }
    void Load  (void) { printf("\nFile Handler: Load");  fflush(stdout); }
    void Save  (void) { printf("\nFile Handler: Save");  fflush(stdout); }
    void SaveAs(void) { printf("\nFile Handler: SaveAs");fflush(stdout); }
    void Del   (void) { printf("\nFile Handler: Del");   fflush(stdout); }
};

// ----------------------------------------------------------------------

void SorryMsgBox (Widget, XolWidgetC *Parent, XolPointerT);
void MenuBarInit (XolMenuC*);
void Input  (Widget, XolMenuC*    , XmDrawingAreaCallbackStruct *);
void Quit   (Widget, XolWidgetC*  , XolPointerT);
void Open   (Widget, XolWidgetC*  , XolPointerT);
void Style  (Widget, XolMenuItemC*, XolPointerT);
void Browse (Widget, XolWidgetC*  , XolPointerT);
void Info   (Widget, XolWidgetC*  , XolPointerT);

// ----------------------------------------------------------------------

int main(int argc, char *argv[])
{
    XolPopupMenuC    *Popup;
    XolIconButtonC   *Icon;
    XolMainWinC      *WorkPlace;
    XolSeparatorC    *Sep;
    XolCanvasWidgetC *Canvas;
    XolIconShellC    *FileIconShell,
                     *ViewIconShell;
    MyFileHandlerC   *FileHandler;

    Display *Dis;
    Screen  *Scr;
    Window  Root;

    sGlobApp = new XolApplicationC("SDA_Tool_Kit","SDA", argc, argv);
    sGlobApp->AddFallbackRes(MyFonts);
```

```
sGlobApp->AddFallbackRes(MyColorList);
sGlobApp->Create();

Dis = sGlobApp->GetDisplay();
Scr = sGlobApp->GetScreen();
Root= RootWindowOfScreen(Scr);
sGlobApp->dprintf("Display: %08X, Screen: %08X, RootWin: %08X"
                               , (int)Dis, (int)Scr, Root);

sGlobApp->mWorkPlace = new XolMainWinC(sGlobApp->mClientArea,"Main");

WorkPlace = sGlobApp->mWorkPlace;
WorkPlace->Create(TRUE, TRUE, TRUE, FALSE, TRUE, XmVERTICAL);

Canvas = new XolCanvasWidgetC(WorkPlace->mPaned,"Canvas");
Canvas->Create(TRUE, 700, 350);
Canvas->SetColors("black","blue3");

WorkPlace->mWorkArea = Canvas;

WorkPlace->mLogoArea = new XolPushButtonC(WorkPlace->mLogoFrame, "Logo"
                                                               , XmPIXMAP);
WorkPlace->mLogoArea->Create(TRUE,"sign.xbm");
WorkPlace->mLogoArea->AddCallback(XmNactivateCallback,
                                  (XolCallbackProcT)Info,
                                  WorkPlace->mLogoArea);
WorkPlace->Manage();

FileHandler = new MyFileHandlerC();

FileIconShell = new XolIconShellC(sGlobApp, "FileIconShell");
FileIconShell->Create();

FileIconShell->Append("NewSystem"   , "SAT_NewSys32.xpm"
                                    , (XolCallbackProcT)FileHandler->NewCB
                                    , (XolPointerT)FileHandler);

FileIconShell->Append("LoadSystem"  , "SAT_LoadSys32.xpm"
                                    , (XolCallbackProcT)FileHandler->LoadCB
                                    , (XolPointerT)FileHandler);

FileIconShell->Append("SaveSystem"  , "SAT_SaveSys32.xpm"
                                    , (XolCallbackProcT)FileHandler->Save
                                    , (XolPointerT)FileHandler);

Icon = new XolIconButtonC(FileIconShell->mIconBar, "SaveSystemAs"
                                                 , "SAT_SaveSysAs32.xpm");
Icon->Create(TRUE);
Icon->AddCallback(XmNactivateCallback,(XolCallbackProcT)FileHandler->SaveAs
                                     ,(XolPointerT)FileHandler);

