Chart Object

ChartObject Programming Guide

ICS

Integrated Computer
Solutions, Incorporated

Integrated Computer Solutions, Inc.

54 B Middlesex Turnpike, Bedford, MA 01730

Tel: 617.621.0060

Fax: 617.621.9555

E-mail: info@ics.com

Third printing

January 2009

Contents

How to Use This Manual

Overview xi
Introduction xii
Document Road Map xiii
Notation Conventions xiii

Chapter 1—ChartObject Components

Overview 1
Introduction to ChartObject 2
Object Architecture 3
  Graphic Object Library Components 4
  Coordinate System 4
  Class Hierarchy 5
  Object Interface 5
  Pointer Resources 6
  Hello World Example 7
DataObject 9
  DataObject Components 9
  Examples 10
  Understanding Groups 12
  Linked Views 13
  Drag and Drop 13
  Missing Values 14
  Creating a View 14
  Data Editing 14
  Memory Allocation 16
  Navigating Inside a DataGroup 16
Components of ChartObject 17
  Creating a 2D Bar Chart Example 18
  Chart Components for 2D Plots 20
  Chart Components for 3D Plots 22
  Class Hierarchy 23
  Customizing Chart Components 24
  Transposition 25
  Combination of Plots 26
  Combination Plot Example 27
  Auto-scaling 31
  Creating And Using Templates 31

Object Editing 32
Object Selection 32
Moving and Resizing Objects 34
Verify Callback 34
Built-in Resource Editor 35
Customization of a Built-In Resource Editor 36
Creating Your Own Resource Editor 38
Setting Resources 40
Using Hardcoded Resources 41
Resource File 41
Restricted Resources 42
Resource File Example 43
Graphic Attributes 45
ObjectEditor Example 46
Editing Functions 49
Drag and Drop 50
Hardcopy 52
PostScript Output 52
CGM Output 54
Real-time Applications 54
Zoom 57
Data Viewport 58
Chart Zoom Example 59
Customizing or Creating New Chart Types 63
Inserting Application’s Defined Objects 63
Customizing an Existing Chart 64
Creating a New Chart Type 66

Chapter 2—Widget Reference

Overview 69
CompBase Widget Metaclass 70
Inherited Behavior and Resources 70
CompBase Resources 70
CompBase Functions 71
EditObject Widget Class 81
Coordinate System 81
Object Selection 81
Object Editing 81
Object Display 82
Input/Output 82
Clipboard 82

Locator 82
EditObject Widget Appearance 83
Inherited Behavior and Resources 83
EditObject Resources 84
EditObject Actions 89
EditObject Translations 92
EditObject Callbacks 93
EditObject Functions 98
Macros 108
ObjectEditor Widget Class 108
Object Editor Layout 108
ObjectEditor Widget Appearance 109
Inherited Behavior and Resources 109
ObjectEditor Resources 110
ObjectEditor Functions 116

Chapter 3—Graphic Object Reference

Overview 121
GraphicObject Library 122
Summary of Components 122
Graphic Object Metaclass 123
Interactive Editing 123
Visual Attributes 123
Coordinate System 123
Inherited Behavior and Resources 124
Defined Callbacks 130
Graphic Functions 131
Macros 131
Group Object Class 132
Interactive Grouping 132
Example 132
Group Resources 133
Group Functions 134
Macros 134
Image Object Class 135
Image Resources 135
Defined Functions 137
Macros 137
Line Object Class 138
Arrow Shape 138
Line Resources 138

- Line Callbacks 141
- Line Functions 141
- Macros 141

MultiPoint Object Metaclass 142
- MultiPoint Object Interactive Creation 142
- MultiPoint Object Editing 142
- Inherited Behavior and Resources 142

Oval Object Class 145
- Inherited Behavior and Resources 145
- Oval Callbacks 145
- Oval Functions 145
- Macros 145

Polyline Object Class 146
- Creation 146
- Editing 146
- Optimization 146
- Inherited Behavior and Resources 146
- Polyline Callbacks 148
- Polyline Functions 149
- Macros 149

Rectangle Object Class 149
- Inherited Behavior and Resources 149
- Rectangle Callbacks 151
- Rectangle Functions 151
- Macros 151

Symbol Object Class 152
- Symbol Editor 153
- Symbol Resources 153
- Symbol Keywords 156
- Symbol Description Syntax 157
- Symbol Callbacks 157
- Symbol Functions 158
- Macros 158

Text Object Class 159
- Text Resources 160
- Text Callbacks 163
- Defined Functions 163
- Macros 163

Chapter 4—DataObject Reference

Overview 165
DataObject 166
  Summary of Components 166
DataGroup Object Class 167
  Inherited Behavior and Resources 167
  Callback Structure 169
  Functions 171
  Macros 172
DataGrid Object 173
  Example 173
  Inherited Behavior and Resources 174
  Callback 177
  Functions 179
  Macros 180
DataLabel Object 181
  Example 181
  Inherited Behavior and Resources 182
  Callback 185
  Functions 186
  Macros 187
DataSampled Object Class 188
  Example 188
  Inherited Behavior and Resources 189
  Callback for Data Updates 191
  Functions 193
  Macros 195
DataSequentialSeries Object Class 196
DataSeries Object Class 196
  Example 196
  Inherited Behavior and Resources 197
  Callbacks 199
  Defined Functions 200
  Macros 202
DataTimeLabel Object Class 203
  Example 203
  Inherited Behavior and Resources 203

Chapter 5—Chart Object Reference

Overview 205
ChartObject Library 206

Chart Object Class 206
- Inherited Behavior and Resources 207
- Constraint Resources 212
- Chart Callbacks 213
- Functions 215
- Macros 222

ChartWidget Widget Class 223
- Inherited Behavior and Resources 223
- Functions 223

AxisObject Object Class 224
- Axis Limits and Increments 224
- Resources 224
- Inherited Resources 229
- AxisObject Callbacks 229
- Functions 230
- Macros 231

Legend Object Class 231
- Inherited Behavior and Resources 232
- Macros 234

Plot2D Object Metaclass 235
- Inherited Behavior and Resources 235
- Functions 240

Plot3D Object Metaclass 241
- Rotation, Translation and Scaling 241
- Inherited Behavior and Resources 242

CellArray Object Class 247
- Data 248
- Inherited Behavior and Resources 248
- Inherited Resources 249
- Macros 249

ComboPlot Object Class 250
- Resources 250
- Inherited Resources 250
- Functions 251
- Macros 252

BarLine Object Class 253
- Data 253
- Inherited Behavior and Resources 254
- Inherited Resources 255
- Macros 256
- BarLine Series 256

Bar3D Object 258
- Data 258
- Inherited Behavior and Resources 259
- Inherited Resources 260
- Macros 260

HighLow Object Class 261
- Data 261
- Inherited Behavior and Resources 262
- Inherited Resources 263
- Macros 263
- HighLow Series 264

Histogram Object 265
- Data 265
- Inherited Behavior and Resources 266
- Inherited Resources 266
- Macros 266
- Histogram Series 267

Pie Object Class 269
- Data 269
- Inherited Behavior and Resources 270
- Inherited Resources 271
- Functions 271
- Macros 271
- Pie Series 272

Surface3D Object 273
- Data 274
- Inherited Behavior and Resources 274
- Inherited Resources 277
- Macros 277

XYPlot Object Class 278
- Data 278
- Inherited Behavior and Resources 279
- Inherited Resources 280
- Macros 280
- PolySeries 281

How to Use This Manual

Overview

This chapter includes the following sections:

- **Introduction** on page xii
- **Document Road Map** on page xiii
- **Notation Conventions** on page xiii

Introduction

Charts have become a part of everyday life in today's multimedia society. You can't attend a meeting, pick up a newspaper, or watch a television news program, for more than a few minutes, without running across a chart of some sort. Charts are especially useful in software applications because they make complex data clearer, more interesting, and easier to understand.

The charting needs of application developers using the X-Window/Motif user interface has spawned a market for software tools that provide a wide range of charting capabilities to the programmer.

The currently available charting tools address the display of charts in a window, but have left the difficult part of interacting with the chart on the shoulders of the application programmer.

ChartObject is designed from the ground up as an interactive charting system based on the Model View Controller (MVC) architecture first introduced in Smalltalk. It brings graphical interactivity to a level never before achieved in a Motif charting package.

Interactive features like chart component editing, built-in attribute dialogs, drag and drop, table integration, cut and paste, multiple linked views, to name a few are all available in ChartObject.

Because ChartObject is a "true" object system, it provides for the quick creation of highly interactive charts, but does not restrict those programmers who require a high level of detailed tailoring in their charts. Each component of a chart is an object that has full graphical editing semantics.

An application programmer familiar with the Motif user interface can master ChartObject in an afternoon. The application interface to ChartObject is just the Xt Intrinsic interface already used by thousands of X/Motif programmers.

Using ChartObject in your application development will allow you to quickly ship highly interactive, visually pleasing applications to your customers.

Before you get started, read the following sections to acquaint yourself with the organization and typographical conventions of this manual.

Document Road Map

To get the most from this manual, we suggest you take the following steps:

- Use *Chapter 1, Introduction* to get acquainted with most major features of the INT ChartObject library and related widgets.
- Use *Chapter 2, Widget Reference* to read about the EditObject class, which is the container widget class for all graphic objects, and the ObjectEditor widget which provides a panel for interactively editing graphic objects.
- Use *Chapter 3, Graphic Object Reference* to understand the Graphic class, which provides the underlying architecture for Chart objects and to learn about all the specific graphic objects that can be used to provide annotation to chart plots.
- Use *Chapter 4, Data Object Reference* to understand how to build data objects and group them to describe your data.
- Use *Chapter 5, Chart Object Reference* to get acquainted with the Chart object and all the sub-objects created by Chart.

These five chapters can be used in any order. For first time users, we suggest that you start with Chapter 1 to get a quick overview of the product. You can then skip most of Chapters 2 and 3, and go directly to Chapters 4 and 5.

Notation Conventions

This guide uses the following text styles and symbols:

- **Boldface** indicates that the name in boldface is a reserved word such as the name of a resource, action or function associated with a widget.
- *Italics* indicates that this is the name of a widget class, the name of a push-button, the name of an argument in an argument list, or the name of a member in a C structure.
- `Monospaced` indicates that this is the text of a C program.

ChartObject Components

1

Overview

This chapter includes the following sections:

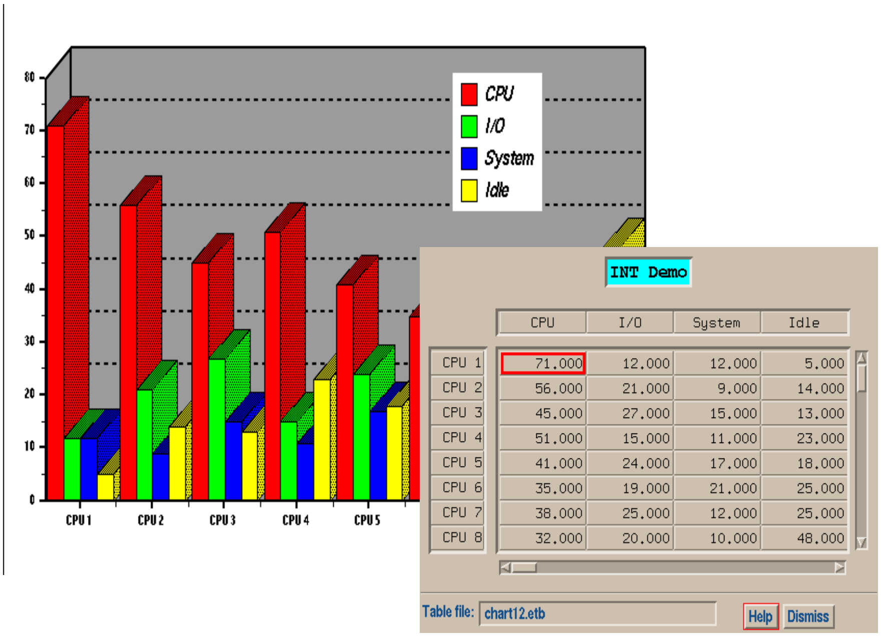
- **Introduction to ChartObject** on page 2
- **Object Architecture** on page 3
- **DataObject** on page 9
- **Components of ChartObject** on page 17
- **Object Editing** on page 32
- **Built-in Resource Editor** on page 34
- **Setting Resources** on page 40
- **Graphic Attributes** on page 45
- **Drag and Drop** on page 50
- **Hardcopy** on page 51
- **Real-time Applications** on page 54
- **Zoom** on page 57
- **Customizing or Creating New Chart Types** on page 63

Introduction to ChartObject

ChartObject is a powerful object-oriented library of 2D and 3D interactive graphic tools that enables X Windows/Motif developers and end users alike to create a wide range of charts. ChartObject goes beyond widget technology to give users a level of interactive real-time performance, flexibility and editing functionality unavailable in any other charting package.

MVC architecture

ChartObject is based on the *Model-View-Controller* architecture (MVC) first used in Smalltalk. This design allows both application developers and users to construct multiple views of the same data easily. For example, a set of data can be viewed simultaneously as tabular rows and columns, a bar chart or as a 3D surface chart, as shown in Figure 1:



	CPU	I/O	System	Idle
CPU 1	71.000	12.000	12.000	5.000
CPU 2	56.000	21.000	9.000	14.000
CPU 3	45.000	27.000	15.000	13.000
CPU 4	51.000	15.000	11.000	23.000
CPU 5	41.000	24.000	17.000	18.000
CPU 6	35.000	19.000	21.000	25.000
CPU 7	38.000	25.000	12.000	25.000
CPU 8	32.000	20.000	10.000	48.000

Figure 1. Multiple Views of a Data Object

ChartObject library

The ChartObject library includes:

- EditObject

  A widget class that is the container widget handling the display and editing of graphic objects. It implements the *controller* component of the MVC architecture. Refer to *Chapter 2—Widget Reference* for a complete description of the resources, callbacks and public functions defined by the EditObject class.

- DataObject

  A library of data objects that contain a representation of the application's data. It implements the *model* component of the MVC architecture. Refer to *Chapter 4—DataObject Reference* for a complete description of the objects contained in DataObject.

- GraphicObject

  A library of low level graphic objects that serves as the foundation of ChartObject. Refer to *Chapter 3—Graphic Object Reference* for a complete description of the objects defined in GraphicObject.

- ChartObject

  A library of high-level graphic objects, designed for charting, that implements the *view* component of the MVC architecture. Refer to *Chapter 5—Chart Object Reference* for a complete description of the objects defined in ChartObject.

Object Architecture

Xt Object Class

The INT Graphic objects are based on the Xt Object class. The Object class is the root of the Xt widget class hierarchy, as well as the root of the INT Graphic class hierarchy. It supports creation, deletion and resource handling. However, it does not support geometry, windows or event handling.

Parent widget

INT Graphic objects must only be created with a parent that is an EditObject widget or a widget whose class is derived from the EditObject class (such as an EditTable widget, for example). The EditObject class defines the actions and callbacks that allow the end-users to interact with the objects, including actions to select, move, reshape, cut and paste, insert objects interactively and add or delete points. The EditObject class also defines a number of public functions to manipulate objects including hardcopy output to PostScript and optionally CGM, plus ASCII input and output.

Graphic Object Library Components

The Graphic object library defines the following object classes:

Object Class	Description
Graphic	The base class for all graphic objects. It defines the basic resources and methods used by all other INT Graphic objects.
Group	To group primitive objects or other groups into a single group object
Image	To display an image.
Line	To draw a line between two points, with or without arrows at the end.
MultiPoint	Base class for all objects with multiple points such as a polyline.
Oval	To draw oval shaped objects.
Polyline	To draw polyline or polygon objects.
Rectangle	To draw rectangular objects.
Symbol	To draw a user specified symbol.
Text	To draw text.

Coordinate System

By default, all graphic objects use the coordinate system provided by their parent widget. The EditObject class coordinate system ranges between 0.0 and 100.0 in both directions, with the origin at the upper left corner. Some subclasses of EditObject, such as EditTable, redefine the default coordinate system. EditTable, for example, defines a coordinate system that ranges between 0 and *ncolumns* in the horizontal direction and between 0 and *nrows* in the vertical direction.

Note: The AxisObject is used by the chart class to define a new coordinate system that matches the user coordinate system inside the plot area.

Class Hierarchy

All graphic objects are based on a class hierarchy, where the subclasses inherit the resources and behavior from their superclass. Figure 2 illustrates the class hierarchy for the basic graphic objects defined in the GraphicObject library. Refer to *Chapter 3—Graphic Object Reference* for a complete description of resources and public functions defined by these object classes.

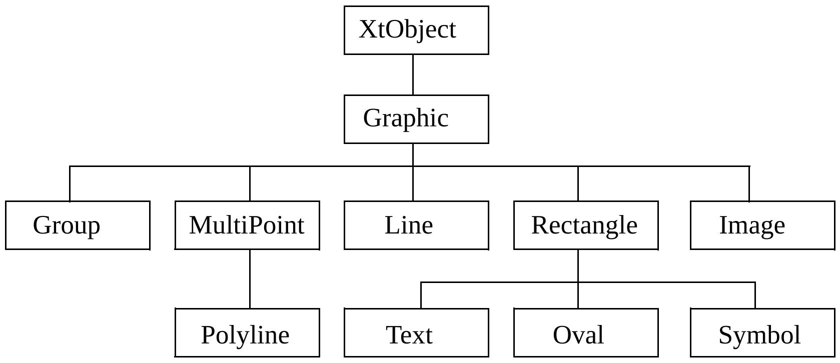


Figure 2. GraphicObject Library Class Hierarchy

The Chart object and all the specialized sub-objects used to build a Chart are also based on a similar hierarchy that is described later in this Chapter.

Object Interface

Objects in the GraphicObject library are declared as data type Object (defined in the Xt Intrinsic.h header file) and can be created in exactly the same manner as Motif widgets.

Using Xt creation functions

You can use the Xt creation functions:

```
Widget XtCreateWidget (String, WidgetClass, ArgList, Cardinal);
Widget XtVaCreateWidget (String, WidgetClass,...);
```

Additionally, each object class provides a creation convenience function is provided by each object class.

Code

For example, you can create a line object using the Xt creation function as follows:

```
Object line_object;
XintLine line;
line_object = (Object) XtVaCreateWidget (“line”,
                       (WidgetClass)xintLineObjectClass,
parent,
                        XmNline, &line, NULL);
```

or, using the creation function provided by the Line class:

```
n = 0;
XtSetArg (arg[n], XmNline, &line); n++;
line_object = XintCreateLine (parent, “line”, arg, n);
```

If you use the Xt creation function you must cast the return type to Object and the object class name to WidgetClass for ANSI style compilers.

Note: DO NOT manage objects. Managing objects causes corruption of the object record.

Changing object attributes

Changing object attributes can be accomplished with the Xt functions:

```
void XtSetValues (Widget, ArgList, Cardinal);
void XtVaSetValues (Widget,...);
```

For example, the following code sample changes the line style of a line object:

```
XtVaSetValues ((Widget) line_object,
               XmNlineStyle, XintLINE_ON_OFF_DASH, NULL);
```

Destroying an object

Finally, to destroy an object you can use functions:

```
void XtDestroyWidget(Widget);
void XintEditObjectDestroyObject(Object);
```

Both functions are equivalent, but the second one is faster.

Pointer Resources

Many resources in the INT library are specified as pointers to data structures, floating point values, or data arrays. Unless specified otherwise, an internal copy of the data is made by the widget or object, so you don’t need to keep the memory allocated. For example, the list of points to a polyline object is specified as an array of points using resource **XmNpointArray**. You can free this array after you have used it to create or modify a polyline object.

Functions XtSetValues and XtVaSetValues, when applied to a pointer resource, return the pointer to the internal widget or object data. You should not modify this data directly. This is a common error that often occurs when trying to modify a resource.

Example The following example, which applies a horizontal translation to a rectangle object, illustrates the problem.

```
Object rect_object;
XintRectangle *rectangle, new_rectangle;
...
XtVaGetValues((Widget) rect_object, XmNrectangle, &rectangle, NULL);

/* DON'T modify structure rectangle directly; copy data first...*/
new_rectangle = *rectangle;

/* apply translation */
new_rectangle.x1 += tx;
new_rectangle.x2 += tx;
XtVaSetValues((Widget) rect_object,
               XmNrectangle, &new_rectangle, NULL);
...
```

Hello World Example

The following example (see file `TextObject.c` in directory `examples/Chart`) illustrates how to build a Hello World application using the Text object:

Example

```
include <Xint/EditObject.h>
include <Xint/Text.h>

main (argc, argv)
int     argc;
char    *argv[];
{
  XtAppContext  app_context;
  Widget        top_level;
  Widget        edit;
  Object        text;
  XintTextLocation text_location;

  top_level  = XtAppInitialize(&app_context, "hello_world",
                    (XrmOptionDescList)NULL, 0, &argc, argv,
                     NULL, NULL, 0);
```

```
    /* Create an EditObject widget */
    edit = XtVaCreateManagedWidget("edit_object",
                    xintEditObjectWidgetClass, top_level,
                    XmNwidth, 400,
                    XmNheight, 400,
                    NULL);

     /* Create the Text object */
     text_location.x = 50;
     text_location.y = 50;
     text = (Object) XtVaCreateWidget("Hello World",
                    (WidgetClass)xintTextObjectClass, edit,
                    XmNtextLocation, &text_location,
                    XmNtextAnchor, XintCENTER,
                    XmNtextString, "Hello World",
                    XmNfontSize, 18,
                    XmNroundEdge, True,
                    XmNfillStyle, XintFILL_SOLID,
                    XmNlineStyle, XintLINE_SOLID,
                    NULL);

     /* Loop */
     XtRealizeWidget(top_level);
     XtAppMainLoop(app_context);
}
```

Figure 3 shows the output from this example:

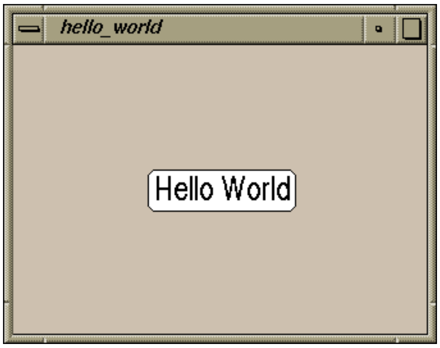


Figure 3. Hello World Example

DataObject

DataObject was created specifically for use with the INT ChartObject, which allows developers to create a wide variety of graphical data displays in the Motif environment easily. DataObject simply provides the mechanism to “package” and edit data, while the ChartObject provides the graphical viewing tools.

DataObject is a system of Xt intrinsic tools that provides for the retrieval, storage and manipulation of sets or groups of data. DataObject serves as the *model* component of the MVC architecture described earlier. As explained in the following pages, treating sets of data as individual objects not only makes the MVC architecture possible, but also provides the mechanism for a wide range of advanced data handling and graphical display features.

DataObject Components

DataObject provides the following object classes that can be used to classify and manipulate sets of data:

Object Class	**Description**
DataGroup	Allows the definition and manipulation of multiple data objects as a single group. For example, a group may contain sampled objects, series objects, grids, or even other groups.
DataGrid	Allows the definition and manipulation of a two-dimensional array of data values as a single object.
DataLabel	Allows the definition and manipulation of a set of labels that are used to identify visually the data in a chart or tabular view.
DataSampled	Allows the definition and manipulation of a one-dimensional array of data as a single object.
DataSeries	Allows the definition and manipulation of a series of (x,y) pairs as a single object.

Similar to the GraphicObject classes, the data classes are all derived from the Xt Object class. These classes can be manipulated using resources, functions, and callbacks, just like any standard Motif/Xt object or widget. The data objects must not be managed. Modification of the data can be accomplished using either standard Xt resource setting mechanisms or the convenience functions provided by INT.

Examples

Figure 4 shows a bar chart view containing annual sales data for four metropolitan areas:

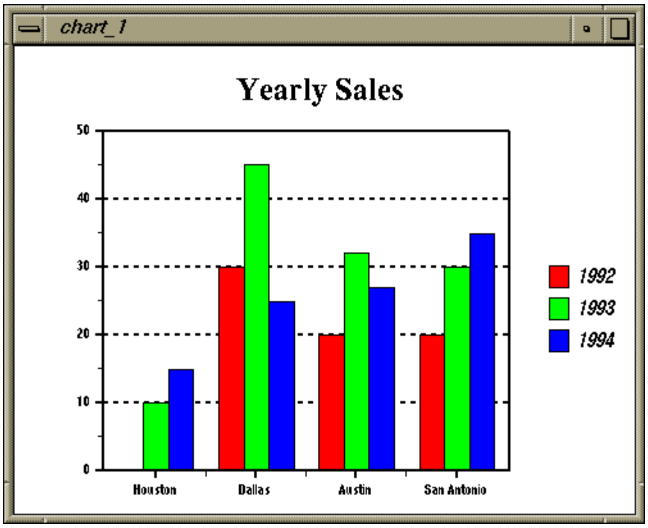


Figure 4. Bar Chart View of Grouped DataSampled Objects

Example

The following example (see file `Barchart.c` in directory `examples/Chart`) shows the complete programming code required to produce the bar chart display shown Figure 4:

```
include <Xint/EditObject.h>
include <Xint/Chart.h>

static String   x_labels[] = {“Houston”, “Dallas”, “Austin”,
                    “San Antonio”};
static float    d1992[]  = { 00.0, 30.0, 20.0, 20.0};
static float    d1993[]  = { 10.0, 45.0, 32.0, 30.0};
static float    d1994[]  = { 15.0, 25.0, 27.0, 35.0};

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

```
{
  XtAppContext  app_context;
  Widget        top_level;
  Widget        edit;
  Object        data_group;
  Object        chart;
  XintGeometry  chart_geometry;

  top_level  = XtAppInitialize(&app_context, “test”,
                 (XrmOptionDescList)NULL, 0,
                 &argc, argv, NULL, NULL, 0);

   /* Create an EditObject widget*/

   edit = XtVaCreateManagedWidget(“edit_object”,
                 xintEditObjectWidgetClass,top_level,
                 XmNwidth, 600, XmNheight, 600,
                 XmNobjectEditMode, XintEDIT_ADJUST,
                 NULL);

   /* Create Chart object */

    chart_geometry.x1 = 0;
    chart_geometry.y1 = 0;
    chart_geometry.x2 = 100;
    chart_geometry.y2 = 100;
    chart = (Object) XtVaCreateWidget(“BarPlot”,
                 (WidgetClass)xintChartObjectClass, edit,
                 XmNgeometry, &chart_geometry,
                 XmNchartType, XintCHART_TYPE_BAR,
                 XmNchartTitle, “Yearly Sales”,
                 XmNshowLegend, True,
                 NULL);

    /* Create a data group */

data_group = XintCreateDataGroup(edit, “Yearly Sales”, NULL, 0);

XtVaCreateWidget(“Cities”, (WidgetClass)xintDataLabelObjectClass,
                 edit,
                 XmNlabelStrings, x_labels,
                 XmNlabelCount, sizeof(x_labels)/sizeof(String),
                 XmNlabelOrientation, XintLABEL_X,
                 XmNdataGroup, data_group, NULL);

XtVaCreateWidget(“1992”, (WidgetClass)xintDataSampledObjectClass,
                 edit,
                 XmNdataArray, d1992,
                 XmNcount, sizeof(d1992)/sizeof(float),
                 XmNdataType, XintDATA_TYPE_FLOAT,
                 XmNdataGroup, data_group, NULL);
XtVaCreateWidget("1993", (WidgetClass)xintDataSampledObjectClass,
                       edit,
                       XmNdataArray, d1993,
                       XmNcount, sizeof(d1993)/sizeof(float),
                       XmNdataType, XintDATA_TYPE_FLOAT,
                       XmNdataGroup, data_group, NULL);

XtVaCreateWidget("1994", (WidgetClass)xintDataSampledObjectClass,
                       edit,
                       XmNdataArray, d1994,
                       XmNcount, sizeof(d1994)/sizeof(float),
                       XmNdataType, XintDATA_TYPE_FLOAT,
                       XmNdataGroup, data_group, NULL);

    /* Associate the data group with the chart object */

    XintChartAssociateData(chart, data_group);

    /* Loop forever */

    XtRealizeWidget(top_level);
    XtAppMainLoop(app_context);
}
```

Understanding Groups

In the previous example, that we created a group that contains multiple data objects. The DataGroup conceptually serves as a "container" for all other data object classes. The following diagram shows an example of parent-child relationships that could occur between various objects in a group.

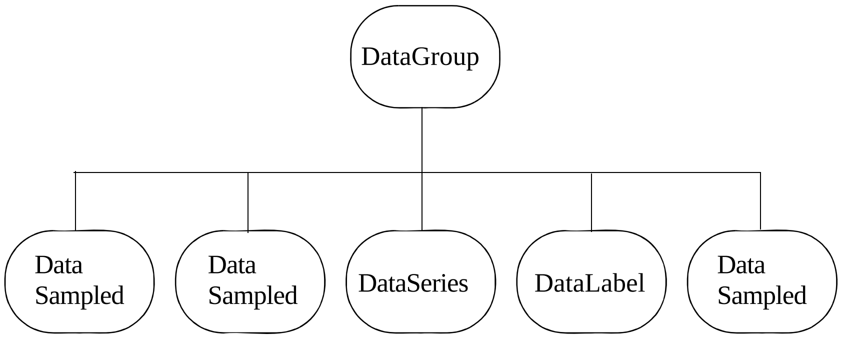


Figure 5. Example of a DataObject Group

Notice that you can have multiple instances of any data object type within the same group. The DataLabel object provides annotation for either the entire group or for individual objects. It accompanies the group to any destination view, and is inserted into the view in the appropriate context (for example, as column/row annotation in a table or as an axis label in a chart).

Linked Views

The object classes discussed above allow interlinking between multiple views of the same data, so that any changes to one view are automatically applied to all other connected views. For example, if the user changes data in a table view (such as INT's EditTable), this automatically updates the model and the model updates all other views, such as bar charts, pie charts, and all the other INT chart types.

Drag and Drop

DataObject provides an advanced, built-in "drag and drop" feature that allows users to select any group of data from one view, then drag the data and drop it into another view, effectively creating a linked view as described above. For instance, the user might highlight a range of cells in a table then drag and drop the highlighted selection into a chart. When this happens, the data points that were dragged to the chart are automatically inserted correctly in the appropriate view context. If the user drags data into a bar chart, it appears as a series of bars; in a pie chart it appears as a series of wedges, and so forth. The drag and drop function can optionally link the views so that updates in one are automatically reflected in the other.

In addition, the data *labelling* follows the data from one view to the next. In a table, the labels might appear as horizontal and vertical titles. In a chart they appear as axis labels. When a user performs a drag-and-drop, the built-in widget mechanism automatically creates the appropriate groups and data objects to accomplish the task. The widget automatically maintains correct hierarchical relationships between data elements, and attaches the appropriate label to the new view.

Missing Values

All data objects that handle numerical values support the concept of missing or null values. To specify that a data value is missing just insert one of the following constants into the data array based on the data format.

Constant	Description
XintUNDEFINED_DOUBLE	Missing value for double data format.
XintUNDEFINED_FLOAT	Missing value for float data format.
XintUNDEFINED_INTEGER	Missing value for integer data format.
XintUNDEFINED_LONG	Missing value for long data format.
XintUNDEFINED_SHORT	Missing value for short data format.

Creating a View

Data objects or groups are associated with chart objects through the following function:

```
void XintChartAssociateData (Object chart, Object data)
```

This function creates a view of the data object that is displayed by the chart object. See Chart Object Reference section in Chapter 5 for more information on this function.

A similar function is available to associate a data object with an EditTable widget:

```
Boolean XintEditTableAssociateData(Widget edit_table, Object data,
                         int col_start, int row_start, Boolean link)
```

See the EditTable manual for a complete description of this function.

Data Editing

DataObject provides editing functions that are available to both the application programmer and the end-user. An application programmer may modify the contents of a data object through resources (XtSetValues or XtVaSetValues) or convenience functions. Whenever possible, it is better to update the data using the convenience functions provided by the various data objects classes for optimization reasons.

Replacing a sample in a type float object

The following code segment shows how to replace the first sample in a DataSampled object of type float:

```
float new_value;
...
XintDataSampledDataReplace (data_sampled, &new_value, 0, 1);
```

The end-user of an application may optionally modify data indirectly by interactively editing a tabular or graphical view of the data. Real-time editing features for replacing, extending or shifting the contents of a data object are also provided. Application programmers can be notified of editing operations using the callback XmNupdateCallback, which can be registered on each data object or at the data group level and may override any requested editing operation. Also, resource **XmNeditable** can be used to prevent a specific data object or data group from being modified through the editing of a graphical view.

Data groups can also be modified at any time by adding or destroying data objects. Again, if the data group is connected to a chart, the chart will automatically update itself to take into account the changes in the data.

Inserting new DataSampled objects

If you need to update several data objects inside a data group that is connected to a chart, use function XintDataBatchUpdate to freeze the propagation of updates to the views and minimize flashing.

The following code sample illustrates how to insert two new DataSampled objects to a data group *data_group* that is already connected to a chart.

```
/* Freeze propagation of updates for this data group */
XintDataBatchUpdate(data_group, True);

/* add the two new data_sampled objects */
XtVaCreateWidget (“new1”, (WidgetClass)
xintDataSampledObjectClass,
                  edit,
                  XmNdataArray, data1,
                  XmNcount,     count1,
                  XmNdataGroup, data_group, NULL);
XtVaCreateWidget (“new2”, (WidgetClass)
xintDataSampledObjectClass,
                  edit,
                  XmNdataArray, data2,
                  XmNcount,     count2,
                  XmNdataGroup, data_group, NULL);

/* allow updates */
XintDataBatchUpdate(data_group, False);
```

Memory Allocation

By default, data objects make an internal copy of the data. Set resource **XmNcopyData** to False, when creating the data object, if you don't want the data object to copy your data. In this case, the data passed to the data object should not be deallocated so long as the data object is not destroyed.

Data resource **XmNlastViewDestroy** can be used to cause a data object be destroyed automatically after it is no longer associated with a view (chart or table). If you are using a data group, you only need to set this resource on the data group; you don't need to set it for the data objects belonging to the data group.

Navigating Inside a DataGroup

If you are using a DataGroup object to store your data, you don't need to keep your own list of the data objects belonging to the data group. Function XintDataGroupIterate is designed to retrieve data objects from a data group.

Printing name of all data objects

For example, the following code sample shows how to print the name of all the data objects of type DataSeries in DataGroup data_group.

```
Object data_group, data_series;
...
index = 0;
while ((data_series = XintDataGroupIterate (data_group,
             xintDataSeriesObjectClass, index++)) != NULL) {
  printf("series name = %s number = %d\n",
         XtName((Widget) data_series), index);
}
```

If you specify NULL for the name of the object class, function XintDataGroupIterate iterates through all the data objects, whatever the type.

Components of ChartObject

ChartObject provides the *view* component of the MVC architecture described earlier. To create a chart, the application or the end-user needs to create a Chart object. This object automatically creates a number of sub-objects to compose the plot. For example, the chart may create Axis objects, Text objects for annotation, a Legend object, a Plot object, Series objects, etc. The number and type of objects created are dependent upon the chart type as described in more detail below.

Object classes

The ChartObject library defines the following object classes:

Object Class	Description
Chart	Main object that creates and manages all the objects necessary for building a chart.
AxisObject	Provides axis annotation and coordinate transformation for the 2D plot components.
Legend	To draw a legend.
Plot2D	Base class for all 2D plot classes
Plot3D	Base class for all 3D plot classes.
ComboPlot	Special class used as a container for multiple plots.
BarLine	Plot class that displays its data as bars, stacked bars or lines.
CellArray	Plot class that displays its data as colored rectangular cells.
Bar3D	Plot class that displays its data as 3D bars.
HighLow	Plot class that displays high-low-open-close graphs.
Histogram	Plot class that displays the distribution of a set of data.
Pie	Plot class that displays its data as 2D pies.
Surface3D	Plot class that displays its data as a 3D surface.
XYPlot	Plot class that displays its data as a 2D area, line or scatter plot.

Creating a 2D Bar Chart Example

This example (`HorizontalStackedBarChart.c` in `examples/Chart`) shows how to create a 2D bar chart with horizontal, stacked, contiguous bars:

Code

```
include <Xint/EditObject.h>
include <Xint/Chart.h>
include <Xint/DataGroup.h>
include <Xint/DataSampled.h>
include <Xint/BarLine.h>

static String   x_labels[] = {“Houston”, “Dallas”, “Austin”,
                              “San Antonio”};
static float    d1992[] = { 10.0, 20.0, 20.0, 8};
static float    d1993[] = { 20.0, 35.0, 32.0, 17};
static float    d1994[] = { 39.0, 41.0, 37.0, 21};

main(argc, argv)
int     argc;
char    *argv[];
{ XtAppContext  app_context;
  Widget        top_level;
  Widget        edit;
  Object        data_group;
  Object        chart, plot;
  XintGeometry  chart_geometry;

  top_level  = XtAppInitialize(&app_context, “Chart Example”,
                              (XrmOptionDescList)NULL, 0,
                              &argc, argv, NULL, NULL, 0);

   /* Create an EditObject widget*/
   edit = XtVaCreateManagedWidget(“edit_object”,
                              xintEditObjectWidgetClass,
                              top_level,
                              XmNwidth, 400,
                              XmNheight, 250,
   /* make the chart editable */
                              XmNobjectEditMode, XintEDIT_ADJUST,
                              NULL);
   /* Create Chart object */
    chart_geometry.x1 = 0;
    chart_geometry.y1 = 0;
    chart_geometry.x2 = 100;
    chart_geometry.y2 = 100;
    chart = (Object) XtVaCreateWidget(“BarPlot”,
                  (WidgetClass)xintChartObjectClass, edit,
                  XmNgeometry, &chart_geometry,
                  XmNchartType, XintCHART_TYPE_BAR,
                  XmNchartTitle, “Yearly Sales”,
                  XmNshowLegend, True,
                  NULL);
   /* Make horizontal,stacked bars with no space between bars */
   plot = XintChartGetComponent(chart, XintCHART_COMPONENT_PLOT);

   XtVaSetValues((Widget) plot,
                  XmNbarOrientation, XintHORIZONTAL,
                  XmNbarStyle, XintSTACKED,
                  XmNclusterWidth, 100,
                  NULL);
/* Create a data group */
     data_group = XintCreateDataGroup(edit, "Yearly Sales", NULL, 0);

     XtVaCreateWidget("Cities", (WidgetClass)xintDataLabelObjectClass,
                        edit,
                        XmNlabelStrings, x_labels,
                        XmNlabelCount, sizeof(x_labels)/sizeof(String),
                        XmNlabelOrientation, XintLABEL_X,
                        XmNdataGroup, data_group, NULL);

     XtVaCreateWidget("1992",
                      (WidgetClass)xintDataSampledObjectClass, edit,
                        XmNdataArray, d1992,
                        XmNcount, sizeof(d1992)/sizeof(float),
                        XmNdataType, XintDATA_TYPE_FLOAT,
                        XmNdataGroup, data_group, NULL);

     XtVaCreateWidget("1993",
                      (WidgetClass)xintDataSampledObjectClass, edit,
                        XmNdataArray, d1993,
                        XmNcount, sizeof(d1993)/sizeof(float),
                        XmNdataType, XintDATA_TYPE_FLOAT,
                        XmNdataGroup, data_group, NULL);
XtVaCreateWidget("1994",
                      (WidgetClass)xintDataSampledObjectClass, edit,
                        XmNdataArray, d1994,
                        XmNcount, sizeof(d1994)/sizeof(float),
                        XmNdataType, XintDATA_TYPE_FLOAT,
                        XmNdataGroup, data_group, NULL);
/* Associate the data group with the chart object */
     XintChartAssociateData(chart, data_group);

     /* Loop forever */
     XtRealizeWidget(top_level);
     XtAppMainLoop(app_context);
}
```

Figure 6 illustrates the output from this example:

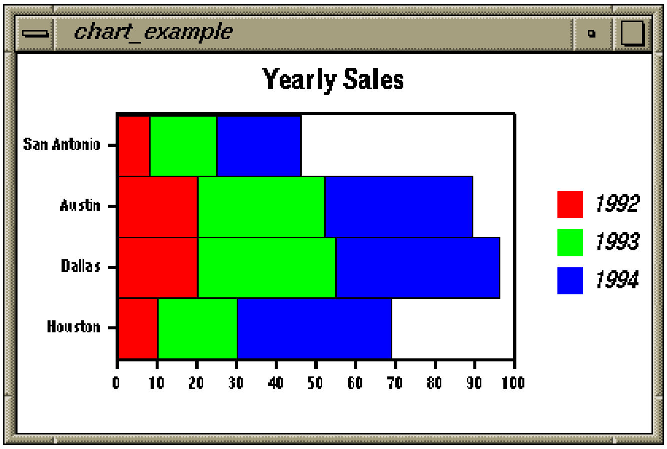


Figure 6. Horizontal Stacked Bars

Chart Components for 2D Plots

A Chart object consists of a grouping of objects created automatically by Chart. Figure 7 illustrates the object set created by Chart for a 2D plot:

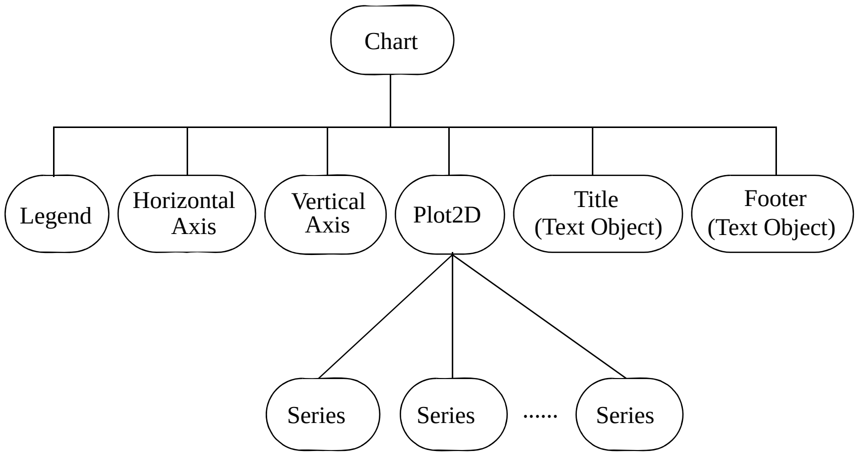


Figure 7. Components Created by Chart for a 2D Plot

In Figure 8, Plot2D is the base class for all 2D plots. Depending on the chart type selected, it could be a BarLine object, a Pie object, a XYPlot object, and so forth. Refer the appropriate plot section (such as BarLine or Pie) in *Chapter 5—Chart Object Reference* for more detailed information about the type of series created.

Note: Series are generic objects associated with data objects.

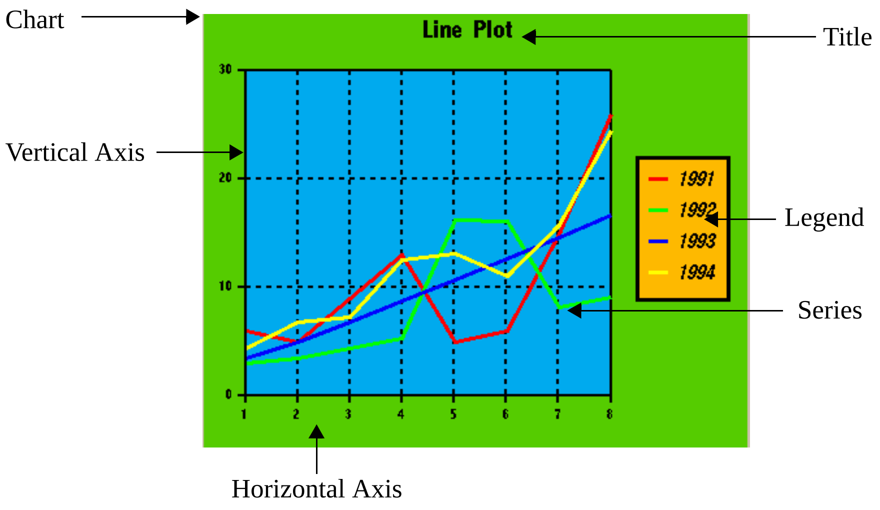


Figure 8. Components of a 2D Chart

Components created when displaying a 2D plot

The following table summarizes all the components that may be created inside a Chart object when a 2D plot is displayed.

Component	Object Class	Description
Vertical Axis	AxisObject	Vertical axis associated with the plot area.
Horizontal Axis	AxisObject	Horizontal axis associated with the plot area.
Plot2D	BarLine CelllArray HighLow Histogram Pie XYPlot	Bar chart. Grid of colored cells. High-Low-Open-Close chart. Histogram chart. Pie chart. Line, scattered or area charts.
Title	Text	Title for the chart.
Footer	Text	Footer for the chart.
Legend	Legend	Legend for the chart.
Series	Graphic	Object representing a data series. Type can vary (polyline, bars, wedge, etc.) based on the Plot2D class. Unless specified otherwise, use only resources defined in the Graphic class to customize attributes.

Chart Components for 3D Plots

Figure 9 illustrates the set of objects created by a Chart object for a 3D plot:

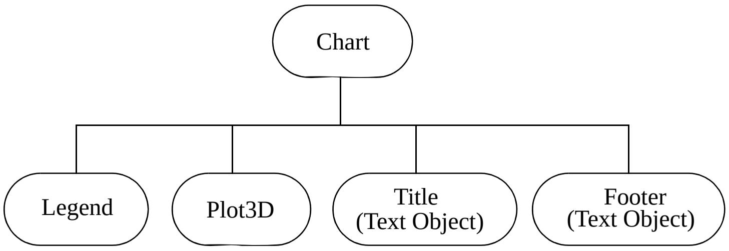


Figure 9. Components Created by Chart for a 3D Plot

In Figure 9, Plot3D is the base class for all 3D plots. Depending on the type of chart selected it could be a Bar3D or a Surface object. See `chart_demo.c` in the `demos` directory, `ChartDemo1` subdirectory. Figure 10 illustrates Selection 8:

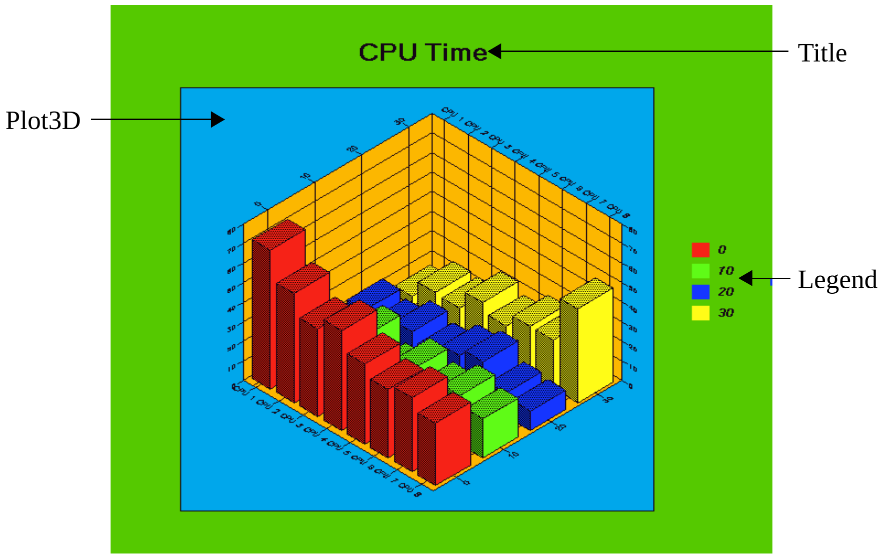


Figure 10. Components of a 3D Chart

Components created when displaying a 3D plot

The following table summarizes all the components that can be created inside a Chart object when a 3D plot is displayed.

Component	Object Class	Description
Plot3D	Bar3D Surface3D	Bar3D chart. Surface chart.
Title	Text	Title for the chart.
Footer	Text	Footer for the chart.
Legend	Legend	Legend for the chart.

Class Hierarchy

All objects comprising a Chart are based on a class hierarchy, where subclasses inherit the resources and behavior from their superclasses. Figure 11 shows the class hierarchy for the objects described in this section:

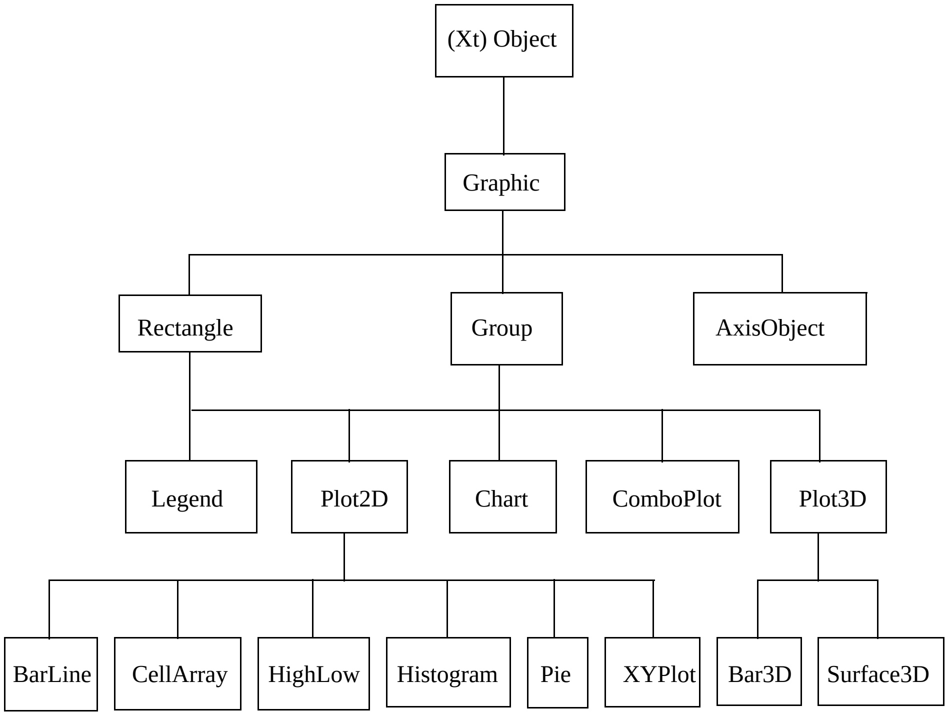


Figure 11. Class Inheritance Diagram

Customizing Chart Components

All components created by a chart object can be accessed and modified to fit the application's requirements. Function XintChartGetComponent allows the application to retrieve a component from a chart object.

Code

The following code sample illustrates how to use this function to retrieve the ID of the legend object of a chart object and to modify it.

```
Object chart, legend;
...
legend = XintChartGetComponent (chart, XintCHART_COMPONENT_LEGEND);
/* we set the legend border thickness, the # of columns and */
/* remove the fill.                                          */
XtVaSetValue((Widget) legend, XmNlineThickness, 3,
                              XmNfillStyle, XintFILL_NONE,
                              XmNcolumns, 2, NULL);
```

Series objects, whose number and type depends on the data associated with the chart and the chart type, cannot be retrieved with function XintChartGetComponent. Instead, you should use function XintChartGetSeriesOfData, which requires you to specify a data component. This function returns a list, which contains in most cases one series. It can also contain more than one series, for example, if transposition is set.

Code

The following code sample illustrates how to change the line style of the series used to display DataSampled object *data_sampled*.

```
Object chart, data_sampled;
Object *series_list;
int count;
...
series_list = XintChartGetSeriesOfData(chart, data_sampled, &count);
if (series_list != (Object *) NULL) {

    XtVaSetValue((Widget) series_list[0],
                 XmNlineStyle, XintLINE_ON_OFF_DASH, NULL);

   /* don't forget to free the list */
   XtFree ((char *) series_list);
}
```

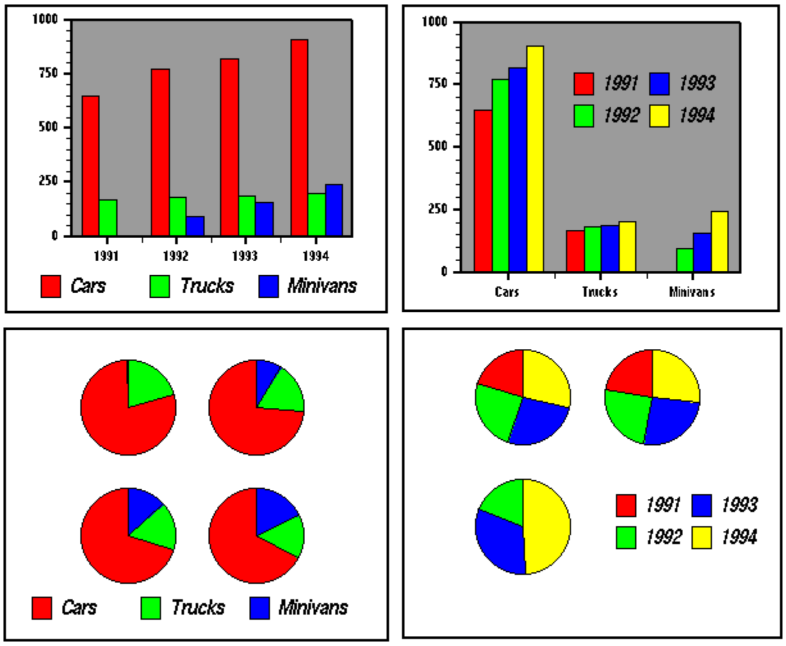
Transposition

Data objects of type DataSampled are displayed using a data series. For example, a DataSampled object in a BarLine chart is represented as a set of bars, each bar belonging to a different group. The number of bars in a group is equal to the number of DataSampled objects connected to the chart. For a Pie chart, each sample of the DataSampled object is displayed as a wedge, each wedge being located in a different pie.

Chart resource **XmNtranspose** can be used to transpose the data contained in DataSampled objects. The transpose option is honored by BarLine, Pie and Bar3D plot classes. For example, when transpose is set for a Pie, each sample of a DataSampled object is represented as a wedge inside the same pie. In this case there would be one pie for each DataSampled object connected to the chart.

Figure 12 illustrates transposition for a dataset containing three DataSampled objects as described in the following table (see `chart_demo.c` in the demos directory, `ChartDemo1` subdirectory, selection 17):

DataSampled Name	**Range (start, inc.)**	**Data Values (in thousands)**
Cars	1991, 1	650, 776, 821, 910
Trucks	1991, 1	170, 184, 191, 203
Minivans	1992, 1	95, 159, 245 (production started in 1992!)



XmNtranspose False XmNtranspose True

Figure 12. Transposition Example

Combination of Plots

Chart library allows you to combine primitive plot types to build a composite plot. For example, Figure 13 shows a composite plot made from a HighLow and a BarLine plot:

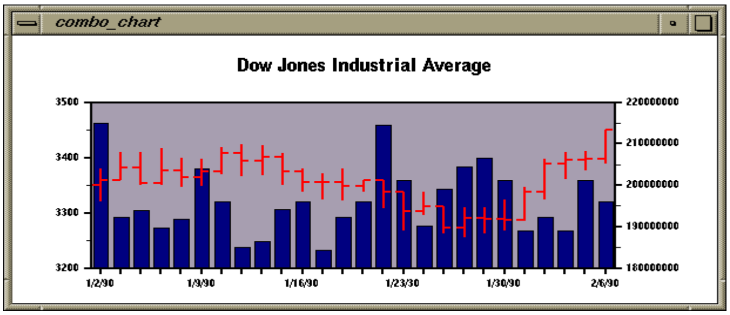


Figure 13. Combination Plot Example

Combining plots The mechanism to combine plots is very flexible. You first create a chart with resource **XmNchartType** set to XintCHART_TYPE_COMBINATION. You then retrieve the ID of the combo plot created and then create as many new plot types as you want inside the combo plot object using function XintComboPlotCreateNewPlot. If several plots have axes displayed on the same side, the axes will automatically be stacked next to each other.

You can combine as many primitive plot types as you want. However, since 3D plot types cannot be made transparent, you should not have more than one 3D plot in a combination plot.

Combination Plot Example

The following code example (see file `ComboPlot.c` in directory `examples/Chart`) illustrates how to create a combination plot composed of a Bar plot and a HighLow plot. Figure 13 shows the output produced by the example.

include <Xint/EditObject.h>

```
include <Xint/Chart.h>
include <Xint/ComboPlot.h>
include <Xint/DataGroup.h>
include <Xint/DataSampled.h>
include <Xint/DataLabel.h>
include <Xint/AxisObject.h>

static int high[] =
   { 3380, 3410, 3411, 3418, 3400, 3399, 3420, 3425, 3423,
     3409, 3381, 3372, 3380, 3360, 3359, 3338, 3338, 3312,
     3310, 3311, 3324, 3348, 3399, 3410, 3412, 3450};
static int low[] =
    { 3321, 3360, 3350, 3350, 3348, 3349, 3370, 3366, 3369,
      3350, 3339, 3324, 3322, 3339, 3309, 3268, 3297, 3263,
      3256, 3264, 3268, 3286, 3325, 3361, 3378, 3390};
static int open[] =
    { 3350, 3360, 3382, 3355, 3378, 3365, 3375, 3408, 3395,
      3401, 3375, 3356, 3357, 3349, 3359, 3338, 3303, 3312,
      3274, 3292, 3289, 3288, 3339, 3389, 3397, 3398};
static int close[] =
    { 3360, 3382, 3355, 3378, 3365, 3375, 3408, 3395, 3401,
      3375, 3356, 3357, 3349, 3359, 3338, 3303, 3312, 3274,
      3292, 3289, 3288, 3339, 3389, 3397, 3398, 3450};
static int volume[] =
    {215123000, 192432123, 194145400, 189900143, 191893456,
     204143657, 196143200, 185123456, 186458099, 194312765,
     196234234, 184456567, 192334345, 196234246, 214523546,
     201234566, 190233456, 199234567, 204455678, 206778456,
     201233434, 189234356, 192324566, 189234567, 201232356,
     196234565};

static float label_position[] = {0.0, 5.0, 10.0, 15., 20.0, 25};
static char *date_strings[] =
    {"1/2/90", "1/9/90", "1/16/90", "1/23/30","1/30/90", "2/6/90"};

main(argc, argv)
  int   argc;
  char  *argv[];
{
  XtAppContext  app_context;
  Widget        top_level, edit;
  Object        data_group, volume_data;
  Object        chart, plot, bar_plot, high_low_plot, axis;
  XintGeometry  chart_geometry;

  top_level  = XtAppInitialize(&app_context, "high_low_test",
                        (XrmOptionDescList)NULL, 0,
                        &argc, argv, NULL, NULL, 0);

    /* Create an edit object */
    edit = XtVaCreateManagedWidget("edit", xintEditObjectWidgetClass,
                        top_level, XmNwidth, 600, XmNheight, 300,
                        NULL);

    /* Create a data group */
    data_group = XintCreateDataGroup(edit, "Dow Jones", NULL, 0);

    XtVaCreateWidget("High", (WidgetClass)xintDataSampledObjectClass,
                    edit,
                    XmNdataArray, high,
                    XmNcount, sizeof(high)/sizeof(int),
                    XmNdataType, XintDATA_TYPE_INTEGER,
                    XmNdataGroup, data_group, NULL);

    XtVaCreateWidget("Low", (WidgetClass)xintDataSampledObjectClass,
                    edit,
                    XmNdataArray, low,
                    XmNcount, sizeof(low)/sizeof(int),
                    XmNdataType, XintDATA_TYPE_INTEGER,
                    XmNdataGroup, data_group, NULL);

    XtVaCreateWidget("Open", (WidgetClass)xintDataSampledObjectClass,
                    edit,
                    XmNdataArray, open,
                    XmNcount, sizeof(open)/sizeof(int),
                    XmNdataType, XintDATA_TYPE_INTEGER,
                    XmNdataGroup, data_group, NULL);

    XtVaCreateWidget("Close", (WidgetClass)xintDataSampledObjectClass,
                    edit,
                    XmNdataArray, close,
                    XmNcount, sizeof(close)/sizeof(int),
                    XmNdataType, XintDATA_TYPE_INTEGER,
                    XmNdataGroup, data_group, NULL);
    /* Create a Date label object */
    XtVaCreateWidget("Date", (WidgetClass)xintDataLabelObjectClass,
                     edit,
                     XmNlabelCount, 6,
                     XmNlabelOrientation, XintLABEL_X,
                     XmNlabelPositionArray, label_position,
                     XmNlabelStrings, date_strings,
                     XmNdataGroup, data_group, NULL);

    /* Create volume data */
    volume_data = (Object) XtVaCreateWidget("Volume",
                     (WidgetClass)xintDataSampledObjectClass, edit,
                     XmNdataArray, volume,
                     XmNcount, sizeof(volume)/sizeof(int),
                     XmNdataType, XintDATA_TYPE_INTEGER,
                     NULL);

    /* Create a chart inside the edit object */
    chart_geometry.x1 = chart_geometry.y1 = 0;
    chart_geometry.x2 = chart_geometry.y2 = 100;
    chart = (Object) XtVaCreateWidget("combination_chart",
                     (WidgetClass)xintChartObjectClass, edit,
                     XmNgeometry, &chart_geometry,
                     XmNchartType, XintCHART_TYPE_COMBINATION,
                     XmNchartTitle, "Dow Jones Industrial Average",
                     NULL);

    plot = XintChartGetComponent(chart,
XintCHART_COMPONENT_PLOT);

    bar_plot = XintComboPlotCreateNewPlot(plot, XintPLOT_TYPE_BAR);
    XintChartAssociateData(bar_plot, volume_data);

    high_low_plot = XintComboPlotCreateNewPlot(plot,
                     XintPLOT_TYPE_HIGH_LOW);
    XintChartAssociateData(high_low_plot, data_group);

    /* Position the axes */
    XtVaSetValues((Widget) bar_plot,
                     XmNxAxisPlacement, XintPLACEMENT_NONE,
                     XmNyAxisPlacement, XintPLACEMENT_RIGHT,
                     NULL);

    XtVaSetValues((Widget) high_low_plot,
                     XmNxAxisPlacement, XintPLACEMENT_BOTTOM,
                     XmNyAxisPlacement, XintPLACEMENT_LEFT,
                     NULL);

    /* Change the format of the vertical axis for the volume */
    axis = XintComboPlotGetComponent(plot,
                     XintCHART_COMPONENT_VERTICAL_AXIS, 0);
    XtVaSetValues((Widget) axis, XmNannotationFormat, "%.0f", NULL);

    XtRealizeWidget(top_level);
    XtAppMainLoop(app_context);
}
```

Plot composition Plot composition is handled by the ComboPlot class, which is a special class that manages multiple primitive plot types, such as BarLine, XYPlot, HighLow, etc. ComboPlot function XintComboPlotCreateNewPlot allows you to create a new plot of the type you want. Figure 14 illustrates the relationships between the various objects created in the Combination Plot example:

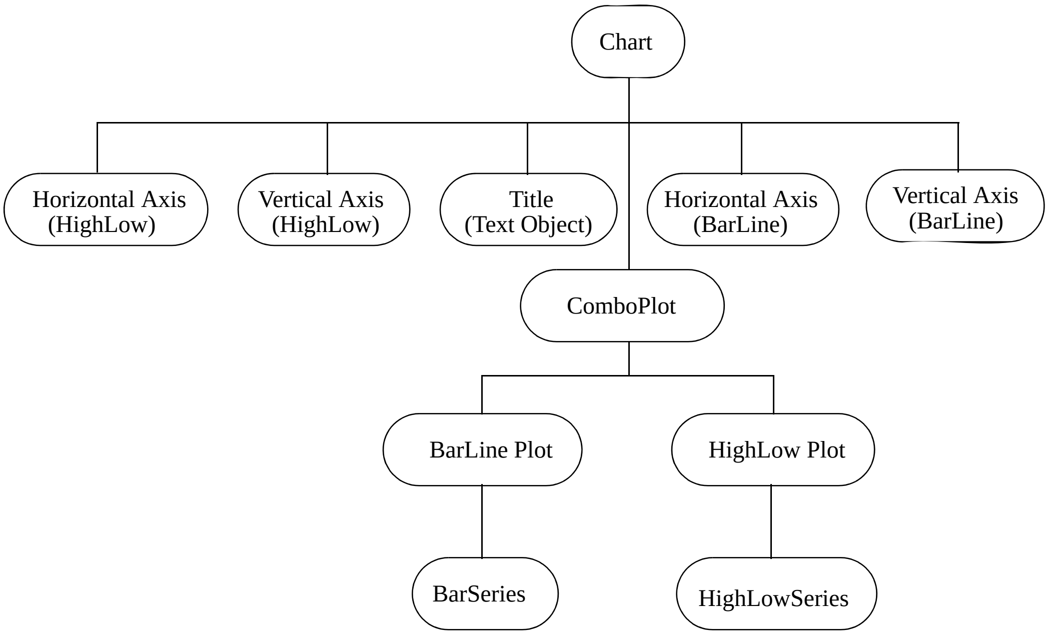


Figure 14. Relationship Between Objects Created in Combination Plot Example

Auto-scaling

When data is associated with a chart, the range of the axes and the annotation increments are calculated automatically based on the data. For example, if your data ranges from 3.8 to 93.7, the Y axis will be adjusted to range from 0 to 100, with an annotation increment set to 10. If you don't want auto-scaling, or if you want to see only a portion of the data (to zoom a plot for example), you can use Plot2D resources **XmN[xy]Limits**, or Plot3D resources **XmN[xyz]Limits** to control the range of the axes. Once you set a limit resource for a particular direction, auto-scaling will be turned off for that direction. The limit resources are specified as a pointer to a data structure of type XintLimits which takes the following form:

```
typedef struct {
     float minimum;
     float maximum;
} XintLimits;
```

If you want the auto-scaling to apply only to the minimum or the maximum, you can set one of the fields to constant XintUNDEFINED_FLOAT. For example, to set the minimum value to 0 and to let the maximum be calculated automatically you can specify the following:

```
Object plot;
XintLimits y_limits;
...
y_limits.minimum = 0;
y_limits.maximum = XintUNDEFINED_FLOAT;

XtVaSetValues ((Widget) Plot, XmNyLimits, &y_limits, NULL);
```

Creating And Using Templates

Two functions are available which greatly reduce the effort of the application designer in creating new charts. The first function, XintChartSaveTemplate, saves a single chart or a list of chart objects in a disk file. The chart objects are saved in template form, which means that only the visual attributes of the chart are saved, not the data.

The following code sample illustrates how simple it is to save a chart object in template form:

```
Object chart;
...

XintChartSaveTemplate(“template”, &chart, 1);
```

Because only the visual attributes are saved, charts are associated with new data when they are restored in another application. They may then be modified by convenience functions in that application. In fact, any processing that can be performed on chart objects can be performed on the restored templates. Function XintChartReadTemplate reads the chart objects stored in a template file created by XintChartSaveTemplate. Using a list of data groups, it associates a data group with each chart object from the file.

The following code shows how to restore a chart from disk.

```
Widget edit;
Object data_group, *list;
int object_count;
...

list = XintChartReadTemplate(edit, “template”, &data_group,
                                             1, &object_count);
```

Two other functions, XintEditObjectWriteFile and XintEditObjectReadFile, can be used to save and restore objects. These functions read and write the entire object description, including the associated data. They are described in the EditObject Widget Class section of this manual.

Object Editing

The ChartObject library offers a lot of flexibility when it comes to selecting, editing or modifying object resources interactively. The following sections cover some of the main issues regarding object editing.

Object Selection

Any graphic object displayed in an EditObject widget can be selected. Selected objects are highlighted using handle bars or by drawing the outline of the object using a specified color. Graphic resource **XmNhighlightMode** can be used to set how each object should be highlighted. By default, each object created is selectable. To prevent the end-user from selecting an object, set Graphic resource **XmNsensitive** to False when you create the object.

A Chart object does not normally propagate resource changes to its sub-objects. To make a Chart object and all of its components non selectable, set resource **XmNpropagate** to True each time you want the change to be applied to all the chart components, as shown in the following code sample:

```
XtVaSetValues ((Widget) chart, XmNsensitive, False,
                      XmNpropagate, True, NULL);
```

The EditObject actions that control object selection are ObjectSelect, InitAreaSelection, ExtendAreaSelection and EndAreaSelection. These actions are connected as follows in the default EditObject translation table:

Action	Default Selection Translation
Ctrl<Btn1Down>	InitAreaSelection(extend) ObjectSelect(single)
None<Btn1Down>	InitAreaSelection(single) ObjectSelect(single)
<PtrMoved	ObjectSelect()
<Btn1Up>	ObjectEditEnd()

This table shows that Button1 is used to select an object. If you press the Ctrl key while doing the selection, the object is added to the list of selected objects. Otherwise, the current selection list is erased before the object is selected. Also, if you drag Button1 while pressing it, all the objects contained in the rectangular area outlined while the cursor is moved will be selected when the button is released. See Advanced Topics section in this Chapter to see how to redefine the translation table.

EditObject callbacks XmNobjectSelectionCallback and XmNobjectDeselectionCallback can be used by the application to determine when an object has been selected or deselected. For objects in a group, the Group object ID is returned as the selected object. Since a Chart is a group of objects, the ChartObject ID will be returned when a chart sub-component is selected. Use function XintChartGetSelectedComponent to find out the type and object ID of the sub-component that was selected.

Example

The following code segment shows how to write a selection callback that prints the name of the selected object:

```
static void ObjectSelectionCallback (Widget, data, cb)
Widget widget;
XtPointer data;
XintEditObjectSelectionCallbackStruct *cb;
{
   Object selected_object;
   int code;

    if (XintIsChart(cb->object))
     selected_object = XintChartGetSelectedComponent(cb->object,
                                                     &code);
    else
       selected_object = cb->object;

    printf("Selected object name = %s\n",
           XtName((Widget) cb->object));
}
```

Moving and Resizing Objects

Object moving and resizing is controlled by EditObject actions ObjectEditStart, ObjectEdit and ObjectEditEnd. These actions are connected to Button1 in the default translation table for the EditObject class as shown in the table below:

Action	Default Editing Translation
None <Btn1Down>	ObjectEditStart()
<PtrMoved>	ObjectEdit()
<Btn1Up>	ObjectEditEnd()

Action ObjectEditStart supports an argument. For example, if you just want the end-user to move objects, you can specify ObjectEditStart(move) in the translation table. If you don't specify an argument to action ObjectEditStart, the behavior of this action is controlled by EditObject resource **XmNobjectEditMode**. The purpose of this resource is to enable you to modify the behavior of action ObjectEditStart without having to redefine a new translation table. By default, resource **XmNobjectEditMode** is set to XintEDIT_NONE which disables all editing. To enable full object editing, set this resource to constant XintEDIT_ADJUST. See editobject.c in the demos directory, EditObject subdirectory.

Object editing can also be controlled on an object by object basis using resources **XmNmove** and **XmNshape**. These resources are set to True by default. To disable editing for a particular object, set **XmNmove** and/or **XmNshape** to False.

Verify Callback

Callback XmNverifyCallback is a callback that can be registered on any graphic object. It is invoked whenever the object has been edited. For example, this callback will be invoked each time an object has been moved or resized. Most classes redefine the callback structure that is returned with this callback to provide specific information relevant to each object class.

For chart objects, in addition to being called for the reasons described above, this callback is also invoked when the chart type has been changed. The callback structure returns both the old and the new chart type.

Built-in Resource Editor

Some Graphic objects, have a built-in panel that allows the end-user to edit the object resources. The objects that have a built-in resource editor are: Text, Chart, AxisObject, BarLine, Bar3D, HighLow, Histogram, Pie, Surface3D, XYPlot, BarSeries, LineSeries, Symbol and Legend. Figure 15 shows the built-in Chart Editor:

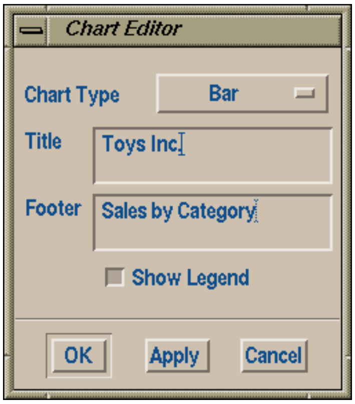


Figure 15. Built-in Chart Editor

The built-in resource editor can be invoked for a specific object using public function XintEditObjectManageResourceDialog. Action ResourceDialog (defined by the EditObject class) is also available for the end-user to interactively edit an object’s resources. This action is connected as follows in the default translation table:

Action	Default Dialog Translation
None <Btn1Down>	ResourceDialog()

Note: Action ResourceDialog actually checks for a double click, so it will only trigger on a double click of Button1.

You can disable the resource editor panel for a specific object by setting Graphic resource **XmNresourceDialog** to False when creating the object. You can disable all resource dialogs by registering callback XmNresourceDialogCallback on the EditObject widget and setting the doit flag member of the callback structure to False, as shown in the following code:

Code

```
static void ResourceDialogCallback(widget, data, cb)
Widget widget;
XtPointer data;
XintEditObjectResourceDialogCallbackStruct *cb;
{
        cb->doit = False
    }
```

If you need to unmanage the parent of a chart object (EditObject widget), you should first invoke Graphic function XintGraphicUnmanageDialog on the chart to unmanage all built-in dialogs that may be active. Failure to do so will prevent those dialog panels from ever reappearing the next time the chart's parent is managed (Motif bug).

Customization of a Built-In Resource Editor

Each built-in resource editor panel can be customized to some extent. For example, it is possible to remove an item in the panel, remove an item inside an option menu or rename a component in the resource editor panel. The application has also the option to provide its own custom editor panel using callback XmNresourceDialogCallback.

Note: all objects of the same class share the same built-in editor panel. So, any change to an object panel will automatically apply to all the other objects of the same class.

The widget(s) used to edit a particular resource are encapsulated inside a container widget (box). For example, an editor for a resource of type String is composed of a box containing a label and a Text widget. Function XintResEditGetBox retrieves the box widget ID for a particular resource.

```
Widget XintResEditGetBox(Object object, String resource_name,
                                  int num);
```

You can unmanage the box to prevent a particular resource from being displayed and edited. Argument *num* represents the number of components for the resource and it should be set to 1 in most cases. Some resources however, such as **XmNlimits** for the AxisObject, have several components (one to edit the minimum value and one to edit the maximum value). Setting argument *num* to 1will retrieve the box used to edit the minimum value. Setting argument *num* to 2 retrieves the box used to edit the maximum value.

Most resource editors use a label widget or gadget to display the text identifying which resource is being set (see strings “Chart Type”, “Title”, “Footer”, “Show Legend” in Figure 15). Function XintResEditGetLabel retrieves the label ID for a particular resource if there is one. If no label is defined, the function returns NULL. This label ID can be used to modify the text displayed in the editor.

```
Widget XintResEditGetLabel(Object object, String resource_name,
                                int num);
```

Some resources, such as **XmNchartType**, are edited using an option menu. Function XintResEditGetMenuButton can be used to retrieve the ID of a particular button in the option menu. Argument *button_name* is the name of the option as it is specified in the menu. Unmanage the button ID returned by function XintResEditGetMenuButton to remove an option from the menu.