Sep = new XolSeparatorC(FileIconShell->mIconBar, "Separator");
Sep->Create(FALSE);

Icon = new XolIconButtonC(FileIconShell->mIconBar, "DelSystem"
                                                 , "SAT_DelSys32.xpm");
Icon->Create(TRUE);
```

```
Icon->AddCallback(XmNactivateCallback, (XolCallbackProcT)FileHandler->Del
                                    , (XolPointerT)FileHandler);


ViewIconShell = new XolIconShellC(sGlobApp, "ViewIconShell");
ViewIconShell->Create();
ViewIconShell->Append("MapView","SAT_MapView32.xpm"
                               ,SorryMsgBox,(XolPointerT)sGlobApp);
ViewIconShell->Append("CirView","SAT_CirView32.xpm"
                           ,SorryMsgBox,(XolPointerT)sGlobApp);
ViewIconShell->Append("CirView","SAT_CirView32.xpm"
                               ,SorryMsgBox,(XolPointerT)sGlobApp);
ViewIconShell->Append("EciView","SAT_EciView32.xpm"
                               ,SorryMsgBox,(XolPointerT)sGlobApp);
ViewIconShell->Append("EcfView","SAT_EcfView32.xpm"
                               ,SorryMsgBox,(XolPointerT)sGlobApp);
ViewIconShell->Append("ObsView","SAT_ObsView32.xpm"
                               ,SorryMsgBox,(XolPointerT)sGlobApp);

Icon = new XolIconButtonC(WorkPlace->mIconArea,"FileIcon"
                                              ,"SAT_File48.xpm");
Icon->Create(TRUE);
Icon->AddCallback(XmNactivateCallback,
                  (XolCallbackProcT)FileIconShell->OpenCloseCB,
                  FileIconShell);

Icon = new XolIconButtonC(WorkPlace->mIconArea,"EditIcon"
                                              ,"SAT_SysEdit48.xpm");
Icon->Create(TRUE);
Icon->AddCallback(XmNactivateCallback,Browse,(XolPointerT)sGlobApp);

Icon = new XolIconButtonC(WorkPlace->mIconArea,"ViewIcon"
                                              ,"SAT_View48.xpm");
Icon->Create(TRUE);
Icon->AddCallback(XmNactivateCallback,
                  (XolCallbackProcT)ViewIconShell->OpenCloseCB,
                  ViewIconShell);

Icon = new XolIconButtonC(WorkPlace->mIconArea,"SimuIcon"
                                              ,"SAT_Simulate48.xpm");
Icon->Create(TRUE);
Icon->AddCallback(XmNactivateCallback,(XolCallbackProcT)Open,Icon);

Sep = new XolSeparatorC(WorkPlace->mIconArea,"Sep1");
Sep->Create(FALSE);

Icon = new XolIconButtonC(WorkPlace->mIconArea,"ExitIcon"
                                              ,"SAT_Exit48.xpm");
Icon->Create(TRUE);
Icon->AddCallback(XmNactivateCallback,(XolCallbackProcT)Open);

WorkPlace->Manage();

MenuBarInit(WorkPlace->mMenuBar);

Popup = new XolPopupMenuC(WorkPlace->mWorkArea,"Popup");
```

```
      new XolPushItemWC(Popup,"Line"  ,'L');
      new XolPushItemWC(Popup,"Square",'S');
      new XolPushItemWC(Popup,"Circle",'C');
      new XolSepItemWC (Popup);
      new XolPushItemWC(Popup,"Quit",'Q',"Ctrl<Key>C","Ctrl-C"
                                   ,Quit);

    Canvas->mDrawArea->AddCallback(XmNinputCallback, (XolCallbackProcT)Input
                                                   , Popup);
    sGlobApp->Run();
}

void Quit(Widget, XolWidgetC* Widget, XolPointerT)
{
  printf("Exit was invoked by Object %s\n",Widget->GetFullName());
  exit(0);
}

void Open(Widget, XolWidgetC*, XolPointerT)
{
    XolMainWinC    *Main;
    XolWmShellC    *TopLevel;
    XolScaleBoxC   *Scale;
    MyFileSelectC  *FileBox;