```
Widget XintResEditGetMenuButton(Object object, String resource_name,
                                String button_name);
```

The following example illustrates how to use these functions. We modify the chart editor shown in Figure 15 as follows:

- Remove the chart footer component.
- Rename “Chart Type” to “Type”.
- Remove option “High Low” from the option menu used to select the chart type.

Code

```
Object chart;
    Widget button, label;
    ...
    /* Unmanage XmNchartFooter editor */
    box = XintResEditGetBox(chart, XmNchartFooter, 1);
    if (box != NULL) XtUnmanageChild(box);

/* Change name of label from “Chart Type” to “Type” */
    label = XintResEditGetLabel(chart, XmNchartType, 1);
    if (label) {
      XmString cstring;
      cstring = XmStringCreateSimple(“Type”);
      XtVaSetValues(label, XmNlabelString, cstring, NULL);
      XmStringFree(cstring);
    }

    /* Unmanage High Low option in chart type option menu */
    button = XintResEditGetMenuButton(chart, XmNchartType,“High
Low”);
    if (button) XtUnmanageChild(button);
```

Creating Your Own Resource Editor

If the editing options provided to customize a resource editor panel are not sufficient for your application, you may want to create your own custom panel using callback XmNresourceDialogCallback. This callback must be registered on the parent EditObject widget and it is called for all objects that need to be edited. It is important that you check for the object class or the object ID so that you display your custom panel only for the specified object class or object. For objects in a group, this callback always returns the object ID of the top group. So if the object returned is of class ChartObject, you should call function XintChartGetSelectedComponent to retrieve the actual component of the chart that needs to be edited. The example below illustrates how to redefine the resource editor panel for the Axis object class. Our custom panel only lets the end-user edit the axis label. The complete listing of this example can be found in file `UserCustomizedResourceEditor.c`, in directory `examples/Chart`.

Code

```
static void ResourceDialogCallback(widget, data, cb)
Widget widget;
XtPointer data;
XintEditObjectResourceDialogCallbackStruct *cb;
{
    static Widget my_axis_panel = NULL;
    static Widget axis_widgets[2];
    Object selected_object;
    int code;

    if (XintIsChart(cb->object))
       selected_object = XintChartGetSelectedComponent(
                                          cb->object, &code);
    else
       selected_object = cb->object;

    if (XintIsAxisObject(selected_object)) {

     /* Create my own panel if first time */
      if (!my_axis_panel)
         my_axis_panel = BuildAxisPanel(widget, axis_widgets);

     /* load axis panel to contain current axis state */
     LoadAxisPanel(axis_widgets, selected_object);

     /* Display my panel */
     XtManageChild(my_axis_panel);

     /* Turn off built-in panel */
     cb->doit = False;
   }
}
```

Function BuildAxisPanel builds the dialog panel and fills the array *axis_widgets* with the widget ID's it will need later. This function uses an INT convenience function, IntCreateDialogPanel, that creates a dialog widget with a set of buttons as specified.

Code

```
static Widget BuildAxisPanel(widget, axis_widget_list)
Widget widget;
Widget *axis_widget_list;
{
  Widget panel, vbox;

  panel = (Widget) IntCreateDialogPanel(XtParent(widget),
                         "My Axis Editor",
                         IntOK | IntAPPLY | IntCANCEL,
                         IntOK, EditAxisCallback,
                         (XtPointer)axis_widget_list,
                         XintVERTICAL, 5, 5, &vbox, NULL);

  /* Create text to edit axis label */
  axis_widget_list[0] = XtVaCreateManagedWidget("axis_label",
                         xmTextWidgetClass, vbox,
                         XmNcolumns, 20,
                         XmNeditMode, XmSINGLE_LINE_EDIT,
                         NULL);
  /* Save edit object ID. We will need it in EditAxisCallback */
  axis_widget_list[1] = widget;

  return panel;
}
```

Function LoadAxisPanel retrieves the resources from the specified objects and loads the values into the panel before we manage it.

```
static void LoadAxisPanel(axis_widget_list, object)
Widget *axis_widget_list;
Object object;
{
  char *label_string;

  XtVaGetValues((Widget) object, XmNlabel, &label_string, NULL);

  XmTextSetString(axis_widget_list[0], label_string);
}
```

Finally, here is the code for callback EditAxisCallback which is the callback invoked when the OK or APPLY button is selected from the custom menu.

Code

```
static void EditAxisCallback(widget, axis_widget_list, cb)
  Widget widget;
  Widget *axis_widget_list;
  XmAnyCallbackStruct   *cb;
{
  char *axis_label;
  Object *list;
  Object object;
  int i, count;
  int code;

  if (cb->reason == IntOK || IntAPPLY) {
    axis_label = XmTextGetString(axis_widget_list[0]);

   /*
    * get the list of selected object, and apply SetValues to the
    * Axis object(s)
    */
    list = XintEditObjectSelectList(axis_widget_list[1], &count);

    for (i=0; i<count; i++) {
      if (XintIsChart(list[i])) {
        object = XintChartGetSelectedComponent(list[i], &code);
        if (XintIsAxisObject(object))
          XtVaSetValues((Widget) object, XmNlabel, axis_label, NULL);
      }
    }

   /* Cleanup */
    if (list) XtFree((char *) list);
    if (axis_label) XtFree(axis_label);
  }
}
```

Setting Resources

In addition to the built-in resource editor, the resources for objects in the ChartObject library can be set by any of the methods that are available for setting resources in X. The two most common methods are hardcoded resources and resources that are defined in a resource file. One of the benefits in using a resource file is to customize the interface for different end-users.

Using Hardcoded Resources

ChartObject component resources are set in the source code in much the same way as Motif widget resources are specified. The principle difference is that the component object ID must be retrieved through the use of the function, XintChartGetComponent. This function returns the object ID, which is then used in XtSetValues or XtVaSetValues to define the required resource values.

The following code segments illustrate these steps:

- Define a ChartObject by using XtCreateWidget or XtVaCreateWidget. Any of the Chart object class resources can be defined in the function's argument list.

```
chart = (Object) XtVaCreateWidget("BarPlot",
                     (WidgetClass)xintChartObjectClass, edit,
                     XmNgeometry, &chart_geometry,
                     XmNchartType, XintCHART_TYPE_BAR,
                     XmNchartTitle, "Yearly Sales",
                     XmNshowLegend, True,
                     NULL);
```

- Get the desired component from the chart.

```
plot = XintChartGetComponent(chart, XintCHART_COMPONENT_PLOT);
```

- Define the necessary component object resources

```
XtVaSetValues((Widget) plot,
                     XmNbarOrientation, XintHORIZONTAL,
                     XmNbarStyle, XintSTACKED,
                     XmNclusterWidth, 100,
                     NULL);
```

Resource File

A resource file provides a simple, direct means for the application designer to give the end user control over those aspects of the look and behavior of the interface which can be allowed to change without interfering with the function of the application itself.

Chart object resources and chart component resources are set in the same manner as are X resources, using instance names and/or class names. Drop the *XmN* prefix from the resource names.

Setting a value to an INT constant

To set a value to an INT constant, drop the *Xint* prefix from the constant and convert the remaining term to lower case. For example:

`XintPLACEMENT_LEFT` becomes `placement_left`

Several of the hardcoded resources in the previous section could be defined by resource file entries, as shown in the following example.

```
*XintChart.chartType: chart_type_bar
*XintChart.chartTitle: Yearly Sales
```

Chart component resources may be identified by their class name in order to specify new values in the resource file. However, this makes changes in all occurrences of the class. The following table lists the object name and the class name for end-user specifiable ChartObject components.

Object Name	Class Name
chart_title	XintText
chart_footer	XintText
chart_legend	XintLegend
plot_haxis (created for 2D plots only)	XintAxisObject
plot_vaxis (created for 2D plots only)	XintAxisObject

Changing resources for all TextObjects

To change resources for all TextObjects use the class name:

```
*XintText.color: cyan
*XintText.fontFamily: helvetica
```

Changing only chart title resources

To change only the chart title's resources use the object name:

```
*chart_title.color: red
*chart_title.fontFamily: helvetica
```

Restricted Resources

Not all ChartObject component object resources can be changed by using a resource file. When chart creates these objects, some of their resources are set by the code. These can be changed by using XtSetValues, but not from a resource file. Any resources which are not set by the code are available to the end-user for customization of the interface.

The following table lists some of the resources set when chart creates its components (they cannot be changed by the end-user in a resource file):

Object Name	Resource Name
chart_title	XmNfontSize XmNfontWeight XmNtextString
chart_footer	XmNfontSize XmNfontWeight XmNtextString
plot_haxis	XmNtickPlacement
plot_vaxis	XmNtickPlacement

Chart resources **XmNchartTitle** and **XmNchartFooter** can be used to set title and footer text strings.

Resource File Example

The following example illustrates the use of a resource file to customize a chart. In Figure 16, a simple bar chart is displayed. In Figure 17, the same bar chart is displayed using a resource file.

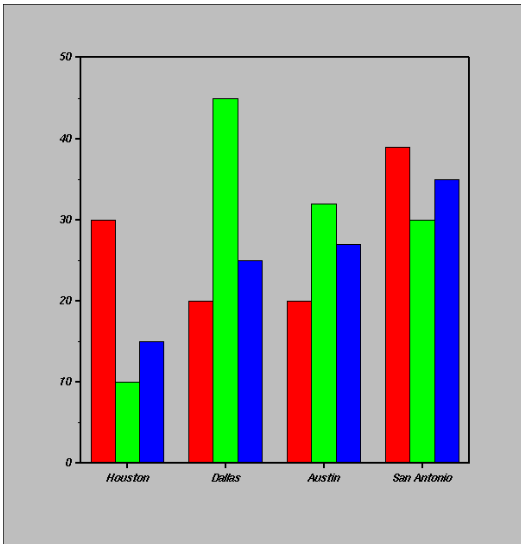


Figure 16. Simple Bar Chart Without a Resource File

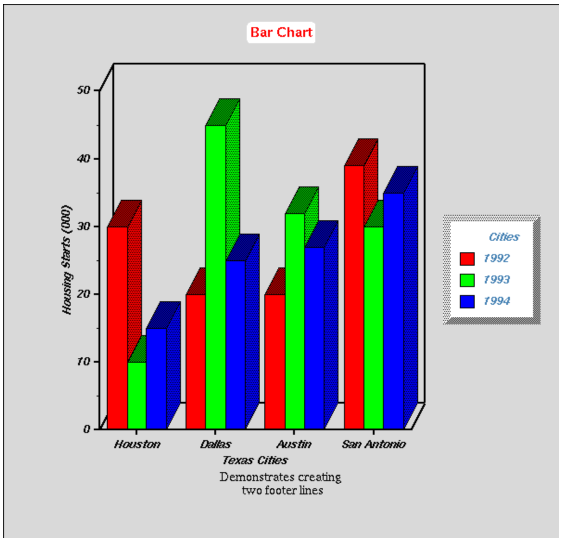


Figure 17. Bar Chart With a Resource File

Resource file

The following resource file contains all resource settings used to change the display from the one shown in Figure 16 to the one in Figure 17:

```
!     resource file example
!
!       Chart Class
!
*chartTitle: Bar Chart
*chartFooter: Demonstrates creating \n two footer lines
*fillColor: gray85
!
!       Text Class
!
*chart_title.color: red
*chart_title.fontFamily: helvetica
*chart_title.dashList: 3,,1
*chart_title.fillColor: white
*chart_title.fillStyle: fill_solid
*chart_title.roundEdge: True
*chart_footer.color: black
*chart_footer.fontFamily: times
```

```
!
!       Legend Class
!
*.showLegend: True
*chart_legend.columns: 1
*chart_legend.LegendTitle: Cities
*chart_legend.font: *Helvetica*-120-*
*chart_legend.highlightMode: highlight_none
*chart_legend.marginHeight: 10
*chart_legend.marginWidth: 10
*chart_legend.color: steel blue
*chart_legend.fillColor: white
*chart_legend.fillStyle: fill_solid
*chart_legend.lineStyle: shadow_out
*chart_legend.lineThickness: 8
!
!       AxisObject
!
*XintAxisObject.labelFont: *-Helvetica*-120-*
*plot_haxis.label: Texas Cities
*plot_haxis.color: black
*plot_vaxis.label: Housing Starts (000)
*plot_vaxis.color: black
!
!       BarLine Class
!
*XintBarLine.drawShadow: True
*XintBarLine.inclination: 20
*XintBarLine.rotation: 10
```

Graphic Attributes

ObjectEditor widget

The graphic class, which is the base class for all graphic objects, defines the resources that control the appearance of an object, including line color, fill color, bitmap pattern, line width, etc. There is no built-in editor for those resources. Instead, widget class ObjectEditor is available for building a menu that can be used to create and/or edit graphic objects interactively. The appearance of the menu created by widget ObjectEditor can be entirely customized to fit your application's requirements. See Chapter 2 for a complete description of the resources and public functions defined by class ObjectEditor.

ObjectEditor Example

The following example (see file `ObjectEditor.c` in directory `examples/Chart`) illustrates how to create an empty EditObject and a ObjectEditor panel that can be used to interactively create and edit objects in the EditObject window.

```
include <Xm/Form.h>
include <Xm/Frame.h>
include <Xint/EditObject.h>
include <Xint/ObjectEditor.h>
include <Xint/Text.h>
include <Xint/RoundedRect.h>
include <Xint/Rectangle.h>
include <Xint/Oval.h>
include <Xint/Line.h>
include <Xint/Polyline.h>
include <Xint/Polygon.h>
include <Xint/FreeHand.h>
include <Xint/Chart.h>

/* Define list of pixmaps used when creating the ObjectEditor widget */
static char *pixmap_names[] = {
    "50_percent_1", "50_percent_2",  "Vertical1", "Horizontal1",
    "Slant_Left1",  "Slant_Right1",  "Weave2",    "Weave4",
    "Diaper",       "Diamond2",      "Trellis1",  "Cross_Hatch",
   "Tread2",       "Herring_Bone2", "Zigzag2",   "Check2", NULL};

main(argc, argv)
int     argc;
char    *argv[];
{
   XtAppContext  app_context;
   Widget        top_level;
   Widget        form, frame, edit, object_editor;
   ObjectClass   object_class_list[12];
   Pixmap        pixmap_list[20];
   int           n;

   top_level  = XtAppInitialize(&app_context, "object_editor",
               (XrmOptionDescList)NULL, 0, &argc, argv, NULL, NULL, 0);
   XtVaSetValues(top_level, XmNallowShellResize, True, NULL);

   /* Create a Form container */
   form = XtVaCreateManagedWidget("form", xmFormWidgetClass,
                                  top_level, NULL);
   /* list the Object classes that we want to create interactively */
   n = 0;
   object_class_list[n] = xintRectangleObjectClass; n++;
   object_class_list[n] = xintRoundedRectObjectClass; n++;
   object_class_list[n] = xintOvalObjectClass; n++;
   object_class_list[n] = xintLineObjectClass; n++;
   object_class_list[n] = xintPolylineObjectClass; n++;
   object_class_list[n] = xintPolygonObjectClass; n++;
   object_class_list[n] = xintTextObjectClass; n++;
   object_class_list[n] = xintFreeHandObjectClass; n++;
   object_class_list[n] = xintChartObjectClass; n++;
   object_class_list[n] = NULL; n++;

   /* Build a list of bitmap patterns */
    n = 0;
    while (pixmap_names[n]) {
        pixmap_list[n] =
XintObjectEditorGetDefinedPixmap(form,
                          pixmap_names[n]);
        n++;
    }
    pixmap_list[n] = XmUNSPECIFIED_PIXMAP;

   /* Create an ObjectEditor widget */
   object_editor = XtVaCreateManagedWidget("object_editor",
                  xintObjectEditorWidgetClass, form,
                  XmNorientation,         XintVERTICAL,
                  XmNnumColumns,          1,
                  XmNobjectOrientation,   XmHORIZONTAL,
                  XmNobjectNumColumns,    3,
                  XmNpixmapNumColumns,    4,
                  XmNpixmapList,          pixmap_list,
                  XmNshowAttributeLabels, False,
                 XmNobjectClassList,     object_class_list,
                    /* Form constraint resources */
                  XmNleftAttachment,      XmATTACH_FORM,
                  XmNtopAttachment,       XmATTACH_FORM,
                  XmNbottomAttachment,    XmATTACH_FORM,
                  NULL);

   /* Create an Frame widget */
   frame = XtVaCreateManagedWidget("frame", xmFrameWidgetClass,
                  form,
                  XmNshadowType,      XmSHADOW_IN,
                    /* Form constraint resources */
                  XmNrightAttachment,  XmATTACH_FORM,
                  XmNtopAttachment,    XmATTACH_FORM,
                  XmNbottomAttachment, XmATTACH_FORM,
                  XmNleftAttachment,   XmATTACH_WIDGET,
                  XmNleftWidget,       object_editor, NULL);
    /* Create an EditObject widget */
    edit = XtVaCreateManagedWidget("edit_object",
                   xintEditObjectWidgetClass, frame,
                   XmNwidth,            600,
                   XmNheight,           600,
                   XmNobjectEditMode,   XintEDIT_ADJUST, NULL);

    /* Connect the object editor to the edit object widget */
    XintObjectEditorAddEditObjectToList(object_editor, edit);

    /* Loop */
    XtRealizeWidget(top_level);
    XtAppMainLoop(app_context);
}
```

Figure 18 displays the output from this example:

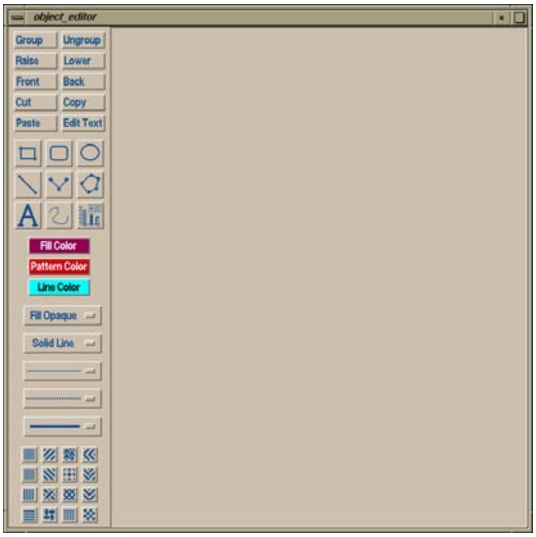


Figure 18. ObjectEditor Example Output

Editing Functions

Some applications may want to provide some of the functionality of the ObjectEditor widget by using their own interface. All of the controls available through the ObjectEditor widget class can be duplicated using public functions defined in the EditObject classes or by setting resources defined in the Graphic class.

Implementing functionality

The following table indicates how to implement the functionality provided by the ObjectEditor widget class.

Functionality	Function or Resource
Inserting an object interactively	XintEditObjectInsert.
Cut/Copy/paste	XintEditObjectCut, XintEditObjectCopy and XintEditObjectPaste.
Group/Ungroup	XintEditObjectGroup and XintEditObjectUngroup.
Back/Front	XintEditObjectBack and XintEditObjectFront.
Lower/Raise	XintEditObjectLower and XintEditObjectRaise.
Set colors	See resources **XmNcolor**, **XmNfillColor** and **XmN-stippleColor**.
Set line style	See resource **XmNlineThickness** and **XmNlineStyle**.
Set dash style	See resource **XmNdashList**.
Set fill style	See resource **XmNfillStyle**.
Set pixmap pattern	See resource **XmNfillPixmap**.

Drag and Drop

ChartObject supports full Motif drag and drop functionality (Note: drag and drop is not supported under X11R4/Motif1.1). One can drag and drop cells from a table widget into a chart object, data from a chart object into another chart object, etc. It is also possible to establish links using drag and drop between a table and a chart object or between two chart objects.

EditObject action MotifStartDrag controls the drag operation. It is connected using the following translation:

```
  <Btn2Down>: MotifStartDrag()
```

There is no specific action defined for dropping objects. The drop operation is activated automatically on a button release.

Drag and Drop operations can be disabled by setting EditObject resources **XmNallowDrag** and **XmNallowDrop** to False. Also, callback XmNdragDropCallback is invoked on both drag and drop operations and can be used to selectively enable or disable either drag or drop operations.

Default behavior

The following table summarizes the default behavior for drag and drop. If there is a key sequence in the translation, it is important that the keys remain pressed until the drag and drop operation has been completed and the button has been released.

Key Sequence	**Drag Source**	**Drop Site**	**Description**
<Btn2Down>	Chart object	EditObject	Moves chart to specified location.
Ctrl <Btn2Down>	Chart object	EditObject	Copies chart to specified location.
Ctrl Shift <Btn2Down>	Chart object	EditObject	Copies chart to specified location and makes a link between the two.
<Btn2Down>	Chart object	Chart object	Moves data from source chart to destination chart.
Ctrl <Btn2Down>	Chart object	Chart object	Copies data from source chart object into destination chart object.
Ctrl Shift <Btn2Down>	Chart object	Chart object	Copies data from source data object into destination chart object. Establish a link between the two charts.

Key Sequence (continued)	**Drag Source**	**Drop Site**	**Description**
<Btn2Down>	Chart object	EditTable	Data contained in chart moved to table at location specified by pointer.
Ctrl <Btn2Down>	Chart object	EditTable	Data contained in chart copied to table at location specified by pointer.
Ctrl Shift <Btn2Down>	Chart object	EditTable	Data contained in chart is copied to table at location specified by pointer. Establishes a link between chart and table.
<Btn2Down>	Motif Text	EditObject	Creates a text object containing the text.
<Btn2Down>	Motif Text	Chart object	Sets Chart title to be string contained in text widget.
<Btn2Down>	Motif Text	EditTable	Contents of Text widget copied into specified table cell.
<Btn2Down>	EditTable	Chart object	Data contained in selected cells copied into chart object (move option disabled for EditTable widget).
Ctrl <Btn2Down>	EditTable	Chart object	Data contained in selected cells copied into chart object.
Ctrl Shift <Btn2Down>	EditTable	Chart object	Data contained in selected cells copied into chart object. Establishes link between table and chart.
<Btn2Down>	Graphic object	EditObject	Moves object to location specified by pointer.
Ctrl <Btn2Down>	Graphic object	EditObject	Copies object to location specified by pointer.

Hardcopy

Hardcopy output functions are provided for the widgets and objects defined in the ChartObject library. Using the supplied functions, an application can produce a file containing a PostScript or CGM representation of a single widget (and all the graphic objects displayed inside) or a composite of several widgets. Once the PostScript or CGM file is generated, you can use your local print/plot utilities to produce the hardcopy output.

All the functions related to hardcopy output are defined in the CompBase widget class. See the CompBase reference section in Chapter 2 for a complete description of those functions.

PostScript Output

Function XintOutputPostscript outputs a scaled monochrome or color encapsulated PostScript description of an INT widget or a container widget such as a Form or a RowColumn widget, containing multiple INT widgets, and of all of the graphic objects contained inside. It produces an image preserving the relative layout of the child widgets. In addition to INT widgets, Motif Label, Text and TextField widgets are also recognized and output.

Function XintOutputMontagePostscript also outputs a PostScript representation of a combination of INT widgets or classes which are derived from EditObject (see EditTable for example). This function lets the application position each widget in the output display manually.

Before requesting the PostScript output function, use function XintPostScriptSetDefaults to define the characteristics (height, width, resolution) of the page to be output. By default, widget background is not painted. To specify a background color, use the function XintPostscriptSetBackground. The PostScript output function will produce a multi-page output display when the scale_factor argument in the function call is greater than one.

Layout example Figures 19 and 20 show a layout handled by the composite hardcopy output.

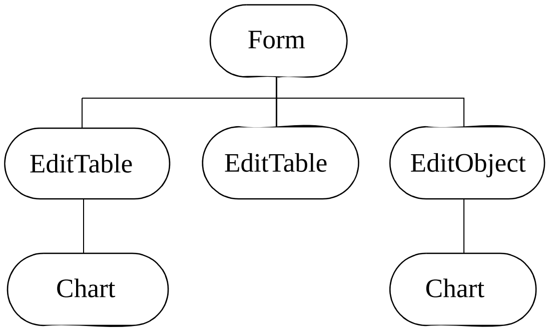
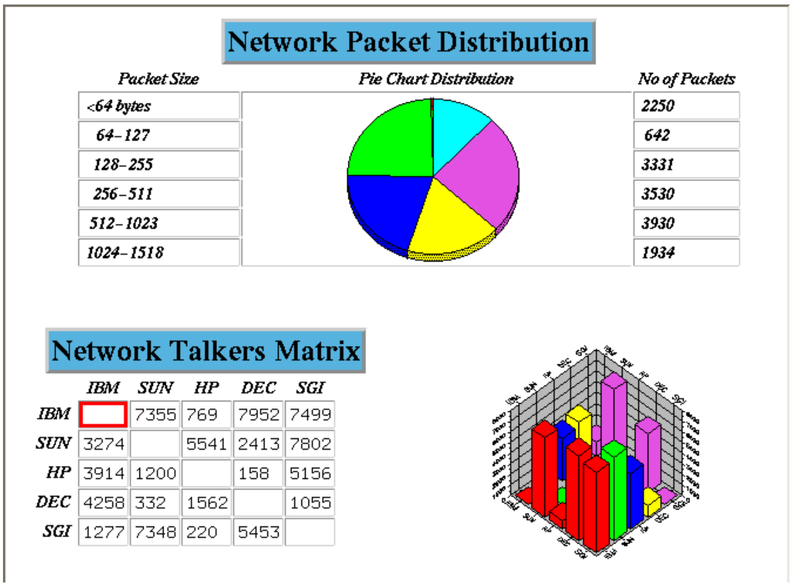


Figure 19. Composite Layout Example



Network Packet Distribution

Packet Size	Pie Chart Distribution	No of Packets
<64 bytes		2250
64–127		642
128–255		3331
256–511		3530
512–1023		3930
1024–1518		1934

Network Talkers Matrix

	IBM	SUN	HP	DEC	SGI
IBM		7355	769	7952	7499
SUN	3274		5541	2413	7802
HP	3914	1200		158	5156
DEC	4258	332	1562		1055
SGI	1277	7348	220	5453	

Figure 20. Composite PostScript Output Example

Function XintOutputMontagePostscript accepts a list of EditObject based widgets, as well as a list of coordinate specifications. This function allows the application to specify the layout of the hardcopy output. See `ChartSlideShow.c` in the `ChartDemo1` subdirectory of the `demos` directory for a code example.

CGM Output

The CGM hardcopy output offers similar functionality to that of the PostScript output. Function XintOutputCGM creates a CGM file containing the graphic representation of an EditObject widget or container widget and of all the graphic objects contained inside. Function XintOutputMontageCGM can also be used to create a composite output of multiple EditObject widgets.

Real-time Applications

The ChartObject library is well suited to handle real-time applications that continuously need to update one or multiple graphs. The DataObject library provides, for each primitive data class, a set of convenience functions to modify the data by replacing, extending or shifting the data values. The DataObject library also provides a batching mechanism, so that multiple updates for different data elements will result in only one redraw. Finally, the ChartObject supports a double-buffering mechanism which allows for smooth graph updates.

Example

In this example, we want to graph and update multiple sets of data. The horizontal direction represents the time in seconds where the samples are generated. The vertical direction represents the point amplitudes, in the application's units. The graph is initially empty. Points are first appended until we reach the number of points defined in constant MAX_POINT. After that, the data is shifted so that only the last MAX_POINT points are displayed. A complete listing of this example is available in file `ChartRealTimeApplication.c`, in directory `examples/Chart`.

The creation of the chart is shown in the following code. We create a line graph and set the vertical limits of the plot between -1 and 1, which is the range of the data values. In the horizontal direction we don't set any limits so that auto-scaling is used. We set resource **XmNxAutoRangeMode** to XintUSE_MIN_MAX so that the chart does not round-up the starting time and ending time limits for the horizontal axis.

Code

```
chart_geometry.x1 = 0.0;
chart_geometry.y1 = 0.0;
chart_geometry.x2 = 100.0;
chart_geometry.y2 = 100.0;

chart = (Object)XtVaCreateWidget("chart",
              (WidgetClass)xintChartObjectClass,
              edit,
              XmNchartTitle, "Real Time Example\nwith Data Series",
              XmNgeometry,    &chart_geometry,
              XmNchartType,   XintCHART_TYPE_LINE,
              XmNfillStyle,   XintFILL_NONE,
              NULL);

y_limits.minimum = -1;
y_limits.maximum = 1;
XtVaSetValues((Widget)
             XintChartGetComponent(chart, XintCHART_COMPONENT_PLOT),
              XmNxAutoRangeMode, XintUSE_MIN_MAX,
              XmNyLimits, &y_limits,
              NULL);
```

For the data, we will use DataSeries objects because the data points can arrive at non constant intervals. We first create a DataGroup object since we want to display multiple curves. We also create a DataLabel object that will be used to display the time values on the horizontal axis. All the data objects are created with no data initially.

```
  data_group  = (Object)XtVaCreateWidget("data_group",
                    (WidgetClass)xintDataGroupObjectClass,
                     edit, NULL);

 /*
  * We use DataSeries to be able to position samples exactly at times
  * where it is generated. Start with no data.
  */
  for (i = 0; i < num_sets; i++)
   XtVaCreateWidget("set", (WidgetClass)xintDataSeriesObjectClass,
                     edit,
                     XmNcount,       0,
                     XmNdataType,    XintDATA_TYPE_FLOAT,
                     XmNdataGroup,   data_group,
                     NULL);

/*
 * Add empty Data Label object to handle the time labels
 */
 XtVaCreateWidget("label",
(WidgetClass)xintDataLabelObjectClass,
                              edit,
                              XmNlabelCount,  0,
                              XmNdataGroup,   data_group,
                              NULL);

  XintChartAssociateData(chart, data_group);
```

The update of the data is done as follows. We first update the data series using function XintDataSeriesExtend if the number of inserted points is less that MAX_POINT and XintDataSeriesShift otherwise. We use function XintDataListIterate to retrieve the ID of the data series from the data group. The update of the data label is done in a similar fashion. We only generate an annotation every 10 points so that labels don’t overlap on the horizontal axis.

Note: We pass the label position (time value) along with the label string to the data label update functions. Finally, the whole update sequence is surrounded by calls to XintDataBatchUpdate so that only one redraw gets generated from all the changes we have made to the data.

```
 XintDataBatchUpdate(data_group, True);

 for (i=0; i<nseries; i++) {

   data_series = XintDataListIterate(&data_group, 1,
                   xintDataSeriesObjectClass, i);
   if (points_inserted < MAX_POINT)
     XintDataSeriesDataExtend(data_series, &float_time, &y[i], 1);
   else
     XintDataSeriesDataShift(data_series, &float_time, &y[i], 1);
 }

/*
 * Update the labels (generate a label only every 10th value)
 */
 if (points_inserted % 10 == 0) {

    data_label = XintDataListIterate(&data_group, 1,
                   xintDataLabelObjectClass, 0);

    sprintf(string_buffer, “%3.1f”, float_time);

    str_array[0] = string_buffer;
    if (points_inserted < MAX_POINT)
      XintDataLabelExtend(data_label, str_array, &float_time, 1);
    else
       XintDataLabelShift(data_label, str_array, &float_time, 1);
 }

 XintDataBatchUpdate(data_group, False);
```

Zoom

Zooming a display involves the enlargement of a selected part of the data being viewed. A simple zoom may be achieved through the use of Xwindow resizing facilities. In some cases this will be sufficient for the user’s needs. In other cases the user will require more control over the process in order to get a more detailed or clearer view of a portion of the data or to maintain window size requirements.

The Chart library facilitates the creation of zoom applications by providing the **XmNareaSelectionCallback** resource and the XintChartZoom convenience function. They enable the application designer to change the data displayed in the viewport easily.

In Figure 21, a subset of the data between 1993 and 2003 has been zoomed from the background plot to fill the viewport in the foreground plot. The code that produced this display follows a discussion of the viewport.

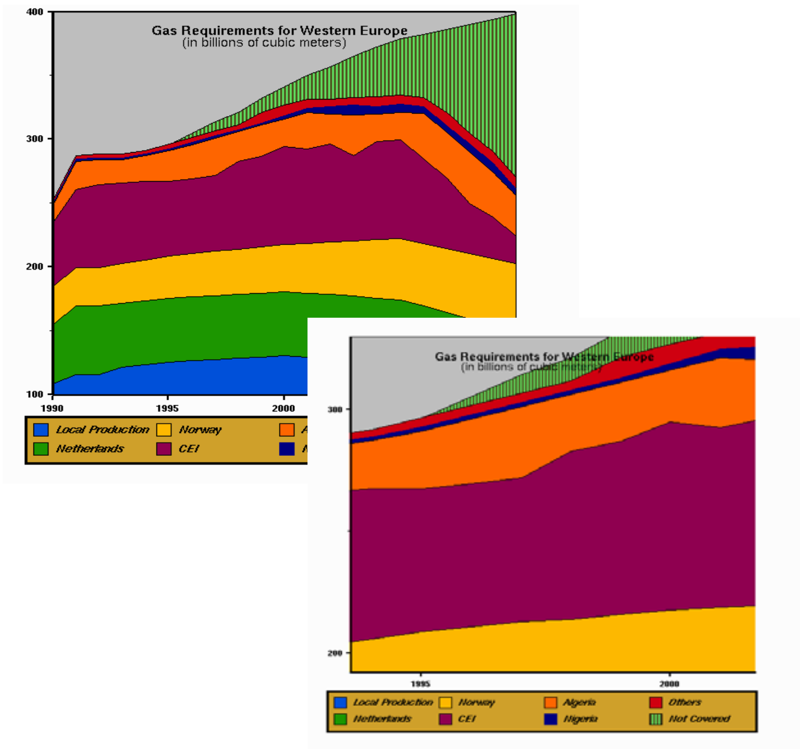


Figure 21. Chart Zoom Example

Data Viewport

The data viewport is the visible area of the plot. The plot area is controlled by the data range of the plot object as specified by the **XmNxLimits** and **XmNyLimits** resources of the Plot2D and Plot3D Object Metaclasses (Plot3D also has an **XmNzLimits** resource). These resources are pointers to type XintLimits, a structure which contains the minimum and maximum data values for the given axis.

If XintLimits is NULL, autoranging is performed and the minimum and maximum data values will be used. Either raw data values or rounded data values can be selected through the use of the **XmNxAutoRange** and **XmNyAutoRange** resources. You can activate autoranging of a single parameter in the XintLimits structure by setting it to XintUNDEFINED_FLOAT. The following code segments illustrate these capabilities.

Code

The **XmN[xy]Limits** resource is a pointer to an XintLimits structure. To assign values, declare a variable of this type:

```
/* Define the Y Axis Range */

 XintLimits yRange;

 yRange.minimum = 0.0;
 yRange.maximum = XintUNDEFINED_FLOAT;
```

- The next two assignment statements define the ChartObject and the PlotObject component of the ChartObject.

```
/* Create a Chart object */

 chart = (Object) XtVaCreateWidget("BarPlot",
                     (WidgetClass)xintChartObjectClass,
                     edit,
                     XmNgeometry, &chart_geometry,
                     XmNchartType, XintCHART_TYPE_BAR,
                     XmNchartTitle, "Yearly Sales",
                     XmNshowLegend, True,
                     NULL);

/*
 * Get the plot object ID.
 */

plot = XintChartGetComponent(chart, XintCHART_COMPONENT_PLOT);
```

Now the axis limits for the plot can be changed. Here we have specified autoranged x-axis limits which will be rounded to a multiple of the major increment for the x-axis. The y-axis range is defined by *yRange*. The minimum value has been set to a constant 0.0 and the maximum value has been set to XintUNDEFINED_FLOAT for autoranging.

```
     XtVaSetValues((Widget) plot,
                      XmNxLimits, NULL,
                      XmNxAutoRange, XintROUND_MIN_MAX,
                      XmNyLimits, &yRange,
                      NULL);
```

The XintChartZoom convenience function zooms the selected area by mapping it to the plot area. **XmNxLimits** and **XmNyLimits** are set to the range of the selected area.

```
XintChartZoom(zoom_chart, cb->x, cb->y, cb->width, cb->height);
```

Chart Zoom Example

The following code example (see `ChartZoom.c` in the `examples/Chart directory`) illustrates how to create a zoom application by using the XintChartZoom convenience function:

Code

```
include <Xm/Form.h>
include <Xm/Separator.h>
include <Xm/PushBG.h>
include <Xm/RowColumn.h>
include <Xint/Chart.h>
include <Xint/EditObject.h>

/*
 * Define PushButton controls
 */
define k_zoom         0
define k_zoom_reset   1
define k_exit         2
define k_num_controls 3
static char *control_pb_names[] =  {“Zoom”, “Zoom Reset”, “Exit”};


/*
 * Create two translation tables; one as the default and the other
 * specifically for zoom actions.
 */
static char default_translations[] =
“None <Btn1Down>: TraverseCurrent() ObjectSelect(single)\
                  ObjectEditStart() Locator()
SelectionCallback()\
                  InitAreaSelection(single)
ResourceDialog()”;

static char zoom_translations[] =
“None <Btn1Down>: TraverseCurrent() InitAreaSelection(callback)”;
```

```
XtTranslations default_translations_parsed;
XtTranslations zoom_translations_parsed;

/*
 * Other Global Variables
 */
Widget        edit;
Object        zoom_chart = NULL;


/*
 * The ZoomCallback illustrates an easy method to zoom a plot. It is
 * activated by the Btn1Down zoom button translation. To perform the
 * zoom, it calls the XintChartZoom convenience function with argument
 * values from the XintEditObjectAreaSelectionCallbackStruct.
 */
static void ZoomCallback(widget, data, cb)
  Widget widget;
  XtPointer data;
  XintEditObjectAreaSelectionCallbackStruct *cb;
{7
  Object chart = (Object) data;

  if (zoom_chart == NULL) return;

  XintChartZoom(zoom_chart, cb->x, cb->y, cb->width, cb->height);

  zoom_chart = NULL;
}


/*
 * The following code defines the callback that controls the action for
 * all of the pushbuttons.
 */
static void ControlsProc(widget, client_data, cb)
  Widget    widget;
  XtPointer client_data;
  XmAnyCallbackStruct *cb;
{
  int     code = (int) client_data;
  Object  chart, plot;
  Object  *list;
  int     count;
  register int i;

  if (code == k_exit) exit (0);

  /*
   * Retrieve object list to find ID of chart
   */
   list = XintEditObjectGetList(edit, &count);

   for (i=0; i<count; i++)
     if (XintIsChart(list[i])) break;

   if (i >= count) return;

   plot = XintChartGetComponent(list[i], XintCHART_COMPONENT_PLOT);
```

```
/*
   * Return if it is a 3D plot
   */
   if (XintIsPlot3D(plot)) return;

   if (code == k_zoom) {

      /*
       * Merge zoom_translations_parsed with the EditObject
       * widget's existing translations. Zoom will be activated
       * when the end-user defines a rectangle with Button1
       */
      XtOverrideTranslations(edit, zoom_translations_parsed);

      zoom_chart = list[i];

   } else if (code == k_zoom_reset) {

     /*
      * Reset the zoom. The plot is restored to its original
      * size by autoranging each axis.
      */
     XtVaSetValues((Widget) plot, XmNxLimits, NULL, XmNyLimits,
                   NULL, NULL);
    }
}

main(argc, argv)
  int     argc;
  char    *argv[];
{
  XtAppContext  app_context;
  Widget        top_level, main_form, separator, controls_box, pb;
  Object        chart;
  XmString      cstring;
  register int  i;

  top_level  = XtAppInitialize(&app_context, "Zoom Example",
                       (XrmOptionDescList)NULL, 0,
                       &argc, argv, NULL, NULL, 0);

 /*
  * Parse and compile the default and zoom translation tables.
  */
  default_translations_parsed =
            XtParseTranslationTable(default_translations);
  zoom_translations_parsed =
            XtParseTranslationTable(zoom_translations);

 /*
  * Create form to contain all other widgets
  */
  main_form = XtVaCreateManagedWidget("main_form",
                      xmFormWidgetClass, top_level,
                      NULL);
 /*
  * Create an EditObject widget
  */
  edit = XtVaCreateManagedWidget("edit_object",
                    xintEditObjectWidgetClass,main_form,
                    XmNwidth, 600,
                    XmNheight, 600,
                    XmNtopAttachment, XmATTACH_FORM,
                    XmNleftAttachment, XmATTACH_FORM,
                    XmNrightAttachment, XmATTACH_FORM,
                    NULL);

/*
 * Register callback to zoom a plot. The function is called when
 * a rectangle is selected if zoom has been activated
 */
 XtAddCallback(edit, XmNareaSelectionCallback,
              (XtCallbackProc) ZoomCallback, NULL);

 /*
  * Create a Separator widget
  */
  separator = XtVaCreateManagedWidget("separator",
                    xmSeparatorWidgetClass, main_form,
                    XmNtopAttachment,   XmATTACH_WIDGET,
                    XmNtopWidget,       edit,
                    XmNtopOffset,       20,
                    XmNbottomOffset,    20,
                    XmNleftAttachment,  XmATTACH_FORM,
                    XmNrightAttachment, XmATTACH_FORM,
                    NULL);

/*
 * Create controls box
 */
 controls_box = XtVaCreateManagedWidget("controls_box",
                     xmRowColumnWidgetClass, main_form,
                     XmNspacing,          15,
                     XmNorientation,      XmHORIZONTAL,
                     XmNleftAttachment,   XmATTACH_FORM,
                     XmNrightAttachment,  XmATTACH_FORM,
                     XmNbottomAttachment, XmATTACH_FORM,
                     XmNtopAttachment,    XmATTACH_WIDGET,
                     XmNtopWidget,        separator,
                     NULL)

/*
 * Create control pushbuttons
 */
 for (i=0; i<k_num_controls; i++) {
    cstring = XmStringCreateSimple(control_pb_names[i]);
    pb = XtVaCreateManagedWidget("control_buttons",
                     xmPushButtonGadgetClass, controls_box,
                     XmNpushButtonEnabled,    True,
                     XmNmarginWidth,          4,
                     XmNmarginHeight,         4,
                     XmNlabelString,          cstring,
                     NULL);
    XtAddCallback(pb, XmNactivateCallback,
               (XtCallbackProc) ControlsProc, (XtPointer)i);
    XmStringFree(cstring);
 }
```

```
/*
  * Read chart description from file
  */
  if (!XintEditObjectReadFile(edit, "chart_zoom.obj")) {
      fprintf(stderr, "Cannot find file chart_zoom.obj");
      exit(1);
  }

  XtRealizeWidget(top_level);
  XtAppMainLoop(app_context);
}
```

Customizing or Creating New Chart Types

One of the more powerful features of the ChartObject library is the ability to customize an existing chart type or to create new Chart types, different from the ones provided with the library.

Both of these features are made possible because of the object oriented nature of ChartObject, which gives the application the ability to create and insert inside a chart any object from the GraphicObject library.

Inserting Application's Defined Objects

Once a chart object is created, the application has the possibility of inserting its own objects inside the chart or the plot area of the chart. This feature can be used to insert additional labels inside chart, to annotate specific points, to insert user defined symbols into a plot, etc. Function XintChartInsertObject can be used to insert an object inside a Chart or a Plot2D object. It is not possible to insert objects inside a 3D plot.

Objects inserted inside a chart use the chart coordinate system, which ranges from 0 to 100, with the origin at the upper left corner. Objects inserted in a chart object are clipped to the chart object boundaries.

Objects inserted inside a Plot2D (BarLine, XYPlot, etc.) use the user coordinates specified by the axes attached to the plot. Objects in a plot object are clipped to the plot boundaries. Also, objects inserted in a plot object will be destroyed when the application changes plot type. Use the function XintChartGetComponent to get the object ID of the plot object to use in the XintChartInsertObject function call.

Example The following code sample illustrates how to insert an object inside a chart.

```
XintLine gline;
Object chart, line;
...
/* create a line object */
gline.start_x = gline.start_y = 0;
gline.end_x = gline.end_y = 10;
line = (Object) XtVaCreateWidget (“Line”,
                     (WidgetClass)xintLineObjectClass, edit,
                      XmNline, &gline,
                      XmNlineThickness, 3, NULL);

/* insert line object inside a chart object */
XintChartInsertObject(chart, line);
```

Customizing an Existing Chart

Example The following code example illustrates how to insert an image object inside a chart to create a shaded background. The same technique can be used to insert an image inside a plot. The full listing of the example can be found in directory `examples/Chart`, file `ComboPlot.c`. Figure 22 shows the output of this code example.

```
 Object chart, plot, image;
 Pixel pixel_array[32];
 int   ncolors = 32;
 XintRectangle rect;

...
 /*
  * We assume we have allocated 32 colors containing a continuous
  * color spectrum. We now create a pixmap of width 1 and
  * height 32. The ImageObject will automatically interpolate the
  * pixmap to fit the chart size.
  */
  pixmap = XCreatePixmap(display,
                 RootWindow(display, IntScreenNumber(top_level)),
                 1, ncolors,
                 DefaultDepth(display,
IntScreenNumber(top_level)));

gc = XCreateGC(display, pixmap, 0, NULL);
  for (i=0; i<ncolors; i++) {
     XSetForeground(display, gc, pixel_array[ncolors-i-1]);
     XDrawPoint(display, pixmap, gc, 0, i);
   }

...
    /* Create a chart inside the edit object */
    chart_geometry.x1 = chart_geometry.y1 = 0;
    chart_geometry.x2 = chart_geometry.y2 = 100;
    chart = (Object) XtVaCreateWidget("combination_chart",
                 (WidgetClass)xintChartObjectClass, edit,
                 XmNgeometry, &chart_geometry,
                 XmNchartType, XintCHART_TYPE_COMBINATION,
                 XmNchartTitle, "Dow Jones Industrial Average",
                 NULL);
...
```

```
    /*
     * Create image object, "pixmap", as the background of the chart
     */
     rect.x1 = 0;
     rect.y1 = 0;
     rect.x2 = 100;
     rect.y2 = 100;
     image = (Object) XtVaCreateWidget("Image",
                        (WidgetClass)xintImageObjectClass,
                        edit,
                        XmNsensitive, False,
                        XmNimagePixmap, pixmap,
                        XmNimageColorRecord, color_record,
                        XmNrectangle, &rect,
                        NULL);

     plot = XintChartGetComponent(chart, XintCHART_COMPONENT_PLOT);


    /*
     * Make plot transparent so that we can see the image
     */
     XtVaSetValues((Widget) plot, XmNfillStyle, XintFILL_NONE, NULL);

    /*
     * Insert object into to the chart and lower it
     */
     XintChartInsertObject(chart, image);
     XintEditObjectBack(edit, image);
...
```

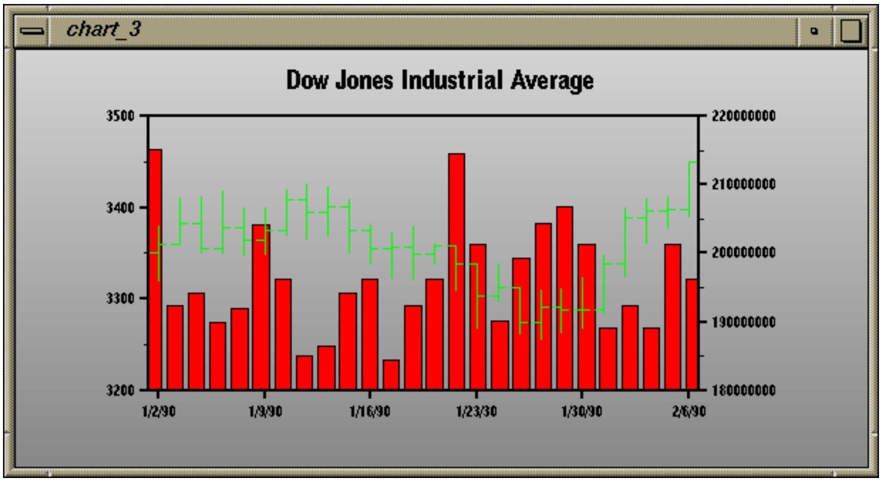


Figure 22. Customized Chart

Creating a New Chart Type

ChartObject has a built-in type for all of the usual kinds of charts. However, it is flexible enough to let the application programmer easily create entirely new chart types. The following example illustrates how to use the Line object to create a chart representing a vector field. Figure 23 shows the output of this example. The full listing can be found in file `ChartField.c` in the `examples/Chart` directory.

Code

```
#include <Xint/Chart.h>
#include <Xint/EditObject.h>
#include <Xint/Line.h>
#include <math.h>

extern double drand48();

#define COUNT 1000

main(argc, argv)
  int     argc;
  char    *argv[];
{
  XtAppContext  app_context;
  Widget        top_level;
  Widget        edit;
  Object        chart, plot;
  XintGeometry  chart_geometry;
  XintLimits    limits;
  XintIncrements increment;
  Pixel         pixel;
  register      int i;

  top_level  = XtAppInitialize(&app_context, "chart_field",
                     (XrmOptionDescList)NULL, 0,
                     &argc, argv, NULL, NULL, 0);

   /* Create an EditObject widget*/
   edit = XtVaCreateManagedWidget("edit_object",
                xintEditObjectWidgetClass,top_level,
                XmNwidth, 600, XmNheight, 600,
                XmNtitle, "Vector Field", NULL);

    /* Create Chart object */
    chart_geometry.x1 = 0;
    chart_geometry.y1 = 0;
    chart_geometry.x2 = 100;
    chart_geometry.y2 = 100;
    chart = (Object) XtVaCreateWidget("LinePlot",
                     (WidgetClass)xintChartObjectClass, edit,
                     XmNgeometry, &chart_geometry,
                     XmNchartType, XintCHART_TYPE_LINE,
                     XmNchartTitle, "Vector FieldPlot",
                     XmNresourceDialog, False,
                     NULL);

  plot = XintChartGetComponent(chart, XintCHART_COMPONENT_PLOT);
  /* Set the plot limits and increments */
  limits.minimum = 0;
  limits.maximum = 1.0;
  increment.minor_increment = .25;
  increment.major_increment = .5;
  XtVaSetValues(plot,
                XmNxLimits, &limits,
                XmNyLimits, &limits,
                XmNxIncrements, &increment,
                XmNyIncrements, &increment,
                XmNxAxisPlacement, XintPLACEMENT_TOP_BOTTOM,
                XmNyAxisPlacement, XintPLACEMENT_LEFT_RIGHT,
                XmNresourceDialog, False,
                NULL);

  srand48((long) 100);

  pixel = XintLoadColor(XtDisplay(edit), "red");

  for (i=0; i<COUNT; i++) {
     Object arrow;
     XintLine line;
     float x = drand48();
     float y = drand48();
     float size = .05;
     float angle = sin((double) 3.14 * x) + .2 * cos((double) 3.14 * y);

     line.start_x = x;
     line.start_y = y;
     line.end_x = x + size * cos((double) angle);
     line.end_y = y + size * sin((double) angle);

    /*
     * Create arrow object
     * Note: the color and the size of the arrows are the same
here,
     *      but they could be set to be different for each object.
     */
     arrow = (Object) XtVaCreateWidget("arrow",
                   (WidgetClass) xintLineObjectClass, edit,
                   XmNlineEnd, XintEND_ARROW,
                   XmNarrowStyle, XintSTICK,
                   XmNarrowLength, 4,
                   XmNtipAngle, 15,
                   XmNline, &line,
                   XmNlineThickness, 1,
                   XmNsensitive, False,
                   XmNcolor, pixel,
                   NULL);
/*
     * Insert arrow object inside the plot
     */
     XintChartInsertObject(plot, arrow);
  }
```

```
    XtRealizeWidget(top_level);
    XtAppMainLoop(app_context);
}
```

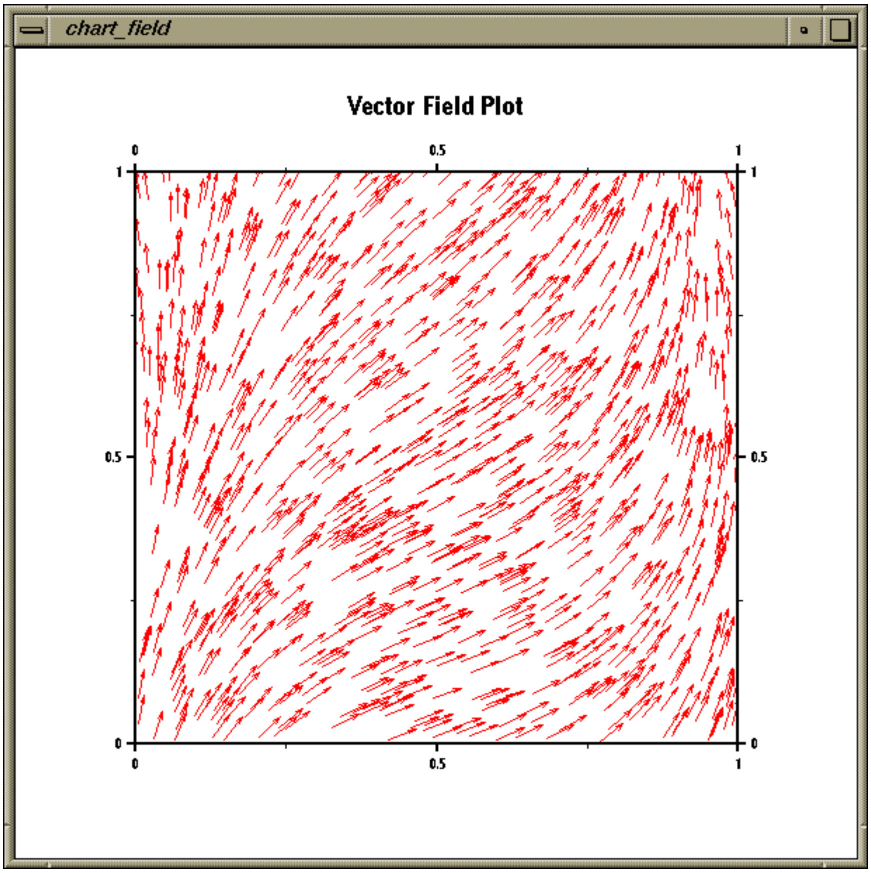


Figure 23. Creating a New Chart Type

Widget Reference

2

Overview

This chapter includes the following sections:

- **CompBase Widget Metaclass** on page 70
- **EditObject Widget Class** on page 81
- **ObjectEditor Widget Class** on page 108

CompBase Widget Metaclass

CompBase is a base widget class that handles hardcopy output for its subclasses, including EditObject and EditTable. The CompBase widget class defines a set of functions for producing disk files containing CGM or PostScript representations of the graphical display of a widget and of its content. CompBase also handles the hardcopy output of the composition of multiple widgets. A composite image must be of multiple EditObject based widgets contained within a Composite widget such as a Motif Form or RowColumn widget.

In addition to hardcopy, CompBase provides a set of convenience functions to map user coordinates to and from device coordinates. The CompBase widget class is a metaclass and cannot be instantiated directly.

Inherited Behavior and Resources

The CompBase widget inherits behavior and resources from the *Core, Composite* and *Manager* classes.

- Class pointer is *xintCompBaseWidgetClass.*
- Class name is *XintCompBase.*
- Header file is included as `<Xint/CompBase.h>`.

CompBase Resources

The following resources are defined by the CompBase class:

Name	Type	Default
XmNfontPath	char *	NULL
XmNwarning	int	XintWARNING_POST

XmNfontPath

Some subclasses of CompBase and some object classes such as Text or Symbol may require the use of scalable fonts, which are displayed using an outline font technology provided with the INT library. Resource **XmNfontPath** can be used to specify the path to the directory where the files containing the font outlines reside. Alternatively, you can use environment variable INT_FONT_PATH to specify this directory. If neither resource **XmNfontPath** nor environment variable INT_FONT_PATH is set, the current directory will be searched.

XmNwarning

Specifies the destination of warning messages that an INT widget might need to display. Use one of the following defined integer constants when specifying a value for this resource:

Resource Value	Description
XintWARNING_NONE	No message will be output.
XintWARNING_PRINT	Message will be written to *stderr.*
XintWARNING_POST (default)	Message will be displayed in a dialog box.

You can combine destinations using a logical OR or an arithmetic + operation. For instance, specify XintWARNING_PRINT + XintWARNING_POST to have any warning message displayed on the screen and written to *stderr.*

CompBase Functions

The following functions are defined for hardcopy output and coordinate system transformations. All of these functions can be applied to any widget instance of a class derived from the CompBase widget class (such as EditObject or EditTable). In addition, the composite hardcopy functions can be applied to any instance of a composite widget such as a widget instantiated from the Motif Form, or the Motif BulletinBoard widget classes.

Function	Description
XintCGMDrawBox	To tell the CGM output whether or not to draw a box around a CGM plot.
XintCGMGetDimensions	Gets the size in inches used by a widget.
XintCGMPixelToInch	Converts a size specified in pixels to a size specified in inches.
XintCGMSetEdgeWidthMode	Sets the edge width for drawing primitives in the CGM display.
XintCGMSetLineWidthMode	Sets the line width for drawing primitives in the CGM display.
XintCGMSetVDCType	Selects either real or integer output coordinates for CGM.
XintGetWidgetSize	Returns the size in pixels that a widget will occupy when output to hardcopy.

Function (continued)	**Description**
XintHorizontalPixelToUser	Pixel to User coordinates conversion in the horizontal direction.
XintHorizontalUserToPixel	User coordinates to Pixel conversion in the horizontal direction.
XintOutputCGM	Creates a CGM file containing the graphic representation of a single widget or of all of the widgets that are children of a composite widget.
XintOutputMontageCGM	Creates a CGM file containing a montage composed of several widgets.
XintOutputMontagePostscript	Creates a PostScript file containing a montage composed of several widgets.
XintOutputPostscript	Creates a PostScript file containing the graphic representation of a single widget or of all of the widgets that are children of a composite widget.
XintPostscriptGet Defaults	Gets the PostScript page characteristics.
XintPostscriptSetBackground	Sets the background for PostScript output.
XintPostscriptSet Defaults	Sets the PostScript page characteristics.
XintVerticalPixelToUser	Pixel to User coordinates conversion in the vertical direction.
XintVerticalUserToPixel	User coordinates to Pixel conversion in the vertical direction.

XintCGMDrawBox

Sets a flag which tells the CGM output routines XintOutputCGM and XintOutputMontageCGM whether or not to draw a rectangular box around the plot.

```
void XintCGMDrawBox (Boolean flag)
```

where *flag* is a Boolean variable that should be set to True to have the box drawn around the plot.

XintCGMGetDimensions

Returns the dimensions (in inches) that a widget (or combination of widgets) will occupy when mapped to a plot. You usually call this function prior to calling a CGM hardcopy function so that you can specify the appropriate dimensions in the CGM hardcopy function call.

```
void XintCGMGetDimensions (...)
```

Widget	widget	The ID of the widget to be output.
float *	width	Width in inches of the widget's extent.
float *	height	Height in inches of the widget's extent.

XintCGMPixelToInch

Converts a size specification from pixels to inches. This function can be used to provide the plot size specification in inches required by function XintOutputMontageCGM.

```
void XintCGMPixelToInch (...)
```

Widget	widget	The ID of the widget.
int	pwidth	Width in pixels.
int	pheight	Height in pixels.
float *	width	Returns the width in inches.
float *	height	Returns the height in inches.

XintCGMSetEdgeWidthMode

Controls the mode for the edge width of filled polygons in the CGM output.

```
void XintCGMSetEdgeWidthMode (...)
```

int	mode	Specify one of the values below.

The argument *mode* must be specified as one of the following:

Resource Value	**Description**
XintCGM_WIDTH_MODE_ABSOLUTE	The edge width in the CGM display is the edge width of the primitive multiplied by the nominal line width of the target device, usually one pixel or device coordinate.
XintCGM_WIDTH_MODE_SCALED (default)	The edge width in the CGM display is specified in the world coordinates of the plot.

XintCGMSetLineWidthMode

Controls the mode for the line width of lines and polygons in the CGM output.

```
void XintCGMSetLineWidthMode (...)
```

int	mode	Specify one of the values below.

The argument *mode* must be specified as one of the following:

Resource Value	**Description**
XintCGM_WIDTH_MODE_ ABSOLUTE	The line width in the CGM display is the line width of the primitive multiplied by the nominal line width of the target device, usually one pixel or device coordinate.
XintCGM_WIDTH_MODE_SCALED (default)	The line width in the CGM display is specified in the world coordinates of the plot.

XintCGMSetVDCType

Allows the application to globally select the type of CGM output file to be created. Output coordinate data can be either real, the default, or integer. The integer type is provided because some CGM previewers and rasterizers do not support the floating point format.

```
void XintCGMSetVDCType (...)
```

int	type	Specify one of the values below.

The argument *type* must be specified as one of the following defined constants.

Resource Value	**Description**
XintCGM_VDC_TYPE_INTEGER	The coordinates are output in 16 bit integer format, as required by the CGM/PIP (Petroleum Industry Profile) specification.
XintCGM_VDC_TYPE_REAL	The coordinates are output in fixed point floating format, as required by the CGM/PIP (Petroleum Industry Profile) specification. This is the default.

XintGetWidgetSize

Returns the size in pixels that a widget will occupy when output to hardcopy. This function is primarily used in conjunction with functions

XintOutputMontagePostscript or XintOutputMontageCGM to position the widgets to be output. The size returned by this function is equal to the widget size, except when the argument is a Motif ScrolledWindow widget, in which case it will return the full size of the child widget.

```
void XintGetWidgetSize (...)
```

Widget	widget	The ID of the widget.
int *	width	Returns the width in pixels.
int *	height	Returns the height in pixels.

XintHorizontalPixelToUser

Converts a pixel coordinate into the corresponding user coordinate using the default horizontal coordinate system of a widget whose class is derived from the XintCompBase widget class.

```
Boolean XintHorizontalPixelToUser (...)
```

Widget	widget	The ID of the CompBase derived widget.
int	pixel	Specifies the horizontal window coordinate.
float *	user	Returns the corresponding user coordinate.

The function returns False if *pixel* is outside the widget’s window.

XintHorizontalUserToPixel

Converts a user coordinate into the corresponding pixel coordinate using the default horizontal coordinate system of a widget whose class is derived from the XintCompBase widget class.

```
Boolean XintHorizontalUserToPixel (...)
```

Widget	widget	The ID of the CompBase derived widget.
float	user	Specifies the horizontal user coordinate.
int *	pixel	Returns the corresponding window coordinate.

The function returns False if *user* is outside the widget’s window.

XintOutputCGM

Writes a color CGM description of a widget or of the contents of a container widget to a disk file. The geometry of the widgets inside a container widget is preserved. However, widgets contained in a Motif ScrolledWindow widget are expanded to their full size and the scrollbars are not displayed. Only widgets instances of INT

CompBase, Scroll, or Motif ScrolledWindow, Label, Text or TextField, or one of their subclasses will be output.

```
Boolean    XintOutputCGM (...)
```

Widget	widget	Widget for output.
char *	filename	Name of CGM file to be created.
float	plot_width	Specifies the width in inches of the CGM plot to be generated.
float	plot_height	Specifies the height in inches of the CGM plot to be generated.

In case of error, the function returns a warning message to the end-user (as controlled by resource **XmNwarning**) and returns False. Otherwise, it returns True.

XintOutputMontageCGM

Writes a color CGM file containing a montage of several widgets into a canvas. The canvas size and the widget positions inside the canvas are specified in pixel units. The convenience function XintGetWidgetSize can be used to obtain the size in pixels of each widget. The widgets in the list must be of a class derived from INT CompBase, Scroll, or Motif ScrolledWindow, Label, Text or TextField.

```
Boolean XintOutputMontageCGM (...)
```

Data Type	Arg Name	Description
Widget *	widget_list	List of widgets to output.
int *	xpos_list	List of x coordinates for the widgets.
int *	ypos_list	List of y coordinates for the widgets.
int	count	Number of widgets to output.
char *	filename	Name of CGM file to be created.
int	canvas_width	Width of canvas in pixels.
int	canvas_height	Height of canvas in pixels.
float	width	Width of plot in inches.
float	height	Height of plot in inches.

In case of error, the function displays a warning message (as controlled by resource **XmNwarning**) to the end user and returns False; otherwise, it returns True.

XintOutputMontagePostscript

Writes a color or monochrome PostScript file containing a montage of several widgets into a canvas. The canvas size and the widget positions inside the canvas are specified in pixel units. Convenience function XintGetWidgetSize can be used to obtain the size in pixels of each widget. The widgets in the list must be from a class derived from INT CompBase, Scroll, or Motif ScrolledWindow, Label, Text or TextField.

```
Boolean XintOutputMontagePostscript (...)
```

Widget *	widget_list	List of widgets to output.
int *	xpos_list	List of x coordinates for the widgets.
int *	ypos_list	List of y coordinates for the widgets.
int	count	Number of widgets to output.
char *	filename	Name of PostScript file to be created.
int	canvas_width	Width of canvas in pixels.
int	canvas_height	Height of canvas in pixels.
float	scale_factor	Specify a real number greater than 0 (see below).
int	color_model	Specify XintMONOCHROME for monochrome output or XintCOLOR for color output.
int	orientation	Specify one of the values below.

When the *color_model* argument is specified as XintCOLOR for a monochrome device, a grayscale display will be produced. Specification of XintMONOCHROME sets all lines and text to black. All fill areas will be grayscale.

The argument *orientation* must be specified as one of the following defined constants:

Resource Value	**Description**
XintORIENTATION_PORTRAIT	Image will be oriented as on screen.
XintORIENTATION_LANDSCAPE	Image will be rotated 90 degrees clockwise from the screen image.
XintORIENTATION_AUTOMATIC	Image will be oriented so that the longest dimension (height or width) will be along the longest dimension of the page.

The *scale_factor* argument in the function call specifies how the image inside the widget window will be scaled when output to the PostScript file. If you specify 1, then the image will be fitted to the page. If you specify a fractional number greater than 0 and less than 1, then the image will be scaled to that fraction of the page. If you specify a number greater than 1, then the image will be scaled by that number and multiple pages, as required by the amount of scaling, will be output to the PostScript file.

In case of error, displays a warning message to the end user (as controlled by resource **XmNwarning**) and returns False; otherwise, returns True.

XintOutputPostscript

Writes a scaled monochrome or color PostScript description of an INT widget, or of all of the widgets inside of a container widget, to a disk file. When a container widget is specified, the geometry of the widgets inside the container widget is preserved. However, widgets contained in a Motif ScrolledWindow widget or an INT Scroll widget are expanded to their full size and the scrollbars are not displayed. Only widgets that are instances of INT CompBase, Scroll, or Motif ScrolledWindow, Label, LabelGadget, Text or TextField, or one of their subclasses will be output.

```
Boolean XintOutputPostscript (...)
```

Widget	widget	Widget for output.
char *	filename	Name of PostScript file to be created.
float	scale_factor	Specify a real number greater than 0 (see below).
int	color_model	Specify XintMONOCHROME for monochrome output or XintCOLOR for color output.
int	orientation	Specify one of the values below.

When the *color_model* argument is specified as XintCOLOR for a monochrome device, a grayscale display will be produced. Specification of XintMONOCHROME sets all lines and text to black. All fill areas will be grayscale. The argument *orientation* must be specified as one of the following:

Resource Value	Description
XintORIENTATION_PORTRAIT	Image will be oriented as on screen.
XintORIENTATION_LANDSCAPE	Image will be rotated 90 degrees clockwise from the screen image.
XintORIENTATION_AUTOMATIC	Image will be oriented so that the longest dimension (height or width) will be along the longest dimension of the page.

The *scale_factor* argument in the function call specifies how the image inside the widget window will be scaled when output to the PostScript file. If you specify 1, then the image will be fitted to the page. If you specify a fractional number greater than 0 and less than 1, then the image will be scaled to that fraction of the page. If you specify a number greater than 1, then the image will be scaled by that number and multiple pages, as required by the amount of scaling, will be output.

In case of error, the function displays a warning message to the end user (as controlled by resource **XmNwarning**) and returns False; otherwise, it returns True.

XintPostscriptGetDefaults

Obtains the PostScript output page characteristics set by using function XintPostscriptSetDefaults.

```
void XintPostscriptGetDefaults (...)
```

int *	resolution	Returns a pointer to an integer specifying the page resolution in dots per inch.
float *	page_width	Returns a pointer to a floating point number specifying the page width in inches.
float *	page_height	Returns a pointer to a floating point number specifying the page height in inches.

XintPostscriptSetBackground

Sets the background color for the PostScript output. By default, the PostScript output will not paint the background. Use this function if you want the plot background to be painted.

```
void XintPostscriptSetBackground (Pixel fill_color)
```

where *fill_color* is a Pixel value. Set *fill_color* to XintNO_FILL to have no background painted.

XintPostscriptSetDefaults

Sets PostScript output page characteristics used by XintOutputPostscript.

```
void XintPostscriptSetDefaults (...)
```

int	resolution	Specifies the page resolution in dots per inch.
float	page_width	Specifies the page width in inches.
float	page_height	Specifies the page height in inches.

XintVerticalPixelToUser

Converts a pixel coordinate into the corresponding user coordinate in the default vertical coordinate system of a widget whose class is derived from the XintCompBase widget class.

```
Boolean XintVerticalPixelToUser (...)
```

Widget	widget	ID of the CompBase widget.
int	pixel	Specifies the pixel location in the vertical direction.
float *	user	Returns the user coordinate corresponding to argument *pixel*.

Returns False if argument *pixel* is outside the widget’s boundaries.

XintVerticalUserToPixel

Converts a user coordinate into the corresponding pixel coordinate using the default vertical coordinate system of a widget whose class is derived from the XintCompBase widget class.