    FileBox = new MyFileSelectC(sGlobApp);
    FileBox->Create();

    TopLevel = new XolWmShellC(sGlobApp,"XolTest");
    TopLevel->CreateRel(FALSE,30,30);
    TopLevel->SetTitle("Test Window");

    Main = new XolMainWinC(TopLevel->mClientArea,"Main");
    Main->Create(FALSE,TRUE,FALSE,FALSE,FALSE,XmNO_ORIENTATION);
    Scale = new XolScaleBoxC(Main->mPaned,"ScaleBox");
    Scale->Create(TRUE, TRUE, TRUE, XOL_MOD_PREC, 7, XmHORIZONTAL, -10, 10, 1);
    Scale->SetParam(-9000,9000,2);
    Scale->SetValue((float)-10.1);
    Main->mWorkArea = Scale;
    Main->Manage();

    TopLevel->Manage();
    return;
}

void Style(Widget, XolMenuItemC *Item, XolPointerT)
{
  XolMenuItemC *TmpItem1, *TmpItem2;
  bool Switch;

  TmpItem1 = (XolMenuItemC*)
           (sGlobApp->GetChild("Main/MenuBar/Lines/Line Style/Single",'/'));
  TmpItem2 = (XolMenuItemC*)
           (sGlobApp->GetChild("Main/MenuBar/Lines/Line Style/Double",'/'));
  Switch = (Item->mItemData == (XtPointer)1);
  TmpItem1->SetSense(!Switch);
  TmpItem2->SetSense(Switch);
```

```
  return;
}

void Input(Widget, XolMenuC *Popup
                        , XmDrawingAreaCallbackStruct *cbs)
{
    if (cbs->event->xany.type != ButtonPress ||
        cbs->event->xbutton.button != 3)
        return;

    Popup->Exposure((XButtonPressedEvent *)(cbs->event));
}

void MenuBarInit(XolMenuC *Menu1)
{
  XolPullDownMenuC *Menu2, *Menu3;

    Menu2 = new XolPullDownMenuC(Menu1, "File", 'F');


      new XolPushItemWC(Menu2,"Open",'O',"Ctrl<Key>O","Ctrl-O"
                                          ,Open);
      new XolPushItemWC(Menu2,"Save",'S');
      new XolSepItemWC (Menu2);
      new XolPushItemWC(Menu2,"Quit",'Q',"Ctrl<Key>C","Ctrl-C"
                                          ,Quit);

    Menu2 = new XolPullDownMenuC(Menu1, "Lines", 'L');

      Menu3 = new XolPullDownMenuC(Menu2, "Line Weight", 'W');

        new XolPushItemWC(Menu3,"1",'1');
        new XolPushItemWC(Menu3,"2",'2');
        new XolPushItemWC(Menu3,"3",'3');
        new XolPushItemWC(Menu3,"4",'4');

      Menu3 = new XolPullDownMenuC(Menu2, "Line Color", 'C');

        new XolPushItemWC(Menu3,"Cyan"   ,'C',"Alt<Key>C","Alt-C");
        new XolPushItemWC(Menu3,"Yellow" ,'Y',"Alt<Key>Y","Alt-Y");
        new XolPushItemWC(Menu3,"Magenta",'M',"Alt<Key>M","Alt-M");

      Menu3 = new XolPullDownMenuC(Menu2, "Line Style", 'S');

        new XolPushItemWC(Menu3,"Single",'S',"Alt<Key>S","Alt-S"
                                              ,(XolCallbackProcT)Style
                                              ,(XolPointerT)1);
        new XolPushItemWC(Menu3,"Double",'D',"Alt<Key>D","Alt-D"
                                              ,(XolCallbackProcT)Style
                                              ,(XolPointerT)2);

    Menu2 = new XolPullDownMenuC(Menu1, "Help", 'H');