```
Boolean XintVerticalUserToPixel (...)
```

Widget	widget	ID of the CompBase widget.
float	user	Specifies the user location in the vertical direction.
int *	pixel	Returns the pixel coordinate corresponding to argument *user.*

Returns False if argument *user* is outside the widget’s boundaries.

EditObject Widget Class

The EditObject widget class provides support for displaying, editing, and storing/retrieving graphic objects based on the Graphic object class. Any widget class that is a subclass of the EditObject widget class inherits the ability to display, edit, and store/retrieve graphic objects.

Display and editing capabilities of EditObject class are implemented through a comprehensive set of actions, callbacks and convenience functions. Editing capabilities include the ability to select one or more objects and to move, size or shape objects. A set of convenience functions is provided for saving and restoring objects to or from an ASCII file. A clipboard mechanism provides cut and paste functionality inside an application or between two different applications that use EditObject widgets (or widgets from a subclass of the EditObject widget class).

Coordinate System

The EditObject class defines a linear coordinate system that is between 0.0 and 100.0 both horizontally and vertically. You can use the functions provided in the CompBase widget class to do transformations between pixel values and the EditObject coordinate system.

Object Selection

Once created, you can select an object using BSelect. Handles displayed around the edges of a selected object, or group of objects, indicate its status. Multiple selection and grouping/ungrouping of objects is also supported.

Object Editing

Objects can be moved, sized and shaped interactively. Size and Shape operations are identical for all rectangular Graphic objects (such as Oval and TextObj). For objects instantiated using subclasses of the MultiPoint class (such as PolyLine and Wavelet), the Shape operation is equivalent to a Move point operation. The Move, Size and Shape operations are activated using BSelect Drag and terminated using BSelect Release. A special operation, called Adjust, combines the Shape and Move operations into one action. If the user performs BSelect Drag close to a handle, a Shape operation will be performed, otherwise a Move operation is performed. Finally, a specific set of actions is provided to interactively add or remove points for MultiPoint based objects.

Note: Objects can also be edited from the program, just like any other Motif widget, using the XtSetValues or XtVaSetValues calls.

Object Display

Objects are displayed in the order they have been created. The object created first is drawn first, while the object created last is drawn last and thus will appear to be on top of the other objects. A set of convenience functions allows the application to move objects forward or backward in the display order.

Input/Output

The EditObject widget class provides a set of convenience functions to write a list of objects into a file and to retrieve them later on. The objects in an object description file can be read by any widget created from a subclass of the EditObject widget class.

Clipboard

A clipboard mechanism is implemented for Cut, Copy and Paste operations on Graphic objects. The clipboard mechanism provides cut/copy/paste operations among EditObject widgets inside the current application or between two different applications. For example, it is possible to Cut an object from an EditObject widget in one application and Paste it into an EditObject widget belonging to another application.

Locator

The EditObject provides a callback and a set of actions that allow the application to track the cursor location.

EditObject Widget Appearance

The EditObject widget is an empty rectangular window. Figure 24 shows a Chart object and several Graphic objects created in an EditObject widget:

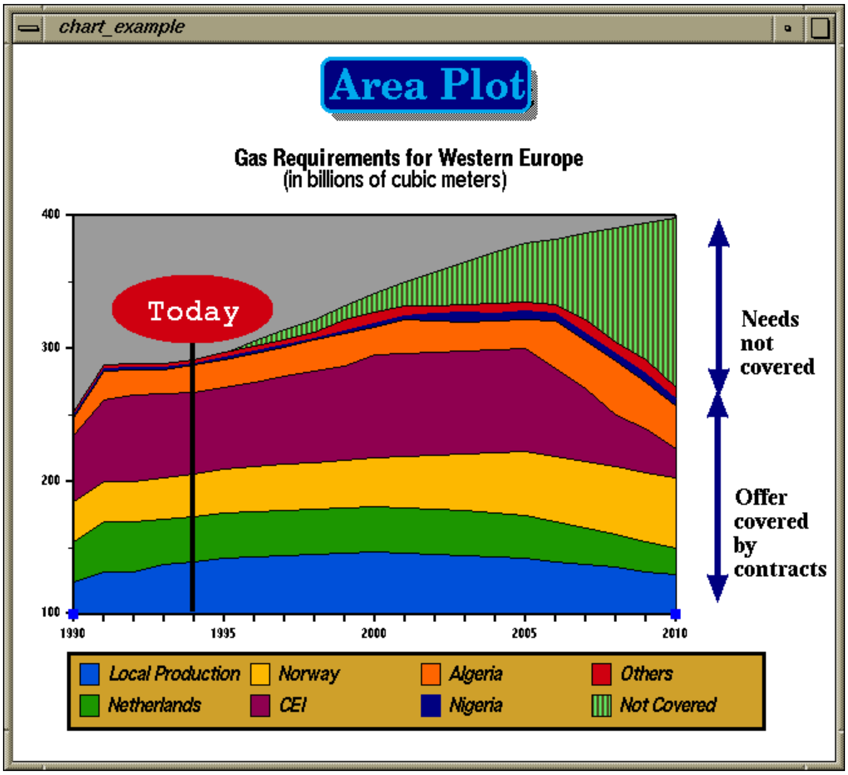


Figure 24. EditObject Containing a Chart and Various Graphic Objects

Inherited Behavior and Resources

The EditObject widget inherits behavior and resources from the *Core*, *Composite*, *Constraint*, *Manager* and *CompBase* classes.

- Class pointer is *xintEditObjectWidgetClass.*
- Class name is *XintEditObject.*
- Header file is included as `<Xint/EditObject.h>`

EditObject Resources

The following resources are defined by the EditObject widget class.

Name	Type Default	Access
XmNallowDrag	Boolean False	CSG
XmNallowDrop	Boolean False	CSG
XmNareaSelectionCallback	XtCallbackList NULL	C
XmNcopyCallback	XtCallbackList NULL	C
XmNcursorType	int XC_crosshair	CSG
XmNcutCallback	XtCallbackList NULL	C
XmNdragDropCallback	XtCallbackList NULL	C
XmNeditObjectCallback	XtCallbackList NULL	C
XmNflip	Boolean False	CSG
XmNhandleColor	Pixel “blue”	CSG
XmNhandleSize	int 4	CSG
XmNinsertObjectCallback	XtCallbackList NULL	C
XmNlocatorCallback	XtCallbackList NULL	C
XmNobjectDeselectionCallback	XtCallbackList NULL	C
XmNobjectEditMode	int XintEDIT_NONE	CSG
XmNobjectSelectionCallback	XtCallbackList NULL	C
XmNpasteCallback	XtCallbackList NULL	C

Name (continued)	Type Default	Access
XmNpointSelectionTolerance	int 4	CSG
XmNresourceDialogCallback	XtCallbackList NULL	C
XmNrubberbandCallback	XtCallbackList NULL	C
XmNselectionCallback	XtCallbackList NULL	C

XmNallowDrag

Specifies whether or not the widget can be used as a drag site for Motif drag and drop operations. Refer to *"XmNdragDropCallback"* on page 86 for more information about selectively allowing or disallowing drag operations.

XmNallowDrop

Specifies whether or not the widget can be used as a drop site for Motif drag and drop operations. Refer to *"XmNdragDropCallback"* on page 86 for more information about selectively allowing or disallowing drag operations.

XmNareaSelectionCallback

Specifies a list of callbacks that is called when a user has selected a rectangular area in the widget window. The callback list is called by the action EndAreaSelection. The coordinates of the selection are returned in the callback structure. Each subclass of the EditObject widget class inherits this resource, but returns a unique callback structure to the associated callback list. For the EditObject widget class, the callback structure returned is XintEditObjectAreaSelectionCallbackStruct.

XmNcopyCallback

Specifies a list of callbacks that is called when function XintEditObjectCopy is invoked. The list of selected objects is returned in the XintEditObjectCallbackStruct callback structure. The reason sent by the callback is XintCR_COPY.

XmNcursorType

Specifies the type of cursor to display in the EditObject widget window. Specify any valid cursor defined by the X Window System or specify XintCROSS_HAIR_CURSOR to obtain a drawn cross hair cursor. The cross hair cursor displays horizontal and vertical lines that intersect at the cursor location and which extend across the EditObject widget window.

XmNcutCallback

Specifies a list of callbacks that is called when function XintEditObjectCut is invoked. The list of selected objects is returned in the XintEditObjectCallbackStruct callback structure. The reason sent by the callback is XintCR_CUT.

XmNdragDropCallback

Specifies a list of callbacks called when application initiates a Motif drag or drop operation. This callback will only be called if resource **XmNallowDrag** (for a drag) or **XmNallowDrop** (for a drop) are set to True. The action controlling the drag operation is MotifDragStart. There is no specific action for the drop operation.

XmNeditObjectCallback

Specifies a list of callbacks called when a graphic object is edited interactively. Supported operations are object move and shape. The callback structure is XintEditObjectCallbackStruct. Reasons returned by the callback are XintCR_OBJECT_EDIT_START, XintCR_EDIT_OBJECT_EDIT and XintCR_OBJECT_EDIT_END.

XmNflip

Specifies how objects based on the Graphic class that have sampled data (such as Wavelet or LogCurve) are drawn. When this resource is set to True, the data samples are associated with the vertical axis. When the resource is False, the data samples are associated with the horizontal axis.

XmNhandleColor

Specifies pixel colors used to draw handle bars of Graphic object when selected.

XmNhandleSize

Specifies size in pixels of handle bars drawn when an Graphic object elected.

XmNinsertObjectCallback

Specifies a list of callbacks that is called after the completion of an object insert operation, initiated using function XintEditObjectInsert.

XmNlocatorCallback

Specifies a list of callbacks that is called by the Locator action. This action is typically connected to the cursor movement by the translation table. Every subclass of the EditObject widget class inherits this resource, but some subclasses return a unique callback structure to the associated callback list. For the EditObject widget class, the callback structure is XintEditObjectLocatorCallbackStruct.

XmNobjectDeselectionCallback

Specifies a list of callbacks that is called when the function

XintEditObjectDeselect-Object is called or when a Graphic object is deselected in the EditObject widget's window. Both the deselected object and the list of objects that remain selected are returned in the callback structure.

XmNobjectEditMode

Specifies the edit mode for the Graphic objects contained in the EditObject window. Most basic editing operations defined on Graphic objects are handled by the editing actions (ObjectEditStart, ObjectEdit and ObjectEditEnd). If the action ObjectEditStart has no argument specified, then the corresponding editing operation is defined using the resource **XmNobjectEditMode**. Two functions, XintEditObjectSetEditMode and XintEditObjectGetEditMode, set and get the value of this resource. You can specify one of the following constants for the value of this resource:

Resource Value	Description
XintEDIT_NONE	ObjectEdit actions do nothing.
XintEDIT_MOVE	ObjectEdit actions implement a Move operation.
XintEDIT_SIZE	ObjectEdit actions implement a Size operation.
XintEDIT_SHAPE	ObjectEdit actions implement a Shape operation.
XintEDIT_ADJUST	ObjectEdit actions implement an Adjust operation. Adjust is a Shape operation if the object selection is close to a handle bar or a Move operation if the object selection is anywhere else inside the object.
XintEDIT_RUBBERBAND	ObjectEdit actions implement a Rubberband operation. Callback XmNrubberbandCallback is invoked continu-ously as the pointer moves.
XintEDIT_INSERT	Object Edit actions implement interactive object creation. Do not specify this value directly, but use the function XintEditObjectInsert instead.

XmNobjectSelectionCallback

Specifies a list of callbacks that is called when function XintEditObjectSelectObject is called or when a Graphic object is selected in the widget window. Both the selected object and the list of currently selected objects are returned in the XintEditObjectSelectionCallbackStruct callback structure. The reason sent by the callback is XintCR_OBJECT_SELECTION.

XmNpasteCallback

Specifies a list of callbacks that is called when function XintEditObjectPaste is called. The list of selected objects is returned in the XintEditObjectEditCallbackStruct callback structure. The reason sent by the callback is XintCR_PASTE.

XmNpointSelectionTolerance

Specifies the margin of tolerance, in pixels, for selecting an object or an object's point.

XmNresourceDialogCallback

Specifies a list of callbacks that is called when action ResourceDialog is invoked. Some objects, such as Text, AxisObject or Chart have a built-in resource editor that is activated from action ResourceDialog. Callback XmNresourceDialogCallback can be used to prevent the built-in editor from being activated, so that the application can provide its own resource editor. This callback can also be used by the application to provide a resource editor for objects that don't have a built-in one.

XmNrubberbandCallback

Specifies a list of callbacks to be called by actions ObjectEditStart, ObjectEdit and ObjectEditMode, when the XmNobjectEditMode resource is set to XintEDIT_RUBBERBAND. If the XmNobjectEditMode resource is set to XintEDIT_RUBBERBAND then the corresponding action does not do anything but invoke this callback list. It is up to the application to implement rubberbanding.

XmNselectionCallback

Specifies a list of callbacks that is called when a user selects the EditObject widget. The coordinates of the selection are returned in the XintEditObjectCallbackStruct callback structure. The reason sent by the callback is XintCR_SELECTION.

EditObject Actions

The following action procedures are defined by the EditObject widget and can be tied to user inputs via a translation table.

Action Name	**Description**
ChangeCursorMask()	Changes color of cross hair cursor in increment.
DrawCursor()	Draws cross hair cursor at location of mouse pointer.
EndDrawCursor()	Terminates cross hair drawing operation.
InitDrawCursor()	Initiates cross hair drawing operation.
TraverseCurrent()	Moves focus to the current widget.
PreviousTabGroup()	Move focus to the previous tab group.
NextTabGroup()	Moves focus to the next tab group.
Increment(left)	Scrolls left one increment (if implemented by subclass).
Increment(right)	Scrolls right one increment (if implemented by subclass).
Increment(up)	Scrolls up one increment (if implemented by subclass).
Increment(down)	Scrolls down one increment (if implemented by subclass).
Locator()	Whenever cursor moves inside widget window, calls the list of procedures specified by XmNlocatorCallback.
MotifDragStart())	Initiates Motif drag operation and calls XmNdragDropCallback. Disabled if resource XmNallowDrag is False.
Page(left)	Scrolls left one page (if implemented by subclass).
Page(right)	Scrolls right one page (if implemented by subclass).
Page(up)	Scrolls up one page (if implemented by subclass).
Page(down)	Scrolls down one page (if implemented by subclass).
SelectionCallback()	When button pressed inside widget window, calls list of procedures specified by resource **XmNselectionCallback**.
InitAreaSelection(callback)	Initiates selection of rectangular area in widget window. Callback XmNareaSelectionCallback called by action EndAreaSelection when the selection is terminated.
InitAreaSelection(single)	Initiates selection of rectangular area within which all graphic objects included will be selected. Previously selected objects are deselected first.
InitAreaSelection(extend)	Initiates selection of rectangular area within which all graphic objects included will be selected. The new selection will extend the current selection.

Action Name (continued)	**Description**
ExtendAreaSelection()	Draws a rectangle bounding the area being selected.
EndAreaSelection()	Terminates the area selection operation. Calls XmNareaSelectionCallback or selects the Graphic objects contained in the selection depending on the argument of the action InitAreaSelection.
ObjectSelect(single)	Selects a Graphic object. Previously selected objects are deselected first.
ObjectSelect(extend)	Selects a Graphic object and adds it to the list of selected objects.
ObjectEditStart()	Initiates an editing operation on the selected Graphic object. The type of editing operation such as Move, Size, Shape, etc. is defined by resource **XmNobjectEditMode**. Also used internally by widget for creating an object interactively.
ObjectEditStart(move)	Initiates Move operation on selected Graphic object.
ObjectEditStart(shape)	Initiates a Shape operation. Shape allows the user to move the points of MultiPoint based object. It is equivalent to a Size for all other objects.
ObjectEditStart(size)	Initiates a Size operation on the selected Graphic object.
ObjectEditStart(adjust)	Initiates an Adjust operation on the selected object. Adjust is a combination of Shape when the selection is close to a handle bar and Move otherwise.
ObjectEditStart(rubberband)	Initiates rubberband operation by calling callback XmNrubberbandCallback continuously as pointer moves. Actual rubberband shape must be drawn by application.
ObjectEdit()	Continues the editing operation initiated by action ObjectEditStart.
ObjectEditEnd()	Terminates editing operation initiated by action ObjectEditStart and calls callback XmNverifyCallback (callback defined in Graphic class). When XmNobjectEditMode set to XintEDIT_INSERT and object being edited is not a MultiPoint object, terminates the insertion operation and calls XmNinsertObjectCallback.
ObjectEditEnd(m)	Terminates the editing operation initiated by action ObjectEditStart and calls callback XmNverifyCallback (callback is defined in Graphic class). When XmNobjectEditMode is set to XintEDIT_INSERT and the object being edited is a MultiPoint object, terminates the insertion operation initiated by action ObjectEditStart and calls callback XmNinsertObjectCallback.

Action Name (continued)	**Description**
ObjectPointAdd(b)	Adds a point to a MultiPoint object. The point is inserted at the beginning of the list.
ObjectPointAdd(e)	Adds a point to a MultiPoint object. The point is inserted at the end of the list.
ObjectPointAdd(x)	Adds a point to a MultiPoint object. The point is inserted according to its horizontal coordinate.
ObjectPointAdd(y)	Adds a point to a MultiPoint object. The point is inserted according to its vertical coordinate.
ObjectPointDelete()	Deletes a point from a MultiPoint object.
ObjectCancel()	Cancels an operation on an object.
ResourceDialog()	Pops up a resource editor for the selected object (if the class defines one) or let the application provide its own. This action will only activate itself on a double click.
Transform3DStart(scale)	Initiates the scaling of a 3D object.
Transform3DStart(shift)	Initiates the translation of a 3D object.
Transform3DStart(rotate)	Initiates the rotation of a 3D object.
Transform3D()	Continues the 3D operation initiated by Transform3D.
Transform3DEnd()	Terminates the 3D operation initiated by Transform3D.

EditObject Translations

The following translation table is used by an EditObject widget. These default translations can be overridden by the end user or application programmer.

Event Sequence	Actions Invoked
<EnterWindow>	ManagerEnter() InitDrawCursor()
<LeaveWindow>	ManagerFocus() EndDrawCursor()
<FocusIn>	ManagerFocusIn()
<FocusOut>	ManagerFocusOut()
Ctrl <Key>k	ChangeCursorMask()
!Shift <Key> Tab	PreviousTabGroup()
None <Key> Tab	NextTabGroup()
!Shift <Btn1Down>	TraverseCurrent() SelectionCallback() InitAreaSelection(callback) ObjectEditEnd() Locator()
!Ctrl <Btn1Down>	TraverseCurrent() SelectionCallback() InitAreaSelection(extend) ObjectSelect(extend) Locator()
None <Btn1Down>	TraverseCurrent() ObjectSelect(single) ObjectEditStart() Locator() SelectionCallback() InitAreaSelection(single) ResourceDialog()
<Btn2Down>	ObjectEditEnd(m) SelectionCallback() MotifDragStart()
<Btn1Up>	EndAreaSelection() ObjectEditEnd()
None <Btn3Down>	SelectionCallback() Transform3DStart(rotate)
Ctrl <Btn3Down>	Transform3DStart(scale)
Shift <Btn3Down>	Transform3DStart(shift)
<Btn3Motion>	Transform3D()
None <Btn3Down>	SelectionCallback() Transform3DStart(rotate)
<PtrMoved>	DrawCursor() ExtendAreaSelection() ObjectEdit() Locator()

EditObject Callbacks

The following callbacks are defined by the EditObject widget class.

Name	Structure	Reason
XmNareaSelectionCallback	XintEditObjectAreaSelectionCallbackStruct	XintCR_AREA_SELECTION
XmNselectionCallback	XintEditObjectCallbackStruct	XintCR_SELECTION
XmNcopyCallback	XintEditObjectEditCallbackStruct	XintCR_COPY
XmNcutCallback	XintEditObjectEditCallbackStruct	XintCR_CUT
XmNdragDropCallback	XintEditObjectDragDropCallbackStruct	XintCR_DRAG XintCR_DROP
XmNeditObjectCallback	XintEditObjectCallbackStruct	XintCR_OBJECT_EDIT_START XintCR_OBJECT_EDIT XintCR_OBJECT_EDIT_END
XmNinsertObjectCallback	XintEditObjectInsertCallbackStruct	XintCR_INSERT_OBJECT
XmNlocatorCallback	XintEditObjectLocatorCallbackStruct	XintCR_LOCATOR
XmNobjectDeselectionCallback	XintEditObjectSelectionCallbackStruct	XintCR_OBJECT_DESELECTION
XmNobjectSelectioCallback	XintEditObjectSelectionCallbackStruct	XintCR_OBJECT_SELECTION
XmNpasteCallback	XintEditObjectEditCallbackStruct	XintCR_PASTE
XmNresourceDialogCallback	XintEditObjectResourceDialogCallbackStruct	XintCR_MANAGE_RESOURCE_ DIALOG
XmNrubberbandCallback	XintEditObjectRubberbandCallbackStruct	XintCR_RUBBERBAND_START XintCR_RUBBERBAND XintCR_RUBBERBAND_END

XintEditObjectAreaSelectionCallbackStruct

The following ordered table lists the members of the callback structure, XintEditObjectAreaSelectionCallbackStruct, returned to each procedure in the callback list specified by the resource **XmNareaSelectionCallback**.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
int	x	X pixel coordinate of the upper left corner of the selected rectangle.
int	y	Y pixel coordinate of the upper left corner of the selected rectangle.
int	width	Width in pixels of the selected rectangle.
int	height	Height in pixels of the selected rectangle.

Each subclass of the EditObject widget class defines its own callback structure for the callback list specified as the value of the **XmNareaSelectionCallback** resource. Only instances of the EditObject widget class use the area selection callback structure described above.

XintEditObjectCallbackStruct

The following ordered table lists the members of the callback structure, XintEditObjectCallbackStruct, returned to each procedure in the callback list specified by the resources **XmNselectionCallback** and **XmNeditObjectCallback**.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
float	user_x	X user coordinate of the cursor location.
float	user_y	Y user coordinate of the cursor location.
int	pixel_x	X pixel coordinate of the cursor location.
int	pixel_y	Y pixel location of the cursor location.
Object	object	ID of object being edited (**XmNeditObjectCallback** only).

XintEditObjectDragDropCallbackStruct

The following ordered table lists the members of the callback structure, XintEditObjectDragDropCallbackStruct, returned to each procedure in the callback list specified by the resource **XmNdragDropCallback**.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
Object	object	Graphic object being dragged or dropped to. This field is NULL if drag or drop is not from or to a graphic object.
int	operation	This field is 0 on a drag. On a drop, it can be set to XintDROP_COPY, XintDROP_MOVE or XintDROP_LINK. You can modify this field on a drop to change the operation.
Atom *	atoms	Array of source or destination atoms supported.
int	atom_count	Size of array *atoms*.
int	x,y	Location of the pointer when drag/drop started.
Boolean	doit	Set to False to cancel the drag or drop operation.

XintEditObjectEditCallbackStruct

The following ordered table lists the members of the callback structure, XintEditObjectEditCallbackStruct, returned to each procedure in the callback list specified by resources **XmNcopyCallback**, **XmNcutCallback** and **XmNpasteCallback**.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
Object	list	List of objects to be edited.
int	count	Number of objects to be edited.

XintEditObjectSelectionCallbackStruct

The following ordered table lists the members of the callback structure, XintEditObjectSelectionCallbackStruct, returned to each procedure in the callback list specified by resources **XmNobjectSelectionCallback** and **XmNobjectDeselectionCallback**.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
Object	object	The ID of the object selected or deselected. If multiple objects have been selected/deselected, it contains the ID of the first object in the list. See *select_list* to access all the selected/deselected objects.
Object *	select_list	Points to the list of objects selected/deselected in this operation.
int	select_count	The number of selected/deselected objects.

XintEditObjectInsertCallbackStruct

The following ordered table lists the members of the callback structure, XintEditObjectInsertCallbackStruct, returned to each procedure in the callback list specified by resource **XmNinsertObjectCallback**.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
Object	object	The ID of the new object.
Boolean	doit	Set to False if you don’t want the object to be created.

XintEditObjectLocatorCallbackStruct

The following ordered table lists the members of the callback structure, XintEditObjectLocatorCallbackStruct, returned to each procedure in the callback list specified by resource **XmNlocatorCallback**.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback. Contains the window coordinates of the cursor.
int	pixel_x	The X location of the cursor in the window coordinate system.
int	pixel_y	The Y location of the cursor in the window coordinate system.
float	user_x	The X location of the cursor hot spot in the user coordinate system.
float	user_y	The Y location of the cursor hot spot in the user coordinate system.

Some subclasses of the EditObject widget class define their own callback structures for the callback list specified as the value of the **XmNlocatorCallback** resource. Instances of the EditObject widget class use the locator callback structure described above.

XintEditObjectResourceDialogCallbackStruct

The following ordered table lists the members of the callback structure, XintEditObjectResourceDialogCallbackStruct, returned to each procedure in the callback list specified by resource **XmNresourceDialogCallback**.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback. Contains the window coordinates of the cursor.
Object	object	ID of the selected object.
Boolean	doit	Set to False to prevent the built-in resource editor from being activated.

XintEditObjectRubberbandCallbackStruct

The following ordered table lists the members of the callback structure, XintEditObjectRubberbandCallbackStruct, returned to each procedure in the callback list specified by resource **XmNrubberbandCallback**.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
int	start_x	The X location of the cursor when the rubberband operation started.
int	start_y	The Y location of the cursor when the rubberband operation started.
int	x_offset	The offset between the current location of the pointer and the original X location.
int	y_offset	The offset between the current location of the pointer and the original Y location.

EditObject Functions

The following functions are defined for creating and manipulating an EditObject widget.

Function	Description
XintCreateEditObject	Creates an EditObject widget.
XintDrawCursorFromData	Causes cross hair cursor to be drawn at specified location in EditObject’s window.
XintEditObjectBack	Moves current object behind other objects in widget.
XintEditObjectCopy	Copies the selected objects into the clipboard.
XintEditObjectCurrent	Returns the last object selected.
XintEditObjectCut	Copies all selected objects into clipboard and destroys them.
XintEditObjectDeselectAll	Deselects all currently selected objects.
XintEditObjectDeselectObject	Removes the specified object from the selected list.
XintEditObjectDestroyObject	Fast destroy function for objects.
XintEditObjectFreeze	Controls the update of an EditObject display.

Function (continued)	**Description**
XintEditObjectFront	Moves current object to front of other objects in widget.
XintEditObjectGetIntersectList	Returns a list containing all of the objects that are children of the EditObject widget and intersect a specified rectangle.
XintEditObjectGetList	Returns a list containing all of the objects that are children of the specified EditObject widget.
XintEditObjectGroup	Groups the selected objects.
XintEditObjectInsert	Creates and inserts an object interactively.
XintEditObjectLower	Moves current object one place down in stacking order.
XintEditObjectManageResource-Dialog	Manages resource editor panel of specific object, if one is available.
XintEditObjectMove	Allows interactive movement of the selected object.
XintEditObjectNew	Destroys all objects belonging the EditObject widget.
XintEditObjectOpen	Manages a dialog that allows the loading of an ASCII object description file.
XintEditObjectPaste	Pastes objects in clipboard into EditObject widget.
XintEditObjectRaise	Moves current object one place up in stacking order.
XintEditObjectReadFile	Reads an ASCII file containing a description of objects and places them into the EditObject widget.
XintEditObjectSave	Saves all objects of EditObject widget into ASCII file.
XintEditObjectSaveAs	Manages a dialog box that prompts for the name of a file to store an ASCII description of the objects of an EditObject widget.
XintEditObjectSelectAll	Selects all Graphic objects of EditObject widget.
XintEditObjectSelectList	Returns list of the selected objects.
XintEditObjectSelectObject	Adds an object to the list of selected objects.
XintEditObjectSetEditMode	Sets the value of resource **XmNobjectEditMode**.
XintEditObjectSize	Allows interactive sizing of currently selected object.
XintEditObjectUngroup	Ungroups the currently selected group object.
XintEditObjectWriteFile	Saves all objects of EditObject widget into ASCII file.

XintCreateEditObject

Creates an unmanaged EditObject widget.

```
Widget XintCreateEditObject (...)
```

Widget	parent	Parent of new EditObject widget.
char *	name	Name of new EditObject widget.
ArgList	arglist	List of resource/value items.
Cardinal	argcount	Number of items in arglist.

XintDrawCursorFromData

Causes the cross hair cursor to be drawn at a specified location in an EditObject widget. This function has an effect only when the value of resource **XmNcursorType** is XintCROSS_HAIR_CURSOR.

```
void XintDrawCursorFromData (...)
```

Widget	widget	EditObject widget ID
float	user_x	The horizontal location of where the cursor is to be drawn.
float	user_y	The vertical location of where the cursor is to be drawn.

XintEditObjectBack

Changes the stacking order of a widget so that the specified object becomes last in the display list. If argument object is NULL, the function will be applied to the currently selected object.

```
void XintEditObjectBack (Widget widget, Object object)
```

widget	ID of an EditObject widget
object	ID of the object to move to the back.

XintEditObjectCopy

Places all the selected Graphic objects of an EditObject widget into the clipboard. Objects on the clipboard can be pasted back into any EditObject widget using function XintEditObjectPaste.

```
void XintEditObjectCopy (Widget widget)
```

widget	ID of an EditObject widget.

XintEditObjectCurrent

Returns the currently selected object of an EditObject widget.

```
Object XintEditObjectCurrent (Widget widget)
```

widget	The ID of an EditObject widget.

XintEditObjectCut

Places the selected objects into the clipboard and then destroys them. Objects on the clipboard may be pasted into any EditObject widget using function XintEditObjectPaste.

```
void   XintEditObjectCut (Widget widget)
```

widget	The ID of an EditObject widget.

XintEditObjectDeselectAll

Deselects all the selected objects of an EditObject widget.

```
void XintEditObjectDeselectAll (Widget widget)
```

widget	The ID of an EditObject widget.

XintEditObjectDeselectObject

Allows the application programmer to remove a Graphic object from the list of selected objects.

```
void XintEditObjectDeselectObject (Widget widget,
                                Object object)
```

widget	The ID of an EditObject widget
object	The ID of the object to remove from the selected list.

XintEditObjectDestroyObject

Destroys an object, and is identical functionally to XtDestroyWidget except that it is faster.

```
void XintEditObjectDestroyObject (Object object)
```

object	The ID of the object to destroy.

XintEditObjectFreeze

Controls the update of an EditObject display. When this function is called with argument *state* set to True, the display will not be updated until the function is called again with *state* set to False. Typically used when changes are made to multiple objects to minimize flashing on the screen.

```
void XintEditObjectFreeze (...)
```

Widget	widget	EditObject widget ID.
Boolean	state	True to freeze, False to update the display.

XintEditObjectFront

Changes the stacking order of a widget so that the specified object becomes first in the display list. If argument object is NULL, the function will be applied to the currently selected object.

```
void XintEditObjectFront (Widget widget, Object object)
```

widget	The ID of an EditObject widget
object	The ID of the object to move to the front.

XintEditObjectGetIntersectList

Returns a list of all of the objects within the specified rectangle that are children of an EditObject widget. This includes objects that are only partially inside the defined area.

```
Object* XintEditObjectGetIntersectList (...)
```

Widget	edit_object	EditObject widget ID.
int	x	X value of upper left corner of intersection rectangle, in pixels.
int	y	Y value of upper left corner of intersection rectangle, in pixels.
int	width	Width of intersection rectangle, in pixels.
int	height	Height of intersection rectangle, in pixels.
int *	count	Returned count of objects in list.

If the specified EditObject widget has no Graphic objects as children or if the intersection of the specified rectangle and the EditObject widget is empty, then NULL is returned. The list returned must be freed by the application when it has finished with it.

XintEditObjectGetList

Returns a list of all of the objects that are children of an EditObject widget.

```
Object* XintEditObjectGetList (...)
```

Widget	widget	EditObject widget ID.
int *	count	Returned count of objects in the list.

If the specified EditObject widget has no Graphic objects as children, then NULL is returned. The list returned must be freed by the application when it has finished with it.

XintEditObjectGroup

Groups selected objects into a Group object. Attributes set on a group propagate to the group children. Groups can be nested without limit.

```
Object XintEditObjectGroup (Widget widget)
```

where *widget* is the ID of an EditObject widget. The function returns the ID of the new Group object.

XintEditObjectInsert

Allows the interactive insertion of a Graphic object into a widget whose class is based on EditObject. This function sets the resource **XmNobjectEditMode** to XintEDIT_INSERT and uses actions ObjectEditStart, ObjectEdit and ObjectEditEnd. When using the default translation table, an object is inserted interactively using BSelect Click and BSelect Drag. MultiPoint based objects are inserted using BSelect Click and BTransfer Click for the last point. Once the object is inserted, the original value of resource **XmNobjectEditMode** is restored.

```
void XintEditObjectInsert (...)
```

Widget	widget	Specifies the ID of the EditObject widget.
ObjectClass	class	Specifies the class of the Graphic object to be created.
ArgList	arglist	List of resources to be applied to Graphic object created.
Cardinal	argcount	Number of items in arglist.

Note: Do not specify resources in *arglist* that have to do with the location and size of the object to be created since those will be set by the end user. The creation of a Graphic object does not occur until the user specifies the object interactively. Resources specified using an address must remain allocated until the object is created. For example, all float values that are specified should be declared static. For the same reason, the ID of the new object is not returned by function XintEditObjectInsert. Callback XmNinsertObjectCallback will return the ID of the object when it is created.

XintEditObjectLower

Changes the display order of the specified object by moving it behind the object that it was immediately in front of. If object is NULL, the function will be applied to the currently selected object.

```
void XintEditObjectLower (...)
```

Widget	widget	EditObject widget ID.
Object	object	ID of the object to lower.

XintEditObjectManageResourceDialog

Manages the resource editor panel of the specific object if there is one available. Examples of objects that have a built-in resource editor are: Text, Chart, AxisObject, Symbol.

```
void XintEditObjectManageResourceDialog (...)
```

Widget	widget	EditObject widget ID.
Object	object	ID of the object for which to manage the dialog panel.

XintEditObjectMove

Allows the end-user to move a selected object from a Move menu item. When this function is called, the pointer is warped to the center of the selected object. As the end-user performs Drag, the object outline moves along with the pointer. A BSelect will place the object at the current location and terminate the move operation.

```
void XintEditObjectMove (Widget widget)
```

where *widget* is the ID of an EditObject widget.

XintEditObjectNew

Destroys all Graphic objects belonging to an EditObject widget.

```
    void XintEditObjectNew (Widget widget)
```

widget	ID of an EditObject widget.

XintEditObjectOpen

Manages a dialog box that allows the selection of a object description file. After the file is selected, the objects will be created inside the EditObject widget specified as argument.

```
    void XintEditObjectOpen (Widget widget)
```

widget	ID of an EditObject widget.

XintEditObjectPaste

Pastes all Graphic objects saved in clipboard into specified EditObject widget.

```
    void XintEditObjectPaste (Widget widget)
```

widget	ID of an EditObject widget.

XintEditObjectRaise

Changes the stacking order of the specified object by moving it one place up in the display list. If argument object is set to NULL, the function will be applied to the currently selected object.

```
    void XintEditObjectRaise (...)
```

Widget	widget	EditObject widget ID.
Object	object	ID of the object to raise.

XintEditObjectReadFile

Reads an object description file and creates the objects in the specified EditObject widget. Depending on which include file is added, this function will be redefined to use a file loader that is “aware” of the type of file it needs to load. For instance, if `<Xint/Chart.h>` is included the function is redefined to use a “Chart aware” file loader.

```
    void XintEditObjectReadFile (...)
```

Widget	widget	EditObject widget ID.
char *	filename	Name of the file containing the object description.

Returns False if it cannot open *filename* or if *filename* does not contain a valid object description.

XintEditObjectSave

Saves the Graphic objects contained in the specified EditObject widget into a file that was previously specified in XintEditObjectOpen. Use function XintEditObjectSaveAs or function XintEditObjectWriteFile if you want to specify a different filename.

```
void XintEditObjectSave (Widget widget)
```

widget	ID of an EditObject widget.

XintEditObjectSaveAs

Manages a dialog box that prompts for the name of a file where the Graphic objects contained in the specified EditObject widget are saved.

```
void XintEditObjectSaveAs (Widget widget)
```

widget	ID of an EditObject widget.

XintEditObjectSelectAll

Selects all the objects defined in the specified EditObject widget.

```
void XintEditObjectSelectAll (Widget widget)
```

widget	ID of an EditObject widget.

XintEditObjectSelectList

Returns the list of selected Graphic objects in the specified EditObject widget.

```
Object * XintEditObjectSelectList (...)
```

Widget	widget	EditObject widget ID.
int *	count	Number of objects returned in the list.

The application should free the returned list using function XtFree, after it is no longer needed, if the number of selected objects was not zero.

XintEditObjectSelectObject

Adds specified object to the list of selected objects of an EditObject widget.

```
void XintEditObjectSelectObject (...)
```

Widget	widget	EditObject widget ID.
Object	object	ID of the object to select.

XintEditObjectSetEditMode

Sets the value of resource **XmNobjectEditMode**.

```
void XintEditObjectSetEditMode (...)
```

Widget	widget	EditObject widget ID.
int	edit_mode	New value of resource **XmNobjectEditMode**.

XintEditObjectSize

Allows the end user to size a selected object from a Size menu item. When this function is called, the pointer is warped to the center of the selected object. As the end user moves the pointer, the new shape of the object is outlined. A BSelect Drag will size the object as specified and a BSelect Up will terminate the operation.

```
void XintEditObjectSize (Widget widget)
```

widget	ID of an EditObject widget.

XintEditObjectUngroup

Ungroups currently selected group object in the specified EditObject widget.

```
void XintEditObjectUngroup (Widget widget)
```

widget	ID of an EditObject widget.

XintEditObjectWriteFile

Saves the Graphic object belonging to the specified EditObject widget into a file. The object description file can later be read back using macro XintEditObjectReadFile.

```
Boolean XintEditObjectWriteFile (...)
```

Widget	widget	The ID of the EditObject widget.
char *	filename	The name of the file where the Graphic object description is saved.
char *	mode	The fopen style mode indicating how to open the file.

Returns False if it failed to open the specified file.

Macros

Macro XintEditObjectReadFile reads an object description file and creates the objects in the specified EditObject widget. Depending on which include file is added, this macro will be redefined to use a file loader that is “aware” of the type of file it needs to load. For instance, if `<Xint/Chart.h>` is included the macro is redefined to use a “Chart aware” file loader.

```
Boolean XintEditObjectReadFile (...)
```

Widget	widget	The ID of the EditObject widget.
char *	filename	The name of the file containing the object description.

This function returns False if it cannot open *filename* or if *filename* does not contain a valid object description.

ObjectEditor Widget Class

ObjectEditor is a widget class that can be used to build panels designed to create and edit Graphic objects interactively. This widget can be customized so that the application can select which objects can be created and which menus are available. The application has also control over the layout of the different sub-menus that make up the widget.

ObjectEditor is provided for convenience only. It is highly configurable and should be flexible enough to be used by most applications. However, if this is not the case, all of its behavior can be easily duplicated using the EditObject class convenience functions and Graphic object resources.

Object Editor Layout

The ObjectEditor widget is built upon several groups of components, including a group of push buttons used to perform actions on the objects, a group of pulldown menus used to edit the attributes of the selected objects, a group of push buttons to create objects interactively, a group of pulldown menus to edit the object colors and a group of push buttons to set a bitmap pattern on the selected objects.

Each group is created inside a Motif RowColumn widget. Resources are available to specify the content of each group and to position the elements of the group, rowwise or columnwise. Also, using resource **XmNeditList**, the application can control which groups of components are displayed.

ObjectEditor Widget Appearance

The ObjectEditor widget appears as rectangular window containing a number of push buttons, option menus and bitmap selectors. Figure 25 shows an example of an ObjectEditor widget configured as a dialog box:

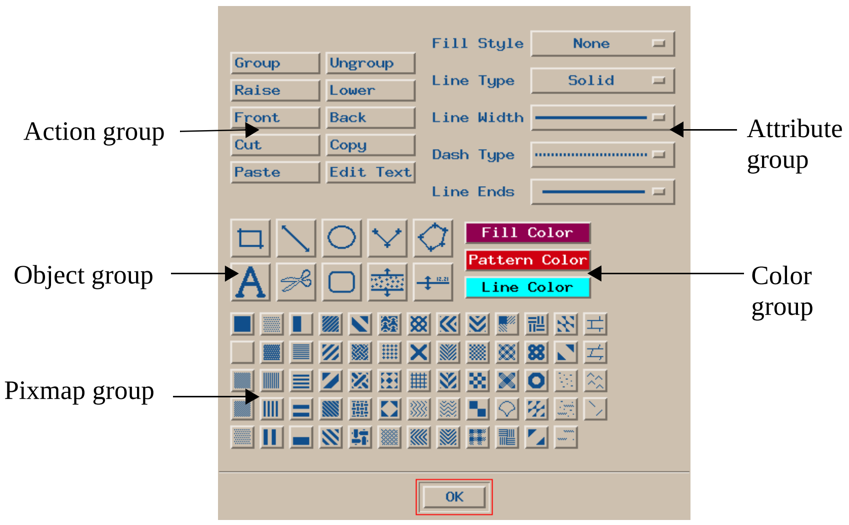


Figure 25. ObjectEditor Widget Configured as a Dialog Box

Inherited Behavior and Resources

The ObjectEditor widget inherits behavior and resources from the *Core*, *Composite*, *Constraint* and *Motif Manager* classes:

- Class pointer is *xintObjectEditorWidgetClass*
- Class name is *XintObjectEditor*
- Header file is included as `<Xint/ObjectEditor.h>`

ObjectEditor Resources

The following resources are defined by the ObjectEditor widget class.

Name	Type Default	Access
XmNactionList	String * dynamic	CSG
XmNactionNumColumns	short 5	CSG
XmNactionOrientation	unsigned char XintHORIZONTAL	CSG
XmNattributeList	String * dynamic	CSG
XmNattributeNumColumns	short 1	CSG
XmNattributeOrientation	unsigned char XintVERTICAL	CSG
XmNcolorAttributeList	String * dynamic	CSG
XmNcolorList	String * dynamic	CSG
XmNcolorNumColumns	short 1	CSG
XmNcolorOrientation	unsigned char XintVERTICAL	CSG
XmNeditList	String * dynamic	CSG
XmNeditObjects	Widget * NULL	CSG
XmNnumColumns	short 2	CSG
XmNnumEditObjects	int 0	CSG
XmNobjectClassList	ObjectClass * dynamic	CSG
XmNorientation	unsigned char XintVERTICAL	CSG

Name (continued)	**Type** **Default**	**Access**
XmNpixmapList	Pixmap * NULL	CSG
XmNpixmapNumColumns	short dynamic	CSG
XmNpixmapOrientation	unsigned char XintVERTICAL	CSG
XmNshowAttributeLabels	Boolean True	CSG

XmNactionList

Specifies a NULL terminated list of strings containing the name of the actions that can be performed on objects. For each action that you specify, a push button will be created inside the Action group. Specify any or all of the actions listed below:

XintGROUP	Creates a Push Button that groups all the selected objects.
XintUNGROUP	Creates a Push Button that ungroups all the selected groups.
XintRAISE	Creates a Push Button that raises one place the selected objects in the display list.
XintLOWER	Creates a Push Button that lowers one place the selected objects in the display list.
XintFRONT	Creates a Push Button that raises the selected objects to the front of the display list.
XintBACK	Creates a Push Button that lowers the selected objects at the end of the display list.
XintCUT	Creates a Push Button that removes the selected objects and places them in the cut and paste buffer.
XintCOPY	Creates a Push Button that places the selected objects in the cut and paste buffer.
XintPASTE	Creates a Push Button that pastes any object in the cut and paste buffer.

Default is a NULL terminated list containing all the actions listed above.

XmNactionNumColumns
XmNactionOrientation

Specifies orientation (XintVERTICAL or XintHORIZONTAL) and number of columns (or rows if directions is set to XintHORIZONTAL) in the row column widget created to display the action push buttons (Action group).

XmNattributeList

Specifies a NULL terminated list of strings containing the name of the attributes that can be edited. A pulldown menu, placed inside the attribute group, will be created to edit each of the attributes specified in the list. Specify any or all of the attributes listed below:

XintFILL_STYLE	Creates pulldown menu to edit fill style of selected objects.
XintLINE_STYLE	Creates pulldown menu to edit line style of selected objects.
XintLINE_WIDTH	Creates pulldown menu to edit line width of selected objects.
XintDASH_TYPE	Creates pulldown menu to edit line dash style of selected objects.
XintLINE_ENDS	Creates pulldown menu to edit line end style (for arrows) of the selected line objects.

Default is a NULL terminated list containing all the listed attributes.

XmNattributeNumColumns
XmNattributeOrientation

Specifies orientation (XintVERTICAL or XintHORIZONTAL) and the number of columns (or rows if directions is set to XintHORIZONTAL) in the row column widget that is created to display the attribute pulldown menus (Attribute group).

XmNcolorAttributeList

Specifies a NULL terminated list of strings containing a list of color attributes that can be edited. A pulldown menu, placed inside the color group, will be created to edit each attributes specified in list. Specify any or all of the attributes listed below:

XintFILL_COLOR	Creates pulldown menu to edit fill color of selected objects.
XintPATTERN_COLOR	Creates pulldown menu to edit pattern color of selected objects.
XintLINE_COLOR	Creates pulldown menu to edit line color of selected objects.

Default is a NULL terminated list containing all the listed color attributes.

XmNcolorList

Specifies a NULL terminated list of color names that will be used in the color editor pulldown menus.

XmNcolorNumColumns
XmNcolorOrientation

Specifies the orientation (XintVERTICAL or XintHORIZONTAL) and the number of columns (or rows if directions is set to XintHORIZONTAL) in the row column widget that is created to display the pulldown menu used to edit the color attributes listed in resource **XmNcolorAttributeList** (Color group).

XmNeditList

Specifies a NULL terminated list of strings containing the names of the group of editors that are created inside the ObjectEditor widget. Specify any or all of the names listed below:

XintACTION_EDITOR	Creates Action group that contains set of push buttons used to perform actions on the objects.
XintOBJECT_EDITOR	Creates Object group that contains set of push buttons used to create objects interactively.
XintCOLOR_EDITOR	Creates Color group that contains set of pulldown menus used to edit the object colors.
XintATTRIBUTE_EDITOR	Creates Attribute group that contains set of pulldown menus used to edit the object attributes.
XintPIXMAP_EDITOR	Creates Pixmap group that contains set of push buttons used to assign a fill pattern to the objects.

Default is a NULL terminated list containing all the items listed above.

XmNeditObjects

Specifies a list of EditObject widgets which can be edited using the ObjectEditor widget. The size of this list is specified using resource **XmNnumEditObjects**. See also convenience function XintObjectEditorAddEditObjectToList which can be used to add widgets to this list.

XmNnumColumns

Specifies the number of columns (or rows if resource **XmNorientation** is set to XintHORIZONTAL) in the RowColumn widget used to layout the different sets of editors that compose the ObjectEditor widget.

XmNnumEditObjects

Specifies the size of the widget list specified with resource **XmNeditObjects**.

XmNobjectClassList

Specifies a NULL terminated list of object class names for which objects can be created interactively. For each class specified, a push button displaying a graphic representation of the object will be created inside the Object group. You can specify a list containing any or all of the classes listed below:

xintChartObjectClass	To create a Chart object interactively.
xintFreeHandObjectClass	To create a Polyline for which points are sampled along the cursor path during creation.
xintLineObjectClass	To create a Line object interactively.
xintOvalObjectClass	To create an Oval object interactively.
xintPolygonObjectClass	To create a closed Polyline object interactively.
xintPolylineObjectClass	To create a Polyline object interactively.
xintRectangleObjectClass	To create a Rectangle object interactively.
xintRoundedRectObjectClass	To create a Rounded Rectangle object interactively.
xintScaledTextObjectClass	To create a scalable Text object interactively.
xintSymbolObjectClass	To create a Symbol object interactively.
xintTextObjectClass	To create a fixed Text object interactively.

Refer to *“Graphic Object Library Components”* on page 4 for more information on each object class. The set of push buttons listed in the previous table will only be created if attribute XintOBJECT is listed in resource **XmNeditList**.

XmNorientation

Specifies the orientation (XintVERTICAL or XintHORIZONTAL) of the RowColumn widget used to lay out the different set of editors that compose the ObjectEditor widget. The number of columns or rows is specified using resource **XmNnumColumns**.

XmNpixmapList

Specifies a list of pixmaps of depth 1 used to create a set of push buttons that can be used to edit an object bitmap pattern. The list must be terminated with constant XmUNSPECIFIED_PIXMAP. The ObjectEditor has a list of predefined patterns of size 32x32 which can be accessed using function XintObjectEditorGetDefinedPixmap defined in the Defined Functions section below.

XmNpixmapNumColumns
XmNpixmapOrientation

Specifies the number of columns and the orientation of the RowColumn widget that contains the array of push buttons that are used for pattern selection (Pixmap group). This set of buttons will only be created if attribute XintPIXMAPS is listed in resource **XmNeditList**.

XmNshowAttributeLabels

Specifies whether a label describing the attribute is displayed in front of the option menus used to edit the object attributes (see resource **XmNattributeList**).

ObjectEditor Functions

The following functions are defined by the ObjectEditor widget class.

Function Name	Description
XintCreateObjectEditor	Creates an ObjectEditor widget.
XintObjectEditorGetDefinedPixmap	Returns a pixmap id from a specified name.
XintObjectEditorAddEditObjectToList	Adds a widget to the list of EditObject widgets associated with an ObjectEditor widget.
XintObjectEditorRemoveEditObject From-List	Removes a widget from the list of EditObject widgets managed by an ObjectEditor widget.

XintCreateObjectEditor

Creates an unmanaged ObjectEditor widget.

```
Widget XintCreateObjectEditor (...)
```

Widget	parent	Parent of new ObjectEditor widget.
char *	name	Name of new ObjectEditor widget.
ArgList	arglist	List of resource/value items.
Cardinal	argcount	Number of items in arglist.

XintObjectEditorGetDefinedPixmap

Returns the ID of a pixmap of depth 1 from the specified name.

```
Pixmap XintObjectEditorGetDefinedPixmap (...)
```

Widget	widget	ID of any Motif or INT widget.
char *	name	Name of the pixmap. Use one of the name in the list below.

Pixmaps names

The following table lists the pixmap names recognized by function XintObjectEditorGetDefinedPixmap:

“Solid”	“Clear”	“25_percent_1”	“25_percent_2”
“25_percent_3”	“25_percent_4”	“50_percent_1”	“50_percent_2”
“75_percent”	“Vertical”	“Vertical1”	“Vertical2”
“Vertical3”	“Horizontal”	“Horizontal1”	“Horizontal2”
“Horizontal3”	“Slant_Left”	“Slant_Left1”	“Slant_Left2”
“Star_Left1”	“Star_Left2”	“Star_Right1”	“Star_Right2”
“Slant_Right”	“Slant_Right1”	“Slant_Right2”	“Zigzag”
“Zigzag1”	“Zigzag2”	“Tread”	“Tread1”
“Tread2”	“Trellis”	“Trellis1”	“Trellis2”
“Weave1”	“Weave2”	“Weave3”	“Weave4”
“12_5_percent_1”	“12_5_percent_2”	“12_5_percent_3”	“12_5_percent_4”
“12_5_percent_5”	“12_5_percent_6”	“12_5_percent_7”	“12_5_percent_8”
“Sandstone”	“Siltstone”	“Shale”	“Limestone”
“Dolomite”	“Chert”	“Basement”	“Check1”
“Check2”	“Check3”	“Wicker1”	“Wicker2”
“Wicker3”	“Cross_Hatch”	“Diamond1”	“Diamond2”
“Diamond3”	“Lattice1”	“Lattice2”	“Lattice3”
“Lattice4”	“Lattice5”	“Lattice6”	“Lattice7”
“Herring_Bone1”	“Herring_Bone2”	“Plaid1”	“Plaid2”
“Plaid3”	“Plaid4”	“Imbrication1”	“Imbrication2”
“Imbrication3”	“Imbrication4”	“Diaper”	“Parquet1”
“Parquet2”	“Horizontal_Bar1”	“Horizontal_Bar2”	“Horizontal_Bar3”
“Horizontal_Bar4”	“Horizontal_Bar5”	“Horizontal_Bar6”	“Horizontal_Bar7”
“Horizontal_Bar8”	“Horizontal_Bar9”	“Horizontal_Bar10”	“Vertical_Bar1”
“Vertical_Bar2”	“Vertical_Bar3”	“Vertical_Bar4”	“Vertical_Bar5”
“Vertical_Bar6”	“Vertical_Bar7”	“Vertical_Bar8”	“Vertical_Bar9”

Figure 26 illustrates the default patterns, as listed in the table. If you don’t find the right pixmap in the list, you can always create your own using the proper X library functions.

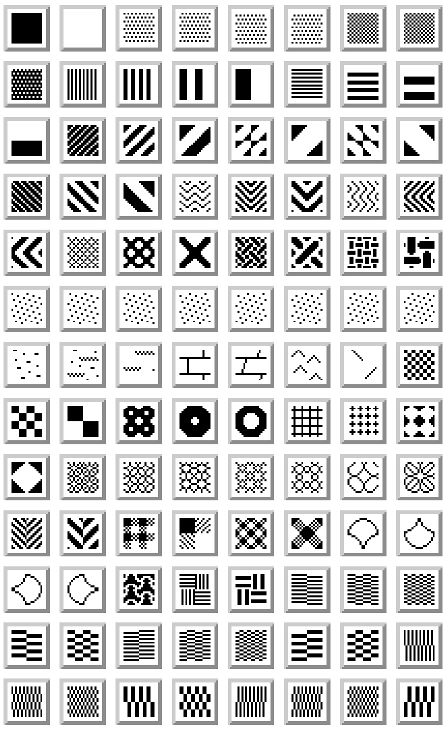


Figure 26. Default Patterns

XintObjectEditorAddEditObjectToList

Adds an EditObject widget to resource list **XmNeditObjects**, which contains the list of widgets managed by the ObjectEditor widget. This function also increments the object count maintained in resource **XmNnumEditObjects**.

```
void XintObjectEditorAddEditObjectToList (...)
```

Widget	widget	ID of the ObjectEditor widget.
Widget	edit_object	ID of an EditObject widget where we want to create or edit objects interactively using the ObjectEditor widget.

XintObjectEditorRemoveEditObjectFromList

A convenience function that removes an EditObject widget from the resource list **XmNeditObjects**, which contains the list of widgets managed by the ObjectEditor widget. This function also decrements the object count maintained in resource **XmNnumEditObjects**.

```
void XintObjectEditorRemoveEditObjectFromList (...)
```

Widget	widget	ID of the ObjectEditor widget.
Widget	eo	ID of the EditObject widget to be removed from the list.

Graphic Object Reference

3

Overview

This chapter includes the following sections:

- **GraphicObject Library** on page 122
- **Graphic Object Metaclass** on page 123
- **Group Object Class** on page 132
- **Image Object Class** on page 135
- **Line Object Class** on page 138
- **MultiPoint Object Metaclass** on page 142
- **Oval Object Class** on page 145
- **Polyline Object Class** on page 146
- **Rectangle Object Class** on page 149
- **Symbol Object Class** on page 152
- **Text Object Class** on page 159

GraphicObject Library

The INT GraphicObject library is a very powerful structured graphic object library. Objects can be selected, moved, resized and grouped together. A full cut and paste functionality is also available, including the ability to cut and paste objects from one application to another. Objects can also be saved and retrieved from a file or a string.

Note: This library provides some of the building blocks for the Chart object and also can be used directly as a stand-alone graphic library or to provide annotation to a Chart display.

Summary of Components

The GraphicObject library defines the following object classes:

Class	Description
Graphic	Base class for all graphic objects. It defines the basic resources and methods used by all other INT Graphic objects.
Group	Groups primitive objects or other groups into a single group object.
Image	Displays an image.
Line	Draws a line between two points, with or without arrows at the end.
MultiPoint	Base class for all objects with multiple points such as a polyline.
Oval	Draws oval shaped objects.
Polyline	Draws polyline or polygon objects.
Rectangle	Draws rectangular objects.
Symbol	Draws a user specified symbol.
Text	Draws text.

Graphic Object Metaclass

The Graphic class defines the basic methods that apply to all INT Graphic objects, including methods to display, select, move and resize objects. Additional methods include hardcopy output for Postscript and CGM, group and ungroup, cut and paste, and file import and export. Most methods defined in the Graphic class are accessible through actions or functions defined in the EditObject widget class. Do not instantiate the Graphic object class directly.

Interactive Editing

Interactive editing of an object is defined by specifying a translation table on the parent widget. The EditObject widget class defines a comprehensive set of actions for object editing. Examples of actions include ObjectSelect, ObjectEdit, ObjectAddPoint and ObjectDeletePoint. The Graphic class defines a set of resources, including **XmNsensitive**, **XmNshape** and **XmNmove** to selectively enable or disable any editing action on a Graphic object.

Visual Attributes

The Graphic class defines all the basic graphic attributes necessary to describe the appearance an object, including line color, size and style. Other resources are available to describe the fill style, fill color, and pattern. The bitmap pattern can be specified from a pixmap or from a file containing a bitmap definition.

Coordinate System

Each object class defines a specific resource or set of resources that describe the geometry of the object. The coordinate system used to describe the object geometry is normally the one defined by its parent. Two resources, **XmNverticalAxis** and **XmNhorizontalAxis**, can be used to specify a different coordinate system vertically, horizontally or both.

Inherited Behavior and Resources

The Graphic object class inherits behavior and resources from the *Xt Object* class:

- Class pointer is *xintGraphicObjectClass*
- Class name is *XintGraphic*
- Header file is included as `<Xint/Graphic.h>`

Resources The following resources are defined by the Graphic object class.

Name	Type Default	Access
XmNclipGrid	Boolean False	CSG
XmNcolor	Pixel foreground	CSG
XmNdashList	char * NULL	CSG
XmNdisplayName	Boolean False	CSG
XmNfillColor	Pixel background	CSG
XmNfillFilename	char * NULL	CSG
XmNfillPixmap	Pixmap NULL	CSG
XmNfillStyle	int XintFILL_NONE	CSG
XmNfont	char * “Helvetica*120*”	CSG
XmNgroup	Object NULL	CSG
XmNhighlightMode	int XintHIGHLIGHT_HANDLE	CSG
XmNhorizontalAxis	Widget NULL	CSG
XmNlineStyle	int XintLINE_SOLID	CSG

Name (continued)	**Type** **Default**	**Access**
XmNlineThickness	int 1	CSG
XmNmove	Boolean True	CSG
XmNmoveDirection	int XintMOVE_ANY	CSG
XmNname	char * NULL	CSG
XmNresourceDialog	Boolean True	CSG
XmNsensitive	Boolean True	CSG
XmNshape	Boolean True	CSG
XmNstippleColor	Pixel foreground	CSG
XmNuserData	XtPointer NULL	CSG
XmNverifyCallback	XtCallbackList NULL	C
XmNverticalAxis	Widget NULL	CSG
XmNvisible	Boolean True	CSG

XmNclipGrid

Specifies whether the object will be clipped to the Grid boundaries if its parent is a Grid widget or a widget which class is derived from Grid.

XmNcolor

Specifies the color (as a pixel value) used to draw the Graphic object edges.

XmNdashList

Specifies a string containing an X Window-style dash pattern. This resource will be used when resource **XmNlineStyle** is set to XintLINE_ON_OFF_DASH or XintLINE_DOUBLE_DASH.

XmNdisplayName

Specifies whether the Graphic object name (from resource **XmNname**) will be drawn.

XmNfillColor

Specifies the pixel color used to draw a filled Graphic object. This resource is used when resource **XmNfillStyle** is set to XintFILL_SOLID or XintFILL_OPAQUE_STIPPLED.

XmNfillFilename

Specifies a filename containing the description of a bitmap that will be used for the fill pattern. This resource is used when the value of resource **XmNfillStyle** is set to XintFILL_STIPPLED or XintFILL_OPAQUE_STIPPLED.

XmNfillPixmap

Specifies a pixmap of depth 1 (bitmap) that will be used for the fill pattern. This resource is used when resource **XmNfillStyle** is set to XintFILL_STIPPLED or XintFILL_OPAQUE_STIPPLED.

XmNfillStyle

Specifies the fill style for the Graphic object. You can specify one of the following constants:

Resource Value	Description
XintFILL_NONE	No fill drawn.
XintFILL_STIPPLED	Fill pattern drawn using stipple color pixel masked by the specified fill pixmap.
XintFILL_OPAQUE_STIPPLED	Fill pattern drawn using fill color pixel for unset bits and the stipple color pixel for set bits of the specified fill pixmap.
XintFILL_SOLID	Fill pattern solid and drawn using the fill color pixel.

XmNfont

Specifies the name of the X-Window font used to draw the object’s label. This resource is used only if resource **XmNdisplayName** is set to True.

XmNgroup

Specifies the ID of a group object in which you want the graphic object to be inserted. You only need to specify this resource if you want this object to be inserted in a group. See Group Object class section for more information on group objects.

XmNhighlightMode

Specifies how an object will be highlighted when selected. The value of this resource is specified as one of the defined constants listed below:

Resource Value	Description
XintHIGHLIGHT_NONE	Object will not be highlighted when selected.
XintHIGHLIGHT_HANDLE (default)	Object's handles will appear when it is selected.
XintHIGHLIGHT_COLOR	Object's color will change to the object's highlight color when it is selected. The object's highlight color is the value of its parent's **XmNhighlightColor** resource.

XmNhorizontalAxis

Specifies the ID of an Axis object which the Graphic object will use for its horizontal coordinate system. If this resource is set to NULL, the horizontal coordinate system of the object's parent will be used.

XmNlineStyle

Specifies the line style used to draw the object's edges. This resource is specified as one of the defined constants listed below:

Resource Value	Description
XintLINE_NONE	The object lines are not drawn.
XintLINE_SOLID	The object lines are drawn using a solid line.
XintLINE_ON_OFF_DASH	The object lines are drawn using a dashed line.
XintLINE_DOUBLE_DASH	The object lines are drawn using a double dashed line.

XmNlineThickness

Specifies the line thickness used to draw the Graphic object's edges.

XmNmove

Specifies whether or not the Graphic object can be moved interactively by the end user.

XmNmoveDirection

Specifies whether a Graphic object move operation is constrained. You can specify one of the following defined constants:

Resource Value	Description
XintMOVE_ANY	The object can be moved freely in any direction.
XintMOVE_VERTICAL	The object can only be moved vertically.
XintMOVE_HORIZONTAL	The object can only be moved horizontally

XmNname

Specifies the name of the Graphic object. The name of the object will be drawn if resource **XmNdisplayLabel** is set to True. If this resource is NULL (default), the object name as specified at creation time will be used.

XmNresourceDialog

Specifies whether or not the built-in resource editor dialog will be enabled. When this entry is set to False, the built-in resource editor cannot be activated.

XmNsensitive

Specifies whether or not a Graphic object can be selected and manipulated interactively by the end-user. When this resource is set to False, the object cannot be selected, moved or resized by the end-user. When this resource is set to True, the object can be selected and manipulated according to the settings of resources such as **XmNmove**.

XmNshape

Specifies whether or not a Graphic object can be shaped interactively by the end-user. For most objects, the shape is equivalent to a resize. For objects based on the MultiPoint class (like Polyline), the shape operation allows the end-user to edit the points of the object.

XmNstippleColor

Specifies the color (as a pixel value) used to draw the set bits of the fill pixmap when the resource **XmNfillStyle** is set to XintFILL_STIPPLED or XintFILL_OPAQUE_STIPPLED.

XmNuserData

Allows the application to attach any specific data to the Graphic object. This resource is not used internally.

XmNverifyCallback

Specifies a callback list, called each time an object is modified interactively by the end-user. The operation type, new object position and
new object size are returned in the callback structure. A confirmation flag allows the application to cancel the operation.

XmNverticalAxis

Specifies the ID of an Axis object which the Graphic object will use for its vertical coordinate system. If this resource is set to NULL, the vertical coordinate system of the object’s parent will be used.

XmNvisible

Specifies whether or not the object is drawn on the screen.

Defined Callbacks

The following callback is defined by the Graphic object class.

Name	Structure	Reason
XmNverifyCallback	XintGraphicVerifyCallbackStruct	XintCR_OBJECT_MOVE XintCR_OBJECT_SHAPE XintCR_OBJECT_ADD_POINT XintCR_OBJECT_DELETE_POINT

Whenever a Graphic object has been modified interactively by the end-user, callback XmNverifyCallback is invoked so that the application can validate the modification. The operation type, the new position and size of the object are returned in the callback structure. A confirmation flag, *doit*, allows the application to cancel the operation.

The following ordered table lists the members of the callback structure, XintGraphicVerifyCallbackStruct, associated with callback XmNverifyCallback.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
Boolean	doit	Set to False to cancel the operation.

Some of the subclasses of the Graphic object class define different callback structures for the callback list specified as the value of the **XmNverifyCallback** resource. The above structure describes only the common members in all of the verify callback structures. Please refer to each Graphic object class reference section for a full description of the data structure returned to the verify callback list.

Graphic Functions

The following functions can be applied on any graphic object.

XintGraphicUnmanageDialog

Unmanages the dialog panel associated with the specified object. Some objects, like Text, AxisObject or Chart have a built-in dialog panel that can be used to edit the object resources interactively. If the specified object is a group, this function will also unmanage all panels that may be associated with objects contained in the group.

```
XintGraphicUnmanageDialog (Object object)
```

object	The Object ID.

XintGraphicGetViewPortList

Retrieves the list of ViewPort objects that are associated with the specified object. ViewPort objects can be used to “clone” an object in another window.

```
Object *XintGraphicGetViewPortList (Object object, int *cnt)
```

object	The Object ID.
cnt	Contains when the function returns the number of ViewPort objects attached to *object*. The list of objects returned by the function should not be freed or modified by the application.

Macros

Macro XintIsGraphic returns True if the specified *object* is a Graphic object.

```
Boolean XintIsGraphic (Object object)
```

Group Object Class

Group is a special object class used to group multiple graphic objects into a single object. Because groups are graphic objects, you can create nested groups.

Note: The Group class is also used as the base class for more complex object classes such as Chart or Plot2D.

Interactive Grouping

The EditObject class defines two convenience functions to manipulate groups. Function XintEditObjectGroup automatically creates a group from the selected objects in the display. Function XintEditObjectUngroup ungroups the objects from the selected group object and destroys the empty group object. See the EditObject Widget reference section for more information on those two functions.

Example

The following code fragment illustrates how to create a group from a text and a line object.

Code

```
Widget edit; /* The parent EditObject widget */
XintTextLocation text_location;
XintLine line_location;
Object group;
...
/* Create the group */
group = (Object) XtVaCreateWidget(“group”,
                  (WidgetClass)xintGroupObjectClass, edit,
                  XmNlist, list,
                  XmNlistCount, 2,
                  NULL);

/* create an Text object and insert in the group */
text_location.x = 50;
text_location.y = 50;
XtVaCreateWidget(“text”, (WidgetClass)xintTextObjectClass, edit,
                  XmNgroup, group,
                  XmNtextLocation, &text_location,
                  XmNtextAnchor, XintBOTTOM_RIGHT,
                  XmNtextString, “Something there?”,
                  XmNfontSize, 12,
                  XmNroundEdge, True,
                  XmNfillStyle, XintFILL_SOLID,
                  XmNlineStyle, XintLINE_SOLID,
                  NULL);

/* Create line object and insert it in the group */
```

```
line_location.start_x = 50;
line_location.start_y = 50;
line_location.end_x = 70;
line_location.end_y = 70;
XtVaCreateWidget(“line”, (WidgetClass)xintLineObjectClass, edit,
                         XmNgroup, group,
                         XmNline, &line_location,
                         XmNlineThickness, 2,
                         XmNlineEnd, XintEND_ARROW,
                         NULL);

...
```

The output from this example is shown in Figure 27:

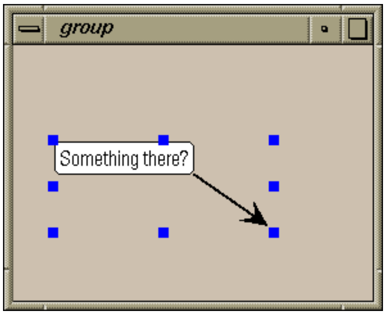


Figure 27. Group Created from a Text and a Line Object

Group Resources

The Group object class inherits behavior and resources from the *Xt Object* and *Graphic* classes:

- Class pointer is *xintGroupObjectClass*
- Class name is *XintGroup*
- Header file is included as `<Xint/Group.h>`

The following resources are defined by the Group object class.

Name	Type Default	Access
XmNlist	Object * NULL	CSG
XmNlistCount	int 0	CSG
XmNpropagate	Boolean False	S

XmNlist

Specifies the list of objects in the group.

XmNlistCount

Specifies the size of the object list defined using resource **XmNlist**.

XmNpropagate

Specifies whether resources in a XtSetValues call are propagated to the objects in the group. After it is used, this resource is always reset to False.

Group Functions

Function XintCreateGroup creates a Group object.

```
Object XintCreateGroup (...)
```

Widget	parent	Parent of new Group object.
char *	name	Name of new Group object.
ArgList	arglist	List of resource/value items.
Cardinal	argcount	Number of items in arglist.

Macros

Macro XintIsGroup returns True if the specified *object* is a Group object.

```
Boolean XintIsGroup (Object object)
```

Image Object Class

The ImageObject class displays an image that is specified as a pixmap. The location and size of the image is specified in user coordinates using inherited resource **XmNrectangle**. If resource **XmNimageDisplayMode** is set to XintIMAGE_INTERPOLATE, the image is interpolated to fit into the specified rectangle. If resource **XmNimageDisplayMode** is set to XintIMAGE_FIXED, the rectangle end points are modified to match the image size.

If the input pixmap is of depth 1, inherited resources **XmNfillColor** (unset bits) and **XmNstippleColor** (set bits) control the color of the bitmap. If inherited resource **XmNfillStyle** (from Graphic class) is set to XintFILL_STIPPLED, the bitmap is drawn in transparent mode, i.e. only the set bits are drawn.

Image Resources

The Image object class inherits behavior and resources from the *Xt Object*, *Graphic* and *Rectangle* classes*:*

- Class po inter is *xintImageObjectClass*
- Class name is *XintImageObject*
- Header file is included as `<Xint/ImageObject.h>`

The following resources are defined by the Image object class.

Name	**Type** **Default**	**Access**
XmNimageColorRecord	XintColorRec * NULL	CSG
XmNimageDisplayMode	int XintIMAGE_INTERPOLATE	CSG
XmNimagePixmap	Pixmap XmUNSPECIFIED_PIXMAP	CSG
XmNfreePixmap	Boolean True	CG

XmNimageColorRecord

Specifies a pointer to a color record structure describing the colors used by the input pixmap **XmNimagePixmap**. This resource needs only to be specified in cases where the image needs to be saved into an ASCII file (using function XintEditObjectWriteFile for example) or cut and pasted from one application to another, so that the colors used by the pixmap get saved and reallocated when the image object is restored. See function XintChartCreateColorRecord to build a color record structure. If used, the color record structure should contain a list of all the pixels used by the image pixmap.

XmNimageDisplayMode

This resource controls how the image is processed. The value of this resource is specified as one of the defined constants listed below:

Resource Value	Description
XintIMAGE_FIXED	The size of the rectangle specified with resource **XmNrectangle** is adjusted to match exactly the size of the input pixmap.
XintIMAGE_INTERPOLATE (default)	The image is interpolated to fit the rectangle specified with resource **XmNrectangle**.

XmNimagePixmap

Specifies the pixmap to display by the image object. This pixmap depth should be 1 or equal to the depth of the widget containing the image.

XmNfreePixmap

Specifies whether or not to free the pixmap, specified using resource **XmNimagePixmap**, when the Image object is destroyed or when a new pixmap is provided in a SetValues operation.

Defined Functions

Function XintCreateImageObject creates an Image object.

```
Object XintCreateImageObject (...)
```

Widget	parent	Parent of new Image object.
char *	name	Name of new Image object.
ArgList	arglist	List of resource/value items.
Cardinal	argcount	Number of items in arglist.

Macros

Macro XintIsImageObject returns True if the specified *object* is an Image object.

```
Boolean XintIsImageObject (Object object)
```

Line Object Class

The Line object class draws a line between two points. Arrows can be drawn on each of the end points of the Line object. Most of the graphic attributes specifying the line color, size and style are set using resources defined in Line's superclass, the Graphic object class. The Line end points are specified by passing a pointer to a data structure of type XintLine.

Arrow Shape

The shape of the arrow is described using three resources, **XmNbaseAngle** to specify the base angle, **XmNtipAngle** to specify the tip angle and **XmNarrowLength** to specify the length of the arrow. The meaning of those resources is illustrated in Figure 28:

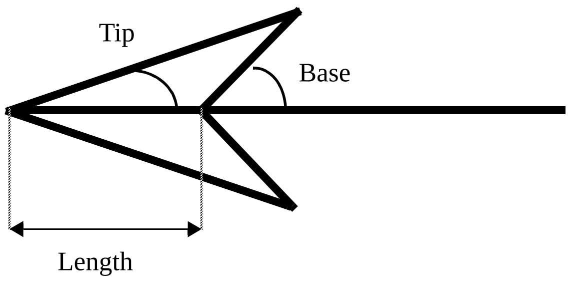


Figure 28. Arrow Shape Resources

Line Resources

The Line object class inherits behavior and resources from the *Xt Object* and *Graphic* classes:

- Class pointer is *xintLineObjectClass*
- Class name is *XintLine*
- Header file is included as `<Xint/Line.h>`

The following resources are defined by the Line object class.

Name	**Type** **Default**	**Access**
XmNarrowLength	int 8	CSG
XmNarrowStyle	int XintFILLED	CSG
XmNbaseAngle	int 45	CSG
XmNline	XintLine * {{0,0}, {1,1}}	CSG
XmNlineEnd	int XintNO_ARROW	CSG
XmNtipAngle	int 25	CSG

XmNarrowLength

Specifies the length of the arrow in pixel units. This resource only applies if resource **XmNlineEnd** is not set to XintNO_ARROW.

XmNarrowStyle

Specifies the style of arrowhead to draw. This resource only applies if resource **XmNlineEnd** is not set to XintNO_ARROW. You can specify one of the following constants:

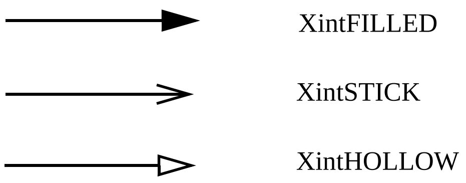


Figure 29. XmNarrowStyle Constants

XmNbaseAngle

Specifies the arrow base angle in degrees. This resource only applies if resource **XmNlineEnd** is not set to XintNO_ARROW.

XmNline

Specifies the end points in user coordinates. This resource is specified as a pointer to a data structure of type XintLine which takes the following form:

```
typedef struct {
          float start_x;
          float start_y;
          float end_x;
          float end_y;
} XintLine;
```

where

Member	Description
start_x, start_y	Coordinates of the line starting point.
end_x, end_y	Coordinates of the line ending point.

XmNlineEnd

Specifies whether arrows are drawn at the ends of the Line object or not. You can specify one of the following constants:

Resource Value	Description
XintNO_ARROW	No arrow is drawn.
XintDOUBLE_ARROW	An arrow on each end of the line is drawn.
XintEND_ARROW	An arrow on the line ending point is drawn.
XintSTART_ARROW	An arrow on the line starting point is draw.

XmNtipAngle

Specifies the tip angle of the arrow in degrees. This resource only applies when the resource **XmNlineEnd** is not set to XintNO_ARROW.

Line Callbacks

The Line object class does not define any new callbacks. However, the callback structure returned with callback XmNverifyCallback (see class Graphic) is redefined. The following ordered table lists the members of the callback structure XintLineVerifyCallbackStruct.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
Boolean	doit	Set to False to cancel the operation on the Line object.
XintLine *	old_line	Current end points for the line.
XintLine *	new_line	Proposed new end points for the line.

Line Functions

Function XintCreateLine creates a Line object.

```
Object XintCreateLine (...)
```

Widget	parent	Parent of new Line object.
char *	name	Name of new Line object.
ArgList	arglist	List of resource/value items.
Cardinal	argcount	Number of items in arglist.

Macros

Macro XintIsLine returns True if the specified *object* is a Line object.

```
Boolean XintIsLine (Object object)
```

MultiPoint Object Metaclass

The MultiPoint class is a superclass for all objects that can be defined using a list of points. It defines the symbology for the points, including the symbol type, size and color. This object class must not be instantiated directly.

MultiPoint Object Interactive Creation

Most Graphic objects can be created interactively with function XintEditObjectInsert using translations BSelect Drag (to initiate the insertion) and BSelect Release (to terminate the insertion). The insertion translations for MultiPoint object are different, since they must give the user the ability to specify any number of points. The default translation to insert a MultiPoint object is BSelect to specify a point and Btn2Down to signal that the last point has been inserted.

MultiPoint Object Editing

The Select, Move and Size editing operations are supported by the MultiPoint object class. The Shape operation has a different meaning than for rectangular base objects. For a MultiPoint object, a Shape operation is equivalent to a point move operation.

Two editing operations are defined exclusively for MultiPoint objects. They are, ObjectPointAdd (for adding points to an object) and ObjectPointDelete (for deleting points from an object).

Inherited Behavior and Resources

The MultiPoint object class inherits behavior and resources from the *Xt Object* and *Graphic* classes:

- Class pointer is *xintMultiPointObjectClass*
- Class name is *XintMultiPoint*
- Header file is included as `<Xint/MultiPoint.h>`

The following resources are defined by the MultiPoint object class.

Name	Type Default	Access
XmNhandleMode	int XintHANDLE_POINTS	CSG
XmNsymbol	Boolean False	CSG
XmNsymbolColor	Pixel Foreground	CSG
XmNsymbolData	XintSymbolData * NULL	CSG
XmNsymbolSize	int 8	CSG
XmNsymbolType	int XintSYMBOL_PLUS	CSG

XmNhandleMode

Specifies where the handle bars are drawn when a MultiPoint object is selected. Specify XintHANDLE_POINTS to have a handle bar drawn at each point location or XintHANDLE_BOUNDS to have a handle bar drawn at each of the four corners of the bounding box. This resource only applies if Graphic resource **XmNhighlightMode** is set to XintHIGHLIGHT_HANDLE.

XmNsymbol

Specifies whether or not symbols are drawn at the point locations.

XmNsymbolColor

Specifies the pixel color used to draw point symbols, if resource **XmNsymbol** is True.

XmNsymbolData

Specifies a pointer to a symbol descriptor, as returned by Symbol function XintSymbolCreate. This resource allows the application to create its own symbols to display points. To display a symbol defined by **XmNsymbolData**, resource **XmNsymbolType** must be set to XintSYMBOL_DATA.

XmNsymbolSize

Specifies the size, in pixels, used to draw point symbols, if resource **XmNsymbol** is True.

XmNsymbolType

Specifies the type of symbol used to draw points, if resource **XmNsymbol** is True. One of the following constants can be specified:

Resource Value	**Description**
XintSYMBOL_X	Draws an “X”.
XintSYMBOL_PLUS	Draws a “+”.
XintSYMBOL_SQUARE	Draws a square.
XintSYMBOL_CIRCLE	Draws a circle.
XintSYMBOL_TRIANGLE	Draws a triangle.
XintSYMBOL_DIAMOND	Draws a diamond.
XintSYMBOL_FILLED_SQUARE	Draws a filled square.
XintSYMBOL_FILLED_CIRCLE	Draws a filled circle.
XintSYMBOL_FILLED_TRIANGLE	Draws a filled triangle.
XintSYMBOL_FILLED_DIAMOND	Draws a filled diamond.
XintSYMBOL_DATA	Draws the symbol specified by the resource **XmNsymbolData**.

Oval Object Class

The Oval object class allows the application to draw an oval shaped object (circular, elliptical, etc.). The coordinates of the Oval are specified in a data structure of type XintRectangle which is defined by the Rectangle object class. Graphic attributes such as the line color, line width, fill style and fill color are set using resources defined in the Graphic class.

Inherited Behavior and Resources

The Oval object class inherits behavior and resources from the *Xt Object, Graphic* and *Rectangle* classes:

- Class pointer is *xintOvalObjectClass*
- Class name is *XintOval*
- Header file is included as `<Xint/Oval.h>`

The Oval object class does not define any new resources. The position and shape of the Oval object are defined using the inherited resource **XmNrectangle**.

Oval Callbacks

The XintRectangleVerifyCallbackStruct callback structure is returned by the callback XmNverifyCallback. Please see the reference section of the Rectangle Object Class for more information regarding this structure.

Oval Functions

Function XintCreateOval creates an oval object.