  Menu1->Manage();

  return;
```

```
}

void Browse(Widget, XolWidgetC*, XolPointerT)
{
  sGlobApp->Browse("", XOL_BROWSE_ADDR | XOL_BROWSE_POS | XOL_BROWSE_SIZE);
  return;
}

void Info(Widget, XolWidgetC *Parent, XolPointerT)
{
  XolMessageBoxC *InfoMsg = new XolMessageBoxC(Parent,"InfoMsgBox");
  InfoMsg->Create(FALSE,TRUE);
  InfoMsg->New(InfoText1,"XOL_INFO_BIG_ITALIC");
  InfoMsg->Append(InfoText2,"XOL_INFO_NORMAL");
  InfoMsg->Append(InfoText3,"XOL_INFO_NORMAL");
  InfoMsg->Append(InfoText4,"XOL_INFO_NORMAL");
  InfoMsg->Append(InfoText5,"XOL_INFO_NORMAL_ITALIC");
  InfoMsg->SetOne(XmNtitle,(XtArgVal)"Info");
  InfoMsg->Manage();
  return;
}

void SorryMsgBox(Widget, XolWidgetC *Parent, XolPointerT)
{
  XolMessageBoxC *SorryMsg = new XolMessageBoxC(Parent,"SorryMsgBox");
  SorryMsg->Create(FALSE,TRUE,NULL,"XOL_HorstNo.xbm");
  SorryMsg->New(SorryText,"XOL_INFO_BIG_ITALIC");
  SorryMsg->SetOkString("Ok? no chance");
  SorryMsg->SetHelpString("Help? No Help");
  SorryMsg->SetCancelString("Cancel? Use it");
  SorryMsg->Manage();
  return;
}
```

References

[BGS92] Andreas Bäcker, Andreas Genau, and Markus Sohlenkamp. *GINA++ - The Generic Interaction Application for C++ and OSF/Motif.* German Natioinal Research Institut for Computer Science, St. Augustin, 1992.

[Fer93] Paula M. Ferguson. *Motif Reference Manual for OSF/Motif Release 1.2*, volume 6B of *The Definition Guides to the X Windows System.* O'Reilly & Associates Inc., Sebastopol, CA, 1993.

[Fet94] R. Fetik. *Motif 1.2 Style Guide Companion.* Prentice-Hall Inc., Englewood Cliffs, NJ, 1994.

[HF94] Dan Heller and Paula M. Ferguson. *Motif Programming Manual for OSF/Motif Release 1.2*, volume 6A of *The Definition Guides to the X Windows System.* O'Reilly & Associates Inc., Sebastopol, CA, 1994.

[LH94] A. Le Hors. *XPM Manual: The X PixMap Format, Version 3.4c.* BULL Research, Valbonne Cedex, March 1994.

[Nye92a] Adrian Nye. *X Toolkit Intrinsics Programming Manual*, volume 4 of *The Definition Guides to the X Windows System, OSF/Motif 1.2 edition.* O'Reilly & Associates Inc., Sebastopol, CA, 1992.

[Nye92b] Adrian Nye. *Xlib Programming Manual for Version 11*, volume 1 of *The Definition Guides to the X Windows System.* O'Reilly & Associates Inc., Sebastopol, CA, 1992.

[Nye92c] Adrian Nye. *Xlib Reference Manual for Version 11*, volume 2 of *The Definition Guides to the X Windows System.* O'Reilly & Associates Inc., Sebastopol, CA, 1992.

[Ope93] Open Software Foundation. *OSF/Motif Programmers Guide, Revision 1.2.* Prentice-Hall Inc., Englewood Cliffs, NJ, 1993.

[Sch96] Gunar Schorcht. *The SST common Object Library.* Technical Report SST-01/96, Technical University of Ilmenau, Dept. of Computer Science and Automation, March 1996.

[Unk] Unknown. *The Motif++ class library.* Unknown.

[You90] Douglas A. Young. *The X Window System: programming and applications with Xt, OSF/Motif edition.* Prentice-Hall Inc., Englewood Cliffs, NJ, 1990.

Index