```
Object XintCreateOval (...)
```

Widget	parent	Parent of new Oval object.
char *	name	Name of new Oval object.
ArgList	arglist	List of resource/value items.
Cardinal	argcount	Number of items in arglist.

Macros

Macro XintIsOval returns True if the specified *object* is an Oval object.

```
Boolean XintIsOval (Object object)
```

Polyline Object Class

The Polyline object class can display a series of points connected by line segments. The end points can be connected in such a way to form an open or closed polyline or filled polygon. In addition, symbols can be drawn at each point location.

Resources to specify the line color, the line thickness, and fill options are inherited from the Graphic class. Resources to specify a point symbol and a point size are inherited from the MultiPoint class.

Creation

A Polyline object can be created as a child of any widget class derived from the EditObject class. Resources are provided to specify the point values and the number of points. A Polyline object can also be created interactively using function XintEditObjectInsert.

Editing

Like any Graphic object, a Polyline object can be edited interactively. Editing operations include move, shape (to move a point), add point and delete point. Editing is specified by installing the proper translation table or by setting resource **XmNobjectEditMode** from the EditObject class.

Optimization

For applications that create a large number of Polyline objects, it is possible to create a single Polyline object composed of multiple discontinuous segments which appear as discrete objects. This is done by defining resource **XmNnullValue** and inserting null points between the sets of points that represent each discrete segment. It is recommended that the programmer only combine segments which are located close together to minimize the object bounding box. This will avoid excessive redrawing when it becomes necessary to repaint a portion of the parent window.

Inherited Behavior and Resources

Polyline object class inherits behavior and resources from the *Xt Object, Graphic* and *MultiPoint* classes:

- Class pointer is *xintPolylineObjectClass*
- Class name is *XintPolyline*
- Header file is included as `<Xint/Polyline.h>`

Resources

The following resources are defined by the Polyline object class:

Name	Type Default	Access
XmNcloseEndPoints	Boolean False	CSG
XmNdrawSymbolCallback	XtCallbackList NULL	CSG
XmNnullValue	float * NULL	CSG
XmNpointArray	XintPolylinePoint * { {10., 10.}, {20., 20.} {30., 30.}}	CSG
XmNpointCount	int 3	CSG

XmNcloseEndPoints

Specifies that the Polyline first and last points are connected with a line to make a closed polygon (True).

XmNdrawSymbolCallback

Specifies list of callbacks called each time a point symbol is drawn. Can be used to draw polylines where point symbols have different sizes, colors and symbol types. Callback structure is XintPolylineDrawSymbolCallbackStruct and the callback reason is XintCR_DRAW_SYMBOL.

XmNnullValue

Specifies pointer to floating point value used to represent null or missing values. Missing values introduce discontinuity in the polyline. For filled polygons, segments between null values are filled separately. You must set only one polyline coordinates (x or y) to null value to specify a missing value.

XmNpointArray

Specifies the coordinates of the points contained in the Polyline object. The polyline is specified as an array of points, where each point is specified using a structure of type XintPolylinePoint which takes the following form:

```
typedef struct {
    float x;
    float y;
} XintPolylinePoint;
```

XmNpointCount

Specifies the number of points passed in resource **XmNpointArray**.

Polyline Callbacks

The following table lists the XintPolylineDrawSymbolCallbackStruct members returned to each procedure in the callback list specified by resource **XmNdrawSymbolCallback**:

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
Boolean	doit	Set to False to prevent symbol from being drawn.
int	index	Index of the point which symbol is being drawn.
double	x,y	Location of the symbol.
double	symbol_size	Symbol size (can be changed).
Pixel	symbol_color	Symbol color (can be changed).
int	symbol_type	Symbol type (can be changed).

Structure member *symbol_type* can be set to one of the following constants:

Member Value	Description
XintSYMBOL_X	Draws an “X”.
XintSYMBOL_PLUS	Draws a “+”.
XintSYMBOL_SQUARE	Draws a square.
XintSYMBOL_CIRCLE	Draws a circle.
XintSYMBOL_TRIANGLE	Draws a triangle.
XintSYMBOL_DIAMOND	Draws a diamond.
XintSYMBOL_FILLED_SQUARE	Draws a filled square.
XintSYMBOL_FILLED_CIRCLE	Draws a filled circle.
XintSYMBOL_FILLED_TRIANGLE	Draws a filled triangle.
XintSYMBOL_FILLED_DIAMOND	Draws a filled diamond.

Polyline Functions

Function XintCreatePolyline creates a polyline object.

```
Object XintCreatePolyline (...)
```

Widget	parent	Parent of new Polyline object.
char *	name	Name of new Polyline object.
ArgList	arglist	List of resource/value items.
Cardinal	argcount	Number of items in arglist.

Macros

Macro XintIsPolyline returns True if the specified *object* is a Polyline object.

```
Boolean XintIsPolyline (Object object)
```

Rectangle Object Class

The Rectangle object class allows the application to draw a rectangular object. It is also used as a base class for other object classes that are drawn in rectangular form such as a Text object. The coordinates of the rectangle are specified in a data structure of type XintRectangle. Graphic attributes such as the line color, line width, fill style and fill color are set using resources defined in the Graphic class.

Inherited Behavior and Resources

The Rectangle object class inherits behavior and resources from the *Xt Object* and *Graphic* classes:

- Class pointer is *xintRectangleObjectClass*
- Class name is *XintRectangle*
- Header file is included as `<Xint/Rectangle.h>`

Resources

The following resources are defined by the Rectangle object class:

Name	Type Default	Access
XmNrectangle	XintRectangle * {10, 10, 25, 25}	CSG
XmNrotateAngle	int 0	CSG
XmNroundEdge	Boolean False	CSG

XmNrectangle

Specifies the coordinates of the rectangle's two opposite points using the data type XintRectangle. This resource is specified as a pointer to a data structure which takes the following form:

```
typedef struct {
    float x1;
    float y1;
    float x2;
    float y2;
} XintRectangle;
```

where

Member	Description
x1,y1	Coordinates of upper left corner of Rectangle object box.
x2, y2	Coordinates of lower right corner of Rectangle object box.

XmNrotateAngle

Specifies an angle of rotation in degrees about the center of the rectangle.

XmNroundEdge

Specifies whether the corners of the rectangle are rectangular (False) or rounded (True).

Rectangle Callbacks

The Rectangle object class does not define any new callbacks. However, the callback structure returned with callback XmNverifyCallback (see class Graphic) is redefined. The following ordered table lists the members of the callback structure XintRectangleVerifyCallbackStruct.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
Boolean	doit	Set to False to cancel the operation.
XintRectangle *	data	Pointer to the new representation proposed for the Rectangle object.

Rectangle Functions

Function XintCreateRectangle creates a rectangular object.

```
Object XintCreateRectangle (...)
```

Widget	parent	Parent of new Rectangle object.
char *	name	Name of new Rectangle object.
ArgList	arglist	List of resource/value items.
Cardinal	argcount	Number of items in arglist.

Macros

Macro XintIsRectangle returns True if the specified *object* is a Rectangle object.

```
Boolean XintIsRectangle (Object object)
```

Symbol Object Class

The Symbol object class allows the application to define and draw custom symbols. The Symbol object class defines a simple ASCII language that can be used to describe a user symbol. Once defined, Symbol objects can be resized, moved, selected, output to hardcopy like any other objects based on the Graphic class.

The size of a symbol can be specified in pixels (for a fixed size symbol) using resources **XmNsymbolWidth** and **XmNsymbolHeight** or in user coordinates using resource **XmNsymbolScale**. When a symbol size is specified in user coordinates, its size will change when the coordinate system changes.

The Symbol object defines a text primitive which draws text inside a symbol. This text, which is fully scalable, is drawn using an outline font technology provided with the INT library. When this text primitive is used, the symbol object must access a file containing a description of the outline for the specified font. The font descriptions, must reside in a directory which you can specify as follows:

- Set CompBase resource **XmNfontPath** on the Symbol parent widget.
- Set environment variable INT_FONT_PATH.

If neither of these is defined, the current directory will be searched. If a valid font description is not found, a stroke font will be used.

Symbol Editor

A symbol editor is built in the symbol object class. This editor allows the end-user to create or modify a symbol object interactively. This editor, like other object editors, can be invoked by double clicking on a symbol object. The editor can also be invoked from an application using function XintEditObjectManageResourceDialog. Figure 30 shows the symbol editor:

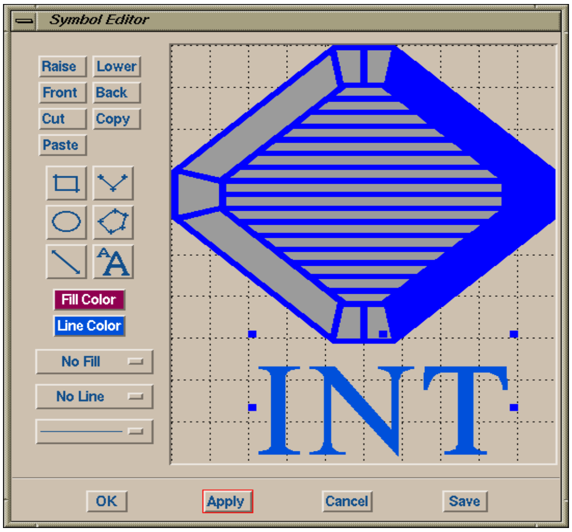


Figure 30. Symbol Editor Example

Symbol Resources

The Symbol object class inherits behavior and resources from the *Xt Object* and *Graphic* classes:

- Class pointer is *xintSymbolObjectClass*
- Class name is *XintSymbol*
- Header file is included as `<Xint/Symbol.h>`

Resources

The following resources are defined by the Symbol object class.

Name	Type Default	Access
XmNlabelGravity	int EastGravity	CSG
XmNlabelSpacing	int 10	CSG
XmNsymbolData	XintSymbolData * NULL	CSG
XmNsymbolHeight	int 21	CSG
XmNsymbolWidth	int 21	CSG
XmNsymbolLocation	XintSymbolLocation * NULL	CSG
XmNsymbolScale	XintSymbolScale * NULL	CSG

XmNlabelGravity

Specifies where to draw an optional label string, defined using the Graphic resource **XmNname**. The label will appear at the specified location if Graphic resource **XmNdisplayName** is set to True. You can specify one of the following constants:

Resource Value	Description
NorthGravity	Label is drawn above the symbol.
EastGravity (default)	Label is drawn to the right of the symbol.
SouthGravity	Label is drawn below the symbol.
WestGravity	Label is drawn to the left of the symbol.

XmNlabelSpacing

Specifies the space in pixels between the symbol and the label. See resource **XmNlabelGravity** for information on how to display a symbol label.

XmNsymbolData

Specifies a pointer to an opaque structure of type XintSymbolData that contains a parsed description of a symbol. See function XintSymbolCreate to create a parsed symbol.

XmNsymbolHeight
XmNsymbolWidth

Specifies the height and width of the symbol in pixels. If you want to specify the size of the symbol in user coordinates, use resource **XmNsymbolScale** instead.

XmNsymbolLocation

Specifies the position of the center of the symbol in user coordinates. This resource is specified using a pointer to a structure of type XintSymbolLocation which takes the following form:

```
typedef struct {
    float x;
    float y;
} XintSymbolLocation;
```

XmNsymbolScale

Specifies the size of the symbol in user coordinates. This resource is specified using a pointer to a structure of type XintSymbolScale which takes the following form:

```
typedef struct {
    float width;
    float height;
} XintSymbolScale;
```

To specify a symbol with a fixed size in pixels, set resource **XmNsymbolScale** to NULL and specify the symbol size using resources **XmNsymbolWidth** and **XmNsymbolHeight** instead.

Symbol Keywords

The following table lists the keywords available for describing a symbol:

Keyword	Arguments	Description
BoundingBox	minx miny maxx maxy	Specifies the bounding box, which defines the coordinate system used to specify the symbol. The origin is located at the upper left corner.
LineColor	color_name	Specifies the current color used to draw lines.
FillColor	color_name	Specifies the current fill color.
LineThickness	line_thickness	Specifies the current line width in pixels.
LineStyle	line_style	Specifies the current line style. Possible choices are line_on_off_dash, line_double_dash and line_solid.
Line	x1 y1 x2 y2	Draw a line from coordinate (x1, y1) to (x2, y2). Coordinates are specified within the coordinate system defined by BoundingBox.
Polyline	x1 y1 x2 y2...xn yn	Draw a polyline through the specified points
Polygon	x1 y1 x2 y2...xn yn	Draw a filled polygon through the specified points.
Rectangle	x y width height	Draw a rectangle
FillRectangle	x y width height	Draw a filled rectangle
Arc	x y width height angle1 angle2	Draw an arc inside a box, starting at angle1 with extension angle2. Angles are expressed in degrees.
FillArc	x y width height angle1 angle2	Draw a filled arc inside a box, starting at angle1 with extension angle2. Angles are expressed in degrees.
Font	family weight slant	Select a font to be used by the Text command.Argument *family* can be set to helvetica, times, courier, symbol or new-century, *weight* can be set to medium or bold, and *slant* can be set to regular or italic.
Text	x y width height string	Draw a string in the specified bounding box. If argument *width* is set to 0, the text width is calculated automatically based on the *height* specification. If argument *height* is set to 0, the text height is calculated automatically based on the *width* specification.

Symbol Description Syntax

If specified, keyword BoundingBox must be placed first (default is-100 -100 100

100). Other keywords can be specified in any order, each starting on a new line.

Code

The following code example illustrates a symbol description:

```
BoundingBox -100 -100 100 100
LineColor blue
FillColor red
LineThickness 5
Line -28.4116 20.4211 -40.0447 -13.2632
Polyline -19.9105 -13.6842 -9.17226 21.6842 2.01342 -14.9474
       13.6465 21.2 632
Polyline 21.2528 19.5789 42.7293 19.1579 39.5973 19.5789
Line 32.4385 19.5789 22.5951 -16.2105
Polyline -16.7785 85.2632 -81.6555 16.2105 -81.6555 -9.89474
       18.5682 84.8421 1.56599 85.2632
Polyline 11.8568 73.8947 62.8635 13.6842 62.8635 -11.1579
       10.0671 -56.2105 12.7517 -69.6842 75. 3915 -17.0526
       75.8389 14.9474 18.1208 84.8421 12.7517 84.4211
       12.7517 74.3158
```

Symbol Callbacks

The Symbol object class does not define any new callbacks. However, the callback structure returned with callback XmNverifyCallback (see class Graphic) is redefined. The following ordered table lists the members of the callback structure XintSymbolVerifyCallbackStruct:

Data Type	Member	Description
int	reason	Indicates why callback was invoked.
XEvent *	event	Points to XEvent that triggered callback.
Boolean	doit	Set to False to cancel the operation.
XintSymbolLocation *	location	Pointer to new location proposed for Symbol object.
int	symbol_width	Proposed width for symbol in pixels.
int	symbol_height	Proposed height for symbol in pixels.

Symbol Functions

The Symbol object class defines the following functions for creating a Symbol object, and creating and freeing parsed symbol data structures:

Function Name	Description
XintCreateSymbol	Creates a Symbol object.
XintSymbolCreate	Creates a parsed symbol from a description string.
XintSymbolFree	Free a parsed symbol data structure.

XintCreateSymbol

Creates a symbol object.

```
Object XintCreateSymbol (...)
```

Widget	parent	Parent of new Symbol object.
char *	name	Name of new Symbol object.
ArgList	arglist	List of resource/value items.
Cardinal	argcount	Number of items in arglist.

XintSymbolCreate

Converts a string containing a symbol description into a parsed symbol data structure that can be passed to resource **XmNsymbolData**. A symbol data structure can be shared between several Symbol objects.

```
XintSymbolData *XintSymbolCreate (char *symbol_string)
```

symbol_string A string containing a description of the symbol to parse.

Structure XintSymbolData is an opaque structure that is not documented.

XintSymbolFree

Frees a parsed symbol data structure. You can free a parsed symbol descriptor immediately after its creation or after a set values operation since it is no longer needed (the Symbol object makes a copy).

```
void XintSymbolFree(XintSymbolData *parsed_symbol)
```

parsed_symbol A pointer to the parsed symbol data structure to free.

Macros

Macro XintIsSymbol returns True if the specified *object* is a Symbol object.

```
Boolean XintIsSymbol (Object object)
```

Text Object Class

Text object class draws a character string that can span multiple lines within a rectangular area. Derived from Rectangle class, Text class also allows you to draw a background fill and/or an outline around the text.

Text object class can display fixed size text using the standard X bitmap font technology or scalable text, or using an outlined font technology provided with the INT library. Resource **XmNtextScale** controls whether fixed-size or scalable fonts are used. If **XmNtextScale** is NULL, a fixed-size font will be used. The size of this font, specified using resource **XmNfontSize**, must be a valid X font size. Otherwise, if **XmNtextScale** is not NULL, a scalable font is used. The size of scalable text object is specified in user coordinates, and its size changes when the coordinate system changes (for example if you resize the parent widget window). Figure 31 illustrates and example of scalable text:

f

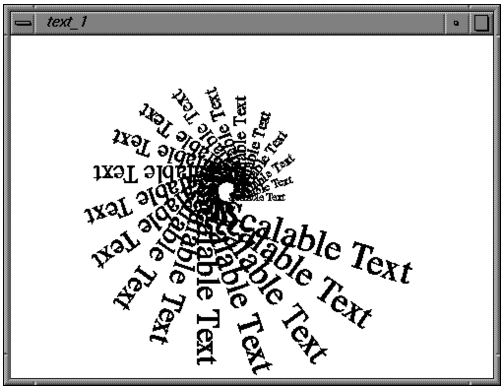


Figure 31. Scalable Text

Font description directories

When using scalable text, the Text object must access a file containing the description of the outline for the specified font. The font descriptions must reside in a directory which you can specify as follows:

- Set CompBase resource **XmNfontPath** on the Text parent widget
- Set environment variable INT_FONT_PATH

If not defined, the current directory is searched, and if a valid font description is not found, a default stroke font is used. Text object supports rotation for both fixed-size and scalable text. The position of text is specified in user coordinates using structure XintTextLocation and resource **XmNtextLocation**. To controls the positioning of text relative to this location, use resources **XmNhorizontalTextAlignment** and **XmNverticalTextAlignment.**

Text Resources

The Text object class inherits behavior and resources from the *Xt Object, Graphic* and *Rectangle* classes.

- Class pointer is *xintTextObjectClass*
- Class name is *XintText*
- Header file is included as `<Xint/Text.h>`

Resources The following resources are defined by the Text object class.

Name	Type Default	Access
XmNfontFamily	int XintHELVETICA	CSG
XmNfontSize	int 10	CSG
XmNfontSlant	int XintSLANT_REGULAR	CSG
XmNfontWeight	int XintWEIGHT_MEDIUM	CSG
XmNhorizontalTextAlignment	int XintHALIGN_LEFT	CSG
XmNmarginHeight	Dimension 4	CSG
XmNmarginWidth	Dimension 4	CSG
XmNrotateAngle	int 0	CSG
XmNtextLocation	XintTextLocation * NULL	CSG
XmNtextScale	XintTextScale * NULL	CSG
XmNtextString	char * NULL	CSG
XmNverticalTextAlignment	int XintVALIGN_TOP	CSG

XmNfontFamily

Specifies the font family used to draw the text string. You can specify one of the following constants:

Resource Value	Description
XintDEFAULT	Use default font family. For scalable text, a stroke font is selected. For fixed size text, the default X font for the specified size and weight is selected.
XintHELVETICA (default)	Use Helvetica font family.
XintTIMES	Use Times font family.
XintCOURIER	Use Courier font family.
XintNEW_CENTURY_SCHOOLBOOK	Use New Century Schoolbook font family
XintSYMBOL	Use Symbol font family.

XmNfontSize

Specifies size in points of font. Specify XintFONT_SIZE_DEFAULT to obtain default font for specified family. Valid font sizes are 8, 10, 12, 14, 18 and 24.

XmNfontSlant

Specifies the slant of the font used to draw the text string. You can specifiy XintSLANT_REGULAR, XintSLANT_OBLIQUE or XintSLANT_DEFAULT.

XmNfontWeight

Specifies the weight of the font used to draw the text string. You can specify XintWEIGHT_MEDIUM or XintWEIGHT_BOLD or XintWEIGHT_DEFAULT.

XmNhorizontalTextAlignment

Specifies the horizontal alignment of the text. You can specify one of the following constants:

Resource Value	Description
XintHALIGN_LEFT	Text is aligned left in the horizontal direction.
XintHALIGN_CENTER	Text is centered in the horizontal direction.
XintHALIGN_RIGHT	Text is aligned right in the horizontal direction.

XmNmarginHeight

Specifies space (in pixels) between text and top and bottom edges of rectangular area containing the text.

XmNmarginWidth

Specifies space (in pixels) between text and left and right edges of rectangular area containing the text.

XmNrotateAngle

Specifies an angle of rotation in degrees for the text object. The text rotates about the location defined in resource **XmNtextLocation**.

XmNtextLocation

Specifies location of text object in user coordinates. Specified as a pointer to a data structure of type XintTextLocation which takes the following form:

```
typedef struct {
    float x;
    float y;
} XintTextLocation;
```

XmNtextScale

Specifies text font size in user coordinates and a horizontal stretch factor. When this resource is set, a scalable font will be used to display the text string. The stretch value should be set to 1.0 unless you want to stretch (> 1.0) or shrink (< 1.0) the text in the horizontal direction. Specified as a pointer to data structure of type XintTextScale that takes the following form:

```
typedef struct {
    float size;
    float stretch;
} XintTextScale;
```

XmNtextString

Specifies string to display. Use the new line symbol ‘\n‘ to separate lines.

XmNverticalTextAlignment

Specifies vertical alignment of text with respect to text location specified in resource **XmNtextLocation**. You can specify one of the following constants:

Resource Value	Description
XintVALIGN_TOP	Text aligned with respect to top in vertical direction.
XintVALIGN_CENTER	Text centered in vertical direction.
XintVALIGN_BOTTOM	Text aligned with respect to bottom in vertical direction.

Text Callbacks

The Text object class does not define any new callbacks. However, the callback structure returned with callback XmNverifyCallback (see class Graphic) is redefined. The following ordered table lists the members of the callback structure XintTextVerifyCallbackStruct.

Data Type	**Member**	**Description**
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
Boolean	doit	Set to False to cancel the operation.
XintTextLocation	old_location	Old location of the Text object.
XintTextLocation	location	New location proposed for the Text object.

Defined Functions

The XintCreateText function creates a Text object.

```
Object XintCreateText (...)
```

Widget	parent	Parent of new Text object.
char *	name	Name of new Text object.
ArgList	arglist	List of resource/value items.
Cardinal	argcount	Number of items in arglist.

Macros

Macro XintIsText returns True if the specified *object* is a Text object.

```
Boolean XintIsText (Object object)
```

DataObject Reference

4

Overview

This chapter includes the following sections:

- **DataObject** on page 166
- **DataGroup Object Class** on page 167
- **DataGrid Object** on page 173
- **DataLabel Object** on page 181
- **DataSampled Object Class** on page 188
- **DataSequentialSeries Object Class** on page 196
- **DataSeries Object Class** on page 196
- **DataTimeLabel Object Class** on page 203

DataObject

The INT DataObject is a system of Xt intrinsic tools that provide for the retrieval, storage and manipulation of sets or groups of data. DataObject and its companion ChartObject are based on the *MVC* or *Model-View-Controller* architecture first used in Smalltalk. DataObject serves as the *model* component of the architecture, while ChartObject serves as the *view* and EditObject as the *controller* components.

Summary of Components

DataObject provides the following object classes that can be used to classify and manipulate sets of data:

Class	Description
DataGroup	Allows the definition and manipulation of multiple data objects as a single group. For example, a group may contain sampled objects, series objects, grids, or even other groups.
DataGrid	Allows the definition and manipulation of a two-dimensional array of data values as a single object.
DataLabel	Allows the definition and manipulation of a set of labels that are used to identify visually the data in a chart or tabular view.
DataSampled	Allows the definition and manipulation of a one-dimensional array of data as a single object.
DataSeries	Allows the definition and manipulation of a series of x,y pairs as a single object.

These classes all are Xt objects that can be manipulated using a series of resources, functions, and callbacks, just like any standard Motif/Xt object or widget. The data objects should not be managed, and modification of the data can be accomplished using either standard Xt resource setting mechanisms or the convenience functions provided by INT.

DataGroup Object Class

The DataGroup object class is used to build groups of related objects. This feature is especially useful for transferring groups of data between different views. For example, in a bar chart with multiple bars, a related group of data objects could be plotted simultaneously against the same axis and share the same label(s). The DataGroup object lets you:

- group data objects in a hierarchical relationship
- determine the maximum/minimum range of all data in the group (e.g., for axis scaling).
- find and retrieve specific objects in the group.

A set of function calls can be used to manipulate ranges of data within specified arrays or individual data points in any array. You can also extend, replace, or shift data in an array.

Inherited Behavior and Resources

The DataGroup object class does not inherit behavior and resources from other classes:

- Class pointer is *xintDataGroupObjectClass*
- Class name is *XintDataGroup*
- Header file is included as `<Xint/DataGroup.h>`

Resources The following resources are defined by the DataGroup object class:

Name	Type Default	Access
XmNhistoryLength	int 0	CSG
XmNlastViewDestroy	Boolean True	CSG
XmNlimitsX	XintLimits * NULL	CSG
XmNlimitsY	XintLimits * NULL	CSG
XmNlimitsZ	XintLimits * NULL	CSG

XmNhistoryLength

This resource (not implemented yet) will allow you to specify the number of editing changes that can be saved to the history buffer for the current group. For example, if this number is set to five, the history buffer will save up to the last five editing changes. A special Undo feature will allow you to retrieve any set of historical changes and restore the data to that stage in the editing process.

XmNlastViewDestroy

Specifies whether or not to automatically destroy the data group when there are no more views connected to it. The DataGroup object will automatically change the value of resource **XmNlastViewDestroy** for every data object inserted into the group to match its own value.

XmNlimitsX
XmNlimitsY
XmNlimitsZ

These resources let you limit the range of displayed data in the entire group to a certain limit in the X, Y, and/or Z direction. If not specified, the range of interest is set by default to the actual minimum and maximum values of data in the data group.

The minimum/maximum limits are defined through the data structure XintLimits, which takes the following form:

```
typedef struct {
   float minimum;
   float maximum;
} XintLimits;
```

where:

Member	Description
minimum	Minimum value of data in the group.
maximum	Maximum value of data in the group.

Callback Structure

The callback XmNupdateCallback is used to provide notification of changes to any of the data objects contained in the group. The callback structure takes the following form:

```
typedef struct {
    int         reason;
    XEvent      *event;
    Object      object;
    union {
        XintDataUpdateCallbackStruct         *data;
        XintDataSampledUpdateCallbackStruct  *sampled;
        XintDataSeriesUpdateCallbackStruct   *series;
        XintDataGridUpdateCallbackStruct     *grid;
    } callback_struct;
} XintDataGroupUpdateCallbackStruct;
```

This particular callback is especially convenient because it provides notification of updates to *any* child of the group, regardless of the data object type. Callback XmNupdateCallback and update callbacks structures are also provided for each separate data object type (e.g., XintDataSeriesUpdateCallbackStruct), but it is more convenient to use a single callback for the whole group than trying to install update notification for each member of the group.

The callback indicates which type of object was modified and why. This information can help optimize the redrawing and updating of the window. The following table lists members of XintDataGroupUpdateCallbackStruct in their required sequence:

Data Type	Member	Description
int	reason	Indicates why callback was invoked (see following section).
XEvent *	event	Standard member of Motif callback structure not used in this context (will always be NULL).
Object	object	ID of updated object.
XintDataLabelCallbackStruct *	label	Indicates update performed if *object* is DataLabel object.
XintDataSampledUpdateCallbackStruct *	sampled	Indicates update performed if *object* is DataSampled object.
XintDataSeriesUdateCallbackStruct *	series	Indicates update performed if *object* is a DataSeries object.
XintDataGridUpdateCallbackStruct *	grid	Indicates update performed if *object* is DataGrid object.

The callback structure includes information explaining the reason why a particular update occurred. The following table lists the possible reasons returned by this callback:

Reason	Description
XintCR_DATA_BATCH	A batch update has been performed and any number of changes on the objects in the group could have happen. No information about the changes is available in the callback structure.
XintCR_DATA_GROUP_INSERT_CHILD	Child was inserted using the **XmNDataGroup** resource of the child. The ID of the data object that was inserted is set in member *object*.
XintCR_DATA_GROUP_DELETE_CHILD	Child was deleted through external command (e.g., XtDestroy). The ID of the data object that was destroyed is set in member *object*.
XintCR_DATA_GROUP_UPDATE_CHILD	Child *object* was updated. Information about the type of update is available in callback structure (use member *label* if *object* is a DataLabel, use member *sampled* if *object* is a DataSampled, etc.)

Functions

The following functions are defined for creating and manipulating groups of data objects.

Function Name	Description
XintCreateDataGroup	Creates a data group.
XintDataBatchUpdate	To turn on/off propagation of updates to views.
XintDataRangeX	Determines the total range of data in the X direction.
XintDataRangeY	Determines the total range of data in the Y direction.
XintDataRangeZ	Determines the total range of data in the Z direction.
XintDataGroupFind	Finds a data object of a certain name in a data group.
XintDataGroupIterate	Retrieves the nth data object of a specified class from a data group.

XintCreateDataGroup

Creates a data group of a certain name with a list of associated resource values.

```
Object XintCreateDataGroup (...)
```

Widget	parent	Name of the parent DataObject widget containing this new data group.
char *	name	Name of the new data group.
ArgList	arglist	List of resources to be set for the new data group.
Cardinal	argcount	Total number of resources set by arglist.

XintDataBatchUpdate

Allows or disallows propagation of updates to the views of a data group. This function can be used to batch multiple updates in a data group to minimize flashing. To batch updates, first call this function with *state* flag set to True, perform the changes on the members of the data group, then call this function again with *state* flag set to False.

```
Object XintDataBatchUpdate (...)
```

Object	data	ID of the data object (usually a data group).
Boolean	state	Set to True to freeze updates. Set to False to allow updates.

XintDataRangeX, XintDataRangeY, and XintDataRangeZ

Determines total range of data for all objects in a group (minimum and maximum values represented in all data arrays). Useful for determining length, end points, and increments along an axis used to plot objects. Returns False there are no samples or all samples are set to null values.

```
Boolean XintDataRangeX (...)
Boolean XintDataRangeY (...)
Boolean XintDataRangeZ (...)
```

Object	object	Object ID of the data group to be examined.
float	minimum	Minimum value in the data range.
float	maximum	Maximum value in the data range.

XintDataGroupFind

Specifies data group and finds an object by name. For example, if you named an object MyData and inserted it into a group, you could retrieve it by name.

```
Object XintDataGroupFind (...)
```

Object	object	Object ID of the data group to be searched.
char *	name	Name of object to be found.
Object	context	Normally NULL. Can use to search for multiple objects with same name. For example, if called with NULL and return value is not NULL, uses return value as *context* argument of next call.

XintDataGroupIterate

Retrieves specific type of data object from a group by index number. To retrieve a series of objects of the same type, start at index 0 and iterate on index numbers. Returns NULL when objects are no longer available.

```
Object XintDataGroupIterate (...)
```

Object	object	ID of the data group.
ObjectClass	data_class	Class name of data object class to retrieve (for example, intDataSampledObjectClass). Specify NULL to retrieve data objects from any type.
int	index	Index number of the data object to be retrieved.

Macros

Macro XintIsDataGroup returns True if specified *object* is a DataGroup object.

```
Boolean XintIsDataGroup (Object object)
```

DataGrid Object

The DataGrid object class is used to store, retrieve and manipulate two-dimensional arrays or *grids* of data. Each data group may contain multiple grid objects, such as multiple blocks of cells in a table. The grid is described by pointing to a defined two-dimensional array, then describing the following:

- Total count of X and Y values in the array
- Order or orientation of data in the array (for example X-oriented, Y-oriented)
- Min/max limits of the data

Data type

The type of data contained in a DataGrid Object can any of the following:

- Float
- Short
- Integer
- Long
- Double

Example

The following example shows the code structure for a typical DataGrid object:

Code

```
float grid[NX*NY];
Object grid_data;
...
for (i=0; i<NX*NY; i++) grid [i] = i
grid_data = (Object) XtVaCreateWidget("Grid",
                  (WidgetClass)xintDataGridObjectClass,
                  edit,
                  XmNgridArray, grid,
                  XmNxCount, NX,
                  XmNyCount, NY,
                  XmNgridOrder, XintX_VECTOR,
                  XmNdataGroup, data_group, NULL);
```

Inherited Behavior and Resources

The DataGrid object class does not inherit behavior and resources from any other object class:

- Class pointer is *xintDataGridObjectClass*
- Class name is *XintDataGrid*
- Header file is included as `<Xint/DataGrid.h>`

Resources The following resources are defined by the DataGrid object class:

Name	Type Default	Access
XmNcopyData	Boolean True	CSG
XmNdataGroup	Object NULL	CSG
XmNdataType	int XintDATA_TYPE_FLOAT	CSG
XmNeditable	Boolean True	CSG
XmNgridArray	XtPointer NULL	CSG
XmNgridOrder	int XintX_VECTOR	CSG
XmNlastViewDestroy	Boolean True	CSG
XmNupdateCallback	XtCallbackList NULL	CSG
XmNxCount	int 0	CSG
XmNyCount	int 0	CSG
XmNxRange	XintRange * NULL	CSG
XmNyRange	XintRange * NULL	CSG

XmNcopyData

If True, this resource will allocate memory to store a copy of the data, then free the memory allocation when the object is destroyed. If False, the widget does not create a copy of the data nor does it manage memory for this purpose.

XmNdataGroup

Specifies the object ID of the data group to which this data grid belongs. This resource must be specified if the data grid object is to be placed inside a data group.

XmNdataType

This resource specifies the data type for the array named in **XmNgridArray**. The resource value may be one of the following:

Resource Value	**Description**
XintDATA_TYPE_FLOAT (default)	Data type is float.
XintDATA_TYPE_SHORT	Data type is short.
XintDATA_TYPE_INTEGER	Data type is integer.
XintDATA_TYPE_LONG	Data type is long.
XintDATA_TYPE_DOUBLE	Data type is double.

XmNeditable

Specifies whether or not this data series can be edited when displayed in a view.

XmNgridArray

This resource specifies the name of the array containing the gridded data. The data type of the array can be defined using the **XmNdataType** resource.

XmNgridOrder

This resource specifies the direction in which the grid is ordered (see Figure 32). For example, in a table with Y orientation, the array values are arranged column-by-column; in X orientation they are arranged row-by-row. The selected X or Y orientation is maintained when the data is transferred to chart views and the data is plotted accordingly.

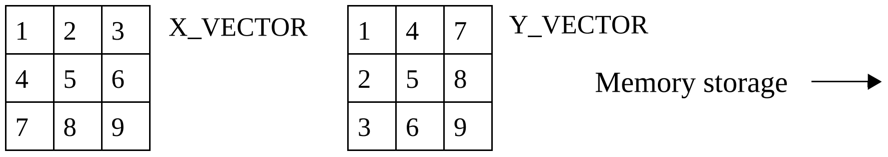


Figure 32. Grid Orientation

The following constants describe the grid orientation for the **XmNgridOrder** resource.

Resource Value	Description
XintX_VECTOR	Grid is ordered in the X direction.
XintY_VECTOR (default)	Grid is ordered in the Y direction.

XmNlastViewDestroy

Specifies whether or not to automatically destroy this data object when there are no more views connected to the data. If this data object is inserted into a DataGroup, you only need to set this resource for the DataGroup object.

XmNupdateCallback

This resource allows you to attach a callback to the Grid object that is invoked each time the data has changed. See the following section for more information on this callback.

XmNxCount

XmNyCount

These resources specify the size of the grid in the X and Y directions. In a table, X count is the number of rows, and Y count is the number of columns.

XmNxRange

XmNyRange

These resources let you define the grid data range in the X or Y direction as a start and an increment. If you don't specify a value for this resource, the starting value is set to be 0 and the increment 1.

The actual start/increment values are defined through the data structure XintRange, which takes the following form:

```
typedef struct {
   float start;
   float increment;
} XintRange;
```

where:

Member	Description
start	Starting value.
increment	Increment.

Callback

Callback XmNupdateCallback provides notification of changes in a grid object. This notification can be used to ensure that updates to one view are properly reflected in all other views. The callback structure takes the following form:

```
typedef struct {
int        reason;
XEvent      *event;
Object     object;
 int        x_start;
 int        y_start;
  int        x_count;
  int        y_count;
  float      minimum;
  float      maximum;
} XintDataGridUpdateCallbackStruct;
```

If you have multiple DataGrid or other objects in a group, this callback must be registered separately for each individual data object. If you prefer to control updates at the group level, you should register this callback only once for the DataGroup object and use the XintDataGroupUpdateCallbackStruct discussed earlier in this manual (see page 169 for details).

Structure XintDataGridUpdateCallbackStruct contains information that indicates why and how the object was modified. This information can help optimize the redrawing and updating of the appropriate views.

The following table lists the XintDataGridUpdateCallbackStruct members of structure:

Data Type	Member	Description
int	reason	Indicates why callback was invoked (see table below).
XEvent *	event	Standard member in Motif callback structure; not used in this context (will always be NULL).
Object	object	ID of the updated object.
int	x_start	Starting index of X value updated (starts at 0).
int	y_start	Starting index of Y value updated (starts at 0).
int	x_count	Total number of X values updated.
int	y_count	Total number of Y values being updated.
float	minimum	New minimum of updated values.
float	maximum	New maximum of updated values.

Member *reason* provides information explaining why and how the update occurred. Possible values for member *reason* are

Reason	Description
XintCR_DATA_UPDATE XintCR_DATA_BATCH	Entire data grid has been changed, no more information is available.
XintCR_DATA_REPLACE	A portion of the grid (as specified by *x_start*, *y_start*, *x_count* and *y_count*) has been replaced.

Functions

The following functions are available for creating and manipulating grid data objects.

Function Name	Description
XintCreateDataGrid	Creates a new data grid object.
XintDataGridGetGridArray	Retrieves any part of a grid.
XintDataGridDataReplace	Replaces data in a grid.

XintCreateDataGrid

Creates a new data grid object. The function argument list should contain any resource settings that you want to specify for the new grid object.

```
Object XintCreateDataGrid(...)
```

Widget	parent	Parent of new data grid object.
char *	name	Name of new data grid object.
ArgList	arglist	List of resources to be set for this data grid.
Cardinal	argcount	Total number of resources being set.

XintDataGridGetGridArray

Retrieves any part of a DataGrid object as a floating point array. You will have to free the array after it is no longer needed.

```
float *XintDataGridGetGridArray (...)
```

Object	object	Object ID of the data grid in which data is being retrieved.
int	x_start	Starting X value in the area of the grid to be retrieved.
int	y_start	Starting Y value in the area of the grid to be retrieved.
int	x_count	Total number of X values to be retrieved.
int	y_count	Total number of Y values to be retrieved.

XintDataGridDataReplace

Replaces the data within a specified area of a specified grid. The function arguments let you specify a pre-defined array to be inserted, along with the starting x,y values and the number of values to be replaced in both the X and Y directions. The function returns False if there is a bad argument or the function cannot be performed.

```
Boolean XintDataGridDataReplace (...)
```

Object	object	Object ID of the data grid in which data is being replaced.
XtPointer	array	Array containing the replacement data. The data type for this array should be the same as the type specified by XmNdataType.
int	x_start	Starting X value in the area of the grid being replaced.
int	y_start	Starting Y value in the area of the grid being replaced.
int	x_count	Total number of X values being replaced.
int	y_count	Total number of Y values being replaced.

Macros

Macro XintIsDataGrid returns True if the specified object is a DataGrid object.

```
Boolean XintIsDataGrid (Object object)
```

DataLabel Object

The DataLabel object can be used to define a label for the data in a group. The label object is an array of text strings that automatically appears in an appropriate context in any view of the group. For example, a label associated with a data series might contain a character string corresponding to each data pair in the series. These strings would appear as row annotation in a table view.

DataLabel objects are created automatically when the user performs a drag-and-drop. If the drag-and-drop begins inside an EditTable object, the column heading becomes the name of the data object, and the row annotation becomes a label array. For instance, if the user selects three columns for drag-and-drop, this will automatically create three DataSampled objects (one for each column), with a DataLabel object containing the annotation for each row.

For a Chart, DataLabel objects are used to annotate the axes. For a Table, DataLabel objects provide row or column annotation. The data label has an orientation which specifies which axis (when attached to a chart) it corresponds to. If you specify more than one DataLabel for a specific orientation, the chart manager will try to merge those labels based on the range information.

Example

The following example shows a typical code structure used to define a Data-Label object.

```
static String x_labels[] = {"Houston","Dallas","San Antonio",
                  "Austin"};
...
XtVaCreateWidget("Cities",
                  (WidgetClass)xintDataLabelObjectClass, edit,
                  XmNlabelStrings, x_labels,
                  XmNlabelCount, sizeof(x_labels)/sizeof(String),
                  XmNlabelOrientation, XintLABEL_X,
                  XmNdataGroup, data_group,
                  NULL);
```

Inherited Behavior and Resources

The DataLabel object class does not inherit behavior and resources from any other object class:

- Class pointer is *xintDataLabelObjectClass*
- Class name is *XintDataLabel*
- Header file is included as `<Xint/DataLabel.h>`

Resources

The following resources are defined by the DataLabel object class. These resources must be set as part of the object creation function XintCreateDataLabel discussed later in this section.

Name	Type Default	Access
XmNcopyData	Boolean True	CSG
XmNdataGroup	Object NULL	CSG
XmNlabelCount	int 0	CSG
XmNlabelOrientation	int XintLABEL_X	CSG
XmNlabelPositionArray	float * NULL	CSG
XmNlabelStrings	char ** NULL	CSG
XmNlastViewDestroy	Boolean True	CSG
XmNsampledRange	XintRange * NULL	CSG
XmNupdateCallback	XtCallbackList NULL	C

XmNcopyData

If True, allocates memory to store a copy of the data (label string array and labels), then free the memory when the object is destroyed. If False, widget does not create a copy of the data or manage memory for this purpose.

XmNdataGroup

Specifies the object ID of the data group that this label belongs to. This resource must be specified if the data label object is to be placed inside a group.

XmNlabelCount

This is the number of separate character strings contained in this DataLabel object.

XmNlabelOrientation

For a Chart, this resource specifies the axis this label is associated with. For a Table, this resource specifies whether the labels are used for row or column annotation. You can use one of the following constants for the value of this resource:

Resource Value	Description
XintLABEL_X (default)	Label is associated with the X axis (column annotation).
XintLABEL_Y	Label is associated with the Y axis (row annotation).
XintLABEL_Z	Label is associated with the Z axis.

XmNlabelPositionArray

This resource specifies an array, of size **XmNlabelCount**, used to position the labels along a chart axis. This resource should only be used if the increment between the labels is not constant. Use resource **XmNsampledRange** otherwise.

XmNlabelStrings

This is the text of the label, or the name of a pre-defined array containing a series of character strings to be used as labels.

XmNlastViewDestroy

Specifies whether or not to automatically destroy this data object when there are no more views connected to the data. If this data object is inserted into a DataGroup, you only need to set this resource for the DataGroup object.

XmNsampledRange

Specifies the start and increment for positioning the labels. For example, if the label orientation is specified as X, this resource would define the start (position of the first label) and the increment (how far apart are the labels) in the X direction. If this resource is not specified, the start is assumed to be 0 and the increment 1. This resource is helpful to position the annotation on the axis. Use resource **XmNlabelPositionArray** to position labels at a non constant interval.

This resource is specified as a pointer to a data structure of type XintRange, which takes the following form:

```
typedef struct {
   float start;
   float increment;
} XintRange;
```

where:

Member	Description
start	Minimum value.
increment	Increment.

XmNupdateCallback

Allows you to attach a callback to the Label object that is invoked each time the data has changed. See the following section for more information on this callback.

Callback

Callback XmNupdateCallback provides notification of changes in a DataLabel object. This notification can be used to ensure that updates to one view are properly reflected in all other views. The callback structure takes the following form:

```
typedef struct {
  int        reason;
  XEvent     *event;
  Object     object;
} XintDataLabelUpdateCallbackStruct;
```

If you have multiple DataLabel or other objects in a group, this callback must be registered separately for each individual data object. If you prefer to control updates at the group level, you should register this callback only once for the DataGroup object and use the XintDataGroupUpdateCallbackStruct discussed earlier in this manual (see page 169 for details).

The following ordered table describes the members of structure XintDataLabelUpdateCallbackStruct:

Data Type	Member	Description
int	reason	Indicates why callback was invoked (see table below).
XEvent *	event	Standard member in Motif callback structure; not used in this context (will always be NULL).
Object	object	ID of the updated object.

Member *reason* provides information explaining why and how the update occurred. Possible values for member *reason* are

Reason	Description
XintCR_DATA_UPDATE XintCR_DATA_BATCH	Label has been changed, no more information is available.

Functions

The following functions are defined for creating and manipulating DataLabel objects.

Function Name	Description
XintCreateDataLabel	Creates a DataLabel object.
XintDataLabelExtend	Extends the labels.
XintDataLabelReplace	Replaces the labels.
XintDataLabelShift	Shifts the labels.

XintCreateDataLabel

Creates a data label and define all resources for it.

```
Object XintCreateDataLabel (...)
```

Widget	parent	Parent of the new data label.
char *	name	Name of new data label object.
ArgList	arglist	List of resource settings for the new object.
Cardinal	argcount	Number of resources being set in arglist.

XintDataLabelExtend

Adds more labels to the end of a data label array. The function returns False if there is a bad argument, if the function cannot be performed, or if resource **XmNcopyData** is false.

```
Boolean XintDataLabelExtend (...)
```

Object	object	Object ID of the label array to be extended.
String *	string_array	Array of strings to be added to the original label array.
float *	pos_array	Array of positions to be added to the original position array (Specify NULL if you are not using this field).
int	count	Number of values being added.

XintDataLabelReplace

Replaces a range of labels in the label array. The function returns False if there is a bad argument.

```
Boolean XintDataLabelReplace(...)
```

Object	object	Object ID of the label array to be replaced.
String *	string_array	Array of new strings to be used to replace existing labels.
float *	pos_array	Array of positions to be used to replace existing positions (Specify NULL if you are not using this field).
int	start	Index of first value to be replaced (starts at 0).
int	count	Number of values being replaced.

XintDataLabelShift

Shifts the labels contained in a DataLabel object by discarding a specified number of values at the beginning of the existing array, and adding an equal number of values to the end of the existing array. The function returns False if argument count is bigger than the number of labels.

```
Boolean XintDataLabelShift(...)
```

Object	object	Object ID of the label array to be shifted.
String *	string_array	Array of new strings to be appended at the end of the list of labels.
float *	pos_array	Array of positions to be used to be appended at the end of the position list (Specify NULL if you are not using this field).
int	count	Number of values to be discarded at the front and appended to the end of the existing array.

Macros

Macro XintIsDataLabel returns True if the specified *object* is a DataLabel object.

```
Boolean XintIsDataLabel (Object object)
```

DataSampled Object Class

The DataSampled object class is used to manipulate a one-dimensional array of values. For instance, a DataSampled object may appear as a row or column of data in a table, a series of bars in a bar chart, a plotted line in a graph and so forth. The type of data contained in a DataSampled Object can be float, short, integer, long, or double. When the one-dimensional array is viewed in a two-dimensional context such as a chart, the second dimension (or implied direction) is generated from a start and an increment which are specified using resource **XmNrange**.

For example, Figure 33 shows a single-dimensioned array plotted on a chart in the default configuration. The X direction is implied from the data by counting the number of values (*n*) in the sample and creating a synthetic array of values based on a start and an increment (0 and 1 in this case):

static float d1[] = { 4.0, 5.0, 5.9, 7.3 };

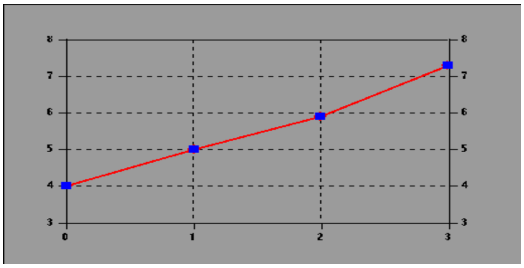


Figure 33. Example of DataSampled Array

Example

The following example code defines a typical DataSampled object:

```
static int population[] = {142000, 256000, 320000, 340000}
Object population_data;
XintRange range;
...
range.start = 1990;
range.increment =1;
population_growth = (Object) XtVaCreateWidget("population"),
                       (WidgetClass)
xintDataSampledObjectClass, parent,
                       XmNdataArray, population,
                       XmNdataType, XintDATA_TYPE_INTEGER,
                       XmNsampledRange, &range,
                       XmNdataGroup, data_group,
                       NULL);
```

Inherited Behavior and Resources

The DataSampled object class does not inherit behavior and resources from any other object class:

- Class pointer is *xintDataSampledObjectClass*
- Class name is *XintDataSampled*
- Header file is included as `<Xint/DataSampled.h>`

Resources The following resources can be defined for the DataSampled object class:

Name	Type Default	Access
XmNcopyData	Boolean True	CSG
XmNcount	int 0	CSG
XmNdataArray	XtPointer NULL	CSG
XmNdataGroup	Object NULL	CSG
XmNdataType	int XintDATA_TYPE_FLOAT	CSG
XmNeditable	Boolean True	CSG
XmNlastViewDestroy	Boolean True	CSG
XmNsampledRange	XintRange * NULL	CSG
XmNupdateCallback	XtCallbackList NULL	CSG

XmNcopyData

If True, allocates memory to store a copy of the data, then free the memory allocation when the object is destroyed. If False, the widget does not create a copy of the data or manage memory for this purpose.

XmNcount

Specifies number of elements in sampled data array named by **XmNdataArray**.

XmNdataArray

Specifies name of a pre-defined array containing the actual data. The array size is expressed through the resource **XmNcount** and the data type is specified by **XmNdataType**, both discussed later in this section. The default value of this resource is NULL.

XmNdataGroup

Specifies object ID of data group to which this sampled array belongs. Must be specified if the sampled data object is to be placed inside a group.

XmNdataType

Specifies the data type for the array named by **XmNdataArray**. You can specify one of the following constants for the value of this resource:

Resource Value	Description
XintDATA_TYPE_FLOAT	Data type is float (default).
XintDATA_TYPE_SHORT	Data type is short.
XintDATA_TYPE_INTEGER	Data type is integer.
XintDATA_TYPE_LONG	Data type is long.
XintDATA_TYPE_DOUBLE	Data type is double.

XmNeditable

Specifies whether or not this data series can be edited when displayed in a view.

XmNlastViewDestroy

Specifies whether or not to automatically destroy this data object when there are no more views connected to the data. If this data object is inserted into a DataGroup, you only need to set this resource for the DataGroup object.

XmNsampledRange

Specifies the start and increment used to generate the implied direction. This is particularly useful for axis scaling in a chart view. If this resource is not specified, the start is assumed to be 0 and the increment 1.

This resource is specified as a pointer to a data structure of type XintRange, which takes the following form:

```
typedef struct {
   float start;
   float increment;
} XintRange;
```

where:

Member	Description
start	Minimum value.
increment	Increment.

XmNupdateCallback

Allows you to attach a callback to the object, as described in the following section.

Callback for Data Updates

The callback XmNupdateCallback provides notification of changes in sampled data objects. As for any widget, you can register callbacks using the Xt function XtAddCallback. The callback structure takes the following form:

```
typedef struct {
  int               reason;
  XEvent            *event;
  Object            object;
  int               start;
  int               count;
  float             minimum;
  float             maximum;
} XintDataSampledUpdateCallbackStruct;
```

This callback is specific to the DataSampled object and must be registered for each separate data object in a group. If you prefer to monitor updates at the group level, you should register callback XmNupdateCallback only for the DataGroup object and use the XintDataGroupUpdateCallbackStruct discussed earlier in this manual (see page 169 for details).

The DataSampled update callback contains information that indicates why and how the object was modified. The following ordered table lists the members of XintDataSampledUpdateCallbackStruct in their required sequence:

Data Type	Member	Description
int	reason	Indicates why the callback was invoked (see table below).

Data Type	Member	Description
XEvent *	event	Standard member of a Motif callback structure; not used in this context (will always be NULL).
Object	object	ID of the updated object.
int	start	Starting index of the data updated (starts at 0).
int	count	Number of items updated.
float	minimum	New minimum value of the object.
float	maximum	New maximum value of the object.

The callback structure includes information explaining the reason why a particular update occurred.

Reason	Description
XintCR_DATA_UPDATE XintCR_DATA_BATCH	Entire data array has been changed, no more information is available.
XintCR_DATA_REPLACE	Data was replaced in the object. (as specified by *start* and *count*).
XintCR_DATA_EXTEND	Data was added to the object (*count* values were added).
XintCR_DATA_SHIFT	Data was shifted in the object (*count* values were shifted).

Functions

The following functions are defined for creating, updating or retrieving information regarding a particular sampled data object.

Function Name	Description
XintCreateDataSampled	Creates a DataSampled object.
XintDataSampledDataExtend	Extends the data in a sampled array.
XintDataSampledGetDataArray	Returns part of data array and converts it to float.
XintDataSampledGetSampledArray	Returns the sampled values.
XintDataSampledDataReplace	Replaces a range of values in a sampled array.
XintDataSampledDataShift	Shifts the data in a sampled array.

XintCreateDataSampled

Creates a data sampled object and defines all the resources for it.

```
Object XintCreateDataSampled (...)
```

Widget	parent	Parent of new data sampled object.
char *	name	Name of new data sampled.
ArgList	arglist	List of resource settings for the new data object.
Cardinal	argcount	Number of resources set by arglist.

XintDataSampledDataExtend

Adds more data to the end of a sampled array. The function returns False if there is a bad argument, if the function cannot be performed, or if CopyData is false.

```
Boolean XintDataSampledDataExtend (...)
```

Object	object	Object ID of the sampled array to be extended.
XtPointer	array	Array of data values to be added to the original sampled array. The data type for this array should be the same as the type specified by XmNdataType.
int	count	Number of values being added.

XintDataSampledGetDataArray

Returns any part of a specified data array as an array of float values. If the data is not already stored as float, it converts the data type to float. You will need to free the returned array after it is no longer needed.

```
float   *XintDataSampledGetDataArray (...)
```

Object	object	Object ID of the sampled array containing the desired data.
int	start	Index of first value to be returned (starts at 0).
int	count	Total number of values to be returned.

XintDataSampledGetSampledArray

Since a sampled data object contains only a single array of data values, the data along the implied direction is generated from the **XmNsampledRange** resource. This function lets you retrieve any part of the synthetic array by specifying the index of the first value and the total number of values (count) to be retrieved. Count cannot be more than the total number of values available, otherwise this function returns NULL. You will need to free the returned array after it is no longer needed.

```
float   *XintDataSampledGetSampledArray (...)
```

Object	object	ID of the sampled array.
int	start	Index of first value to be retrieved.
int	count	Number of values to be retrieved.

XintDataSampledDataReplace

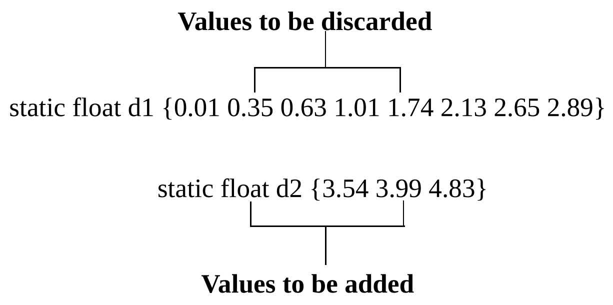
Replaces a range of values in a sampled array. The function returns False if there is a bad argument or the function cannot be performed.

```
Boolean XintDataSampledDataReplace (...)
```

Object	object	Object ID of the sampled array that requires values replaced.
XtPointer	array	Array of new values to be used to replace existing values. The data type for this array should be the same as the type specified by **XmNdataType**.
int	start	Index of first value to be replaced (starts at 0).
int	count	Number of values being replaced.

XintDataSampledDataShift

Shifts the data in the sampled array by discarding a certain number of values (count) at the beginning of the array and adding an equal number of values to the end of the array. The values to be added must be specified in a separate array, as shown in Figure 34:


Values to be discarded
static float d1 {0.01 0.35 0.63 1.01 1.74 2.13 2.65 2.89}
static float d2 {3.54 3.99 4.83}
Values to be added

Final array result: static float d3 {1.01 1.74 2.13 2.65 2.89 3.54 3.99 4.83}

Figure 34. Data Shift

The function returns False if there is a bad argument or the function cannot be performed.

```
Boolean XintDataSampledDataShift (...)
```

Object	object	Object ID of the array to be shifted
XtPointer	array	An array of values to be appended to the end of the current array in the DataSampled object. The data type for this array should be the same as the type specified by **XmNdataType**.
int	count	Number of values to be discarded at the front of the array and appended to the end of the array.

Macros

Macro XintIsDataSampled returns True if the specified *object* is a DataSampled object.

```
Boolean XintIsDataSampled (Object object)
```

DataSequentialSeries Object Class

The DataSequentialSeries class is identical to DataSeries except that it requires the X data values to be sequential (increasing). Using this class when appropriate results in significantly better display performance when dealing with large datasets or in the case of real-time applications.

Resources

The DataSequentialSeries object class inherits behavior and resources from the DataSeries class:

- Class pointer is *xintDataSequentialSeriesObjectClass*
- Class name is *XintDataSequentialSeries*
- Header file is included as `<Xint/DataSequentialSeries.h>`

Note: The DataSequentialSeries object class does not define any new resources. Refer to the *"DataSeries Object Class"* for a complete list of the resources of DataSequentialSeries.

DataSeries Object Class

The DataSeries object class is used to store, retrieve, and manipulate any series of (x,y) data pairs. Each data group may contain multiple DataSeries objects (for instance, multiple columns from a table or multiple lines on a chart). The series is specified as an array of X data points, with a matching array of Y data points. Compared to the DataSampled object, the DataSeries object is more suitable for describing random series of irregularly spaced data points. The type of data contained in a DataSeries Object can be float, short, integer, long, or double.

Example

The following example shows the code structure for a typical DataSeries object.

```
static float x_coord[] = {12.4, 23.5, 28.1, 31.5};
static float y_coord[] = {123, 432, 234, 121};
Object point_series;
...
point_series = (Object) XtVaCreateWidget("series",
              (WidgetClass)xintDataSeriesObjectClass, edit,
              XmNxArray, x_coord,
              XmNyArray, y_coord,
              XmNcount, sizeof(x_coord)/sizeof(float),
              XmNdataType, XintDATA_TYPE_FLOAT,
              XmNcopyData, False,
              NULL);
```

Inherited Behavior and Resources

The DataSeries object class does not inherit behavior and resources from any other object class:

- Class pointer is *xintDataSeriesObjectClass*
- Class name is *XintDataSeries*
- Header file is included as `<Xint/DataSeries.h>`

Resources

The following resources are defined by the DataSeries object class:

Name	**Type** **Default**	**Access**
XmNcopyData	Boolean True	CSG
XmNcount	int 0	CSG
XmNdataGroup	Object NULL	CSG
XmNdataType	int XintDATA_TYPE_FLOAT	CSG
XmNeditable	Boolean False	CSG
XmNlastViewDestroy	Boolean True	CSG
XmNupdateCallback	XtCallbackList NULL	CSG
XmNxArray	XtPointer NULL	CSG
XmNyArray	XtPointer NULL	CSG

XmNcopyData

If True, this resource will allocate memory to store a copy of the data, then free the memory allocation when the object is destroyed. If False, the widget does not create a copy of the data or manage memory for this purpose.

XmNcount

Specifies the number of data values in the X and Y data arrays that were specified by **XmNxArray** and **XmNyArray**.

XmNdataGroup

Specifies object ID of the data group to which this object belongs. Must be specified if the data series object is to be placed inside a group.

XmNdataType

Specifies the data type for the resources **XmNxArray** and **XmNyArray**. You can specify one of the following constants for the value of this resource:

Resource Value	Description
XintDATA_TYPE_FLOAT	Data type is float (default).
XintDATA_TYPE_SHORT	Data type is short.
XintDATA_TYPE_INTEGER	Data type is integer.
XintDATA_TYPE_LONG	Data type is long.
XintDATA_TYPE_DOUBLE	Data type is double.

XmNeditable

Specifies whether or not this data series can be edited when displayed in a view.

XmNlastViewDestroy

Specifies whether or not to automatically destroy this data object when there are no more views connected to the data. If this data object is inserted into a DataGroup, you only need to set this resource for the DataGroup object.

XmNupdateCallback

This resource allows you to attach a callback to the object, as described in the following section.

XmNxArray
XmNyArray

Specifies the names of the pre-defined X and Y data arrays containing the data in this series. The array size is expressed through the resource **XmNcount** and the data type is specified by **XmNdataType**, discussed later in this section. The default value of this resource is NULL.

Callbacks

Callback XmNupdateCallback can be used to provide notification of changes in a DataSeries object. Similarly as for widgets, you can register this callback with the Xt function XtAddCallback. When connected to a view (Chart or Table for example), this notification is used to ensure that updates to one view are properly reflected in all other applicable views.

The callback structure takes the following form:

```
typedef struct {
  int           reason;
  XEvent        *event;
  Object        object;
  int           x_start;
  int           x_count;
  int           y_start;
  int           y_count;
} XintDataSeriesUpdateCallbackStruct;
```

When used with DataSeries objects, this callback must be registered individually for each object. If you prefer to control updates at the group level, register the callback at the group level and use XintDataGroupUpdateCallbackStruct discussed earlier in this manual (see *"Callback Structure"* on page 169 for details).

The callback provides information indicating why and how the object was modified. The following ordered table describes the members of XintDataSeriesUpdateCallbackStruct:

Data Type	Member	Description
int	reason	Indicates why callback was invoked (see table below).
XEvent *	event	Standard member in Motif callback structure; not used in this context (will always be NULL).
Object	object	ID of the updated object.
int	x_start	Starting index of X values updated (starts at 0).
int	x_count	Total number of X values updated.
int	y_start	Starting index of Y values updated (starts at 0).
int	y_count	Total number of Y values updated.

The callback structure includes information explaining the reason why a particular update occurred. Header file `Data.h` contains the following reasons:

Reason	**Description**
XintCR_DATA_UPDATE XintCR_DATA_BATCH	Entire X and Y arrays have been changed, no more information is available.
XintCR_DATA_REPLACE	Data was replaced in the object, as specified by members *x_start*, *x_count*, *y_start* and *y_count*;
XintCR_DATA_EXTEND	Data added to object (*x_count* values where added).
XintCR_DATA_SHIFT	Data shifted in object (*x_count* values where shifted).

Defined Functions

The following functions are defined for creating and manipulating DataSeries objects:

Function Name	**Description**
XintCreateDataSeries	Creates a DataSeries object.
XintDataSeriesDataExtend	Adds extra data points to a data series.
XintDataSeriesDataReplace	Replaces a specified range of values in a data series.
XintDataSeriesDataShift	Shifts the data in a data series to a new range of values.
XintDataSeriesGetXArray	Returns part of X array and converts it to a float value.
XintDataSeriesGetYArray	Returns part of Y array and converts it to a float value.

XintCreateDataSeries

Creates a data series object and defines all the resources for it.

```
Object XintCreateDataSeries (...)
```

Widget	parent	Parent of new data series object.
char *	name	Name of new data series object.
ArgList	arglist	List of resource settings for the new data object.
Cardinal	argcount	Number of resources set by arglist.

XintDataSeriesDataExtend

Adds additional data to the end of a series array. The values to be added must be defined in separate X and Y arrays. Both arrays must contain an equal number of values representing matched (x,y) pairs. The function returns False if there is a bad argument or resource **XmNcopyData** is False.

```
Boolean XintDataSeriesDataExtend (...)
```

Object	object	Object ID of the data series array to be extended.
XtPointer	x_array	Array of X values to be added to the data series. The data type for this array should be the same as the type specified by **XmNdataType**.
XtPointer	y_array	Array of matching Y values to be added to the data series. The data type for this array should be the same as the type specified by **XmNdataType**.
int	count	Number of (x,y) pairs being added.

XintDataSeriesDataReplace

Replaces the range of values specified. The function returns False if there is a bad argument or if the function cannot be performed.

```
Boolean XintDataSeriesDataReplace (...)
```

Object	object	ID of the data series object in which data is being replaced.
XtPointer	x_array	Array containing X replacement values The data type for this array should be the same as the type specified by **XmNdataType**.
XtPointer	y_array	Array containing Y replacement values. The data type for this array should be the same as the type specified by **XmNdataType**
int	start	Index of first value to be replaced in the existing array.
int	count	Total number of (x,y) pairs being replaced.

XintDataSeriesDataShift

Shifts the data contained in a DataSeries object by discarding a specified number of values at the *beginning* of the existing arrays, and adding an equal number of values to the *end* of the existing arrays (see Figure 34 on page 195). For example, you may want to discard the first four points in a data series and add four new points to the end of the data series.

The values to be added must be defined in separate X and Y arrays. Both arrays must contain an equal number of values representing matched (x,y) pairs, and the total number of values in the new arrays must equal the number of values to be discarded from the existing arrays. The function returns False if argument count is incorrect.

```
Boolean XintDataSeriesDataShift (...)
```

Object	object	ID of the data series array.
XtPointer	x_array	Array of X values to be appended to the end of the current X array in the DataSeries object.
XtPointer	y_array	Array of Y values to append to end of current Y array in the DataSeries object. Data type for this array should be the same as the type specified by **XmNdataType**
int	count	Number of values to discard at front and append to end of existing array. The data type for this array should be the same as the type specified by **XmNdataType**

XintDataSeriesGetXArray
XintDataSeriesGetYArray

Retrieve any part of the X or Y array in an existing DataSeries object and returns it as an array of float values. If the data is not already classified as float, it converts the data type to float. You will need to free the returned array after it is no longer needed.

```
float *XintDataSeriesGetXArray (...)
float *XintDataSeriesGetYArray (...)
```

Object	object	ID of the DataSeries object containing the desired data.
int	start	Index of first value to be retrieved.
int	count	Number of values to be retrieved.

Macros

Macro XintIsDataSeries returns True if the specified *object* is a DataSeries object.

```
Boolean XintIsDataSeries (Object object)
```

DataTimeLabel Object Class

The DataTimeLabel class provides automatic time annotation for an axis object. The number of labels, label strings are generated based on the range of the axis and the axis increment.

Example

The following code shows the structure for a typical DataTimeLabel object:

```
XintRange range;
Object data_group;
Widget edit;
...
range.start = time((long *) 0L);
range.increment = 1;

/* Create a Date label object */
XtVaCreateWidget("Real Time",
                    (WidgetClass)xintDataTimeLabelObjectClass,
                    edit,
                    XmNtimeBase, time((long *) 0L),
                    XmNtimeMeasure, XintTIME_SECOND,
                    XmNtimeLabelFormat, "%H:%M:%S",
                    XmNsampledRange, &range,
                    XmNlabelOrientation, XintLABEL_X,
                    XmNdataGroup, data_group, NULL);
```

Inherited Behavior and Resources

DataTimeLabel object class inherits behavior and resources from *DataLabel* class:

- Class pointer is *xintDataTimeLabelObjectClass*
- Cass name is *XintDataTimeLabel*
- Header file is included as `<Xint/DataTimeLabel.h>`

Resources The following resources can be defined for the DataTimeLabel object class:

Name	Type Default	Access
XmNtimeBase	int 0	CSG
XmNtimeMeasure	int XintTIME_SECOND	CSG
XmNtimeLabelFormat	char * dynamic	CSG

XmNtimeBase

Specifies the relative time in seconds since Jan. 1st, 1970. See Unix function mktime to convert a specified date into a number of seconds that can be used as the time base. The time base is added to the limits of the axis when generating the time labels.

XmNtimeMeasure

Specifies the units used to measure time.

Name	Description
XintTIME_SECOND	Time is measured in seconds.
XintTIME_MINUTE	Time is measured in minutes.
XintTIME_HOUR	Time is measured in hours.
XintTIME_DAY	Time is measured in days.
XintTIME_MONTH	Time is measured in months.
XintTIME_YEAR	Time is measured in years.

XmNtimeLabelFormat

Specifies the format used to display the time label using the syntax of function strftime. A partial list of the format specifiers is given below. See man page strftime for a complete description of the syntax.

```
%a    locale’s abbreviated weekday name
%A    locale’s full weekday name
%b    locale’s abbreviated month name
%B    locale’s full month name
%d    day of month [1,31]; single digits are preceded by 0
%D    date as %m/%d/%y
%e    day of month [1,31]; single digits are preceded by a space
%h    locale’s abbreviated month name
%H    hour (24-hour clock) [0,23]; single digits are preceded by 0
%m    month number [1,12]; single digits are preceded by 0
%M    minute [00,59]; leading zero is permitted but not required
%n    insert a newline
%p    locale’s equivalent of either a.m. or p.m.
%y    year within century [00,99]
%Y    year, including the century (for example 1993)
```

Chart Object Reference

5

Overview

This chapter includes the following sections:

- **ChartObject Library** on page 206
- **Chart Object Class** on page 206
- **ChartWidget Widget Class** on page 223
- **AxisObject Object Class** on page 224
- **Legend Object Class** on page 231
- **Plot2D Object Metaclass** on page 235
- **Plot3D Object Metaclass** on page 241
- **CellArray Object Class** on page 247
- **ComboPlot Object Class** on page 250
- **BarLine Object Class** on page 253
- **Bar3D Object** on page 258
- **HighLow Object Class** on page 261
- **Histogram Object** on page 265
- **Pie Object Class** on page 269
- **Surface3D Object** on page 273
- **XYPlot Object Class** on page 278

ChartObject Library

This chapter is a reference for resources and public functions for the Chart object, as well as for all sub-objects that can be created by a Chart. Except for the Chart object itself, all other objects described in this chapter are created automatically based on the specified chart type and the type of data.

Object classes

The ChartObject library defines the following object classes:

Class	Description
Chart	Main component that creates and manages all objects necessary for building a chart.
ChartWidget	Convenience widget class that automatically create a Chart object inside an EditObject widget.
AxisObject	Provides axis annotation and coordinate transformation for 2D plot components.
Legend	Draws a legend.
Plot2D	Base class for all 2D plot classes.
Plot3D	Base class for all 3D plot classes.
CellArray	Plot class that displays its data as a 2D grid of colored cells.
ComboPlot	Special class used as a container for multiple plots.
BarLine	Plot class that displays its data as bars, stacked bars or lines.
Bar3D	Plot class that displays its data as 3D bars.
HighLow	Plot class that displays high-low-open-close graphs.
Histogram	Plot class that displays the distribution of a set of data.
Pie	Plot class that displays its data as 2D pies.
Surface3D	Plot class that displays its data as a 3D surface.
XYPlot	Plot class that displays its data as a 2D area, line or scatter plot.

Chart Object Class

ChartObject provides the *view* component of the MVC architecture described earlier. To create a chart, the application or the end-user needs to create a Chart object. This object automatically creates a number of sub-objects to compose the plot. For example, the chart may create axes objects, Text objects for annotation, a Legend object, a plot object, series objects, and so forth.

Inherited Behavior and Resources

Chart object class inherits behavior and resources from the *Xt Object*, *Graphic* and *Group objec*t classes:

- Class pointer is *xintChartObjectClass*
- Class name is *XintChart*
- Header file included as `<Xint/Chart.h>`

Resources

The Chart object class defines the following resources:

Name	**Type** **Default**	**Access**
XmNaxisSpacing	int 10	CSG
XmNchartFooter	String NULL	CSG
XmNchartMargins	XintChartMargins * {10., 10., 10., 10.}	CSG
XmNchartTitle	char * NULL	CSG
XmNchartType	int XintCHART_TYPE_BAR	CSG
XmNcolorList	String * {“red”, “green”, “blue”, “yellow”, “violet”, “cyan”, “salmon”, “PaleGreen2”, NULL}	CSG
XmNdoubleBuffer	Boolean True	CSG
XmNgeometry	XintGeometry * {15, 15, 85, 85}	CSG
XmNpropagate	Boolean False	CSG
XmNshowLegend	Boolean False	CSG
XmNsymbolCount	int dynamic	CSG
XmNsymbolList	int * dynamic	CSG
XmNtranspose	Boolean False	CSG
XmNzValueColorRecord	XintColorRec * NULL	CSG

XmNaxisSpacing

Specifies the space to leave between adjacent axes in the horizontal or vertical direction.

XmNchartFooter

Specifies a string that will be used as the Chart footer. If this resource is specified, the Chart object will create and manage a Text object. If you need to customize the appearance of the footer, use function XintChartGetComponent to extract the footer object ID. See constraint resource XmNconstraint to set the position of the footer object.

XmNchartMargins

Specifies the margins for the chart. The margins correspond to the space left between the chart boundaries and the axes (or plot if there is no axis on one side), as a percentage between 0 and 100. **XmNchartMargins** is specified as a pointer to a data structure of type XintChartMargins, which takes the following form:

```
typedef struct {
   int left;
   int right;
   int top;
   int bottom;
} XintChartMargins;
```

left	Left margin as a percentage between 0 and 100.
right	Right margin as a percentage between 0 and 100.
top	Top margin as a percentage between 0 and 100.
bottom	Bottom margin as a percentage between 0 and 100.

XmNchartTitle

Specifies a string that will be used as the Chart title. If this resource is specified, the Chart object will create and manage a Text object. To customize the appearance of the title, use function XintChartGetComponent to extract the title object ID. See constraint resource **XmNconstraint**, described in the next section, to set the position of the title object.

XmNchartType

Specifies the default chart type, as follows:

Constant	**Description**
XintCHART_TYPE_AREA	Area plot.
XintCHART_TYPE_BAR	2D Bar plot.
XintCHART_TYPE_BAR_3D	3D bars plot.
XintCHART_TYPE_CELL_ARRAY	2D array of colored cells.
XintCHART_TYPE_COMBINATION	Combination plot.
XintCHART_TYPE_HIGH_LOW	High-Low plot.
XintCHART_TYPE_HISTOGRAM	Histogram plot.
XintCHART_TYPE_LINE	Line plot.
XintCHART_TYPE_PIE	2D Pie plot.
XintCHART_TYPE_SCATTERED	Scattered plot.
XintCHART_TYPE_SURFACE_3D	3D Surface plot.

XmNcolorList

Specifies a NULL terminated list of color names (string array) used to assign the data series colors. The colors are assigned sequentially in the list. When the end of the list is reached, it is reset to the beginning. To change the color assigned to a specific series, use the function XintChartGetSeriesOfData and the appropriate resource (usually **XmNcolor** or **XmNfillColor**) to retrieve the series object ID.

XmNdoubleBuffer

Specifies whether or not double buffering is set. The default is True. For Plot3D objects this resource is ignored and double buffering is always set.

XmNgeometry

Specifies the geometry of the chart object in the parent coordinate system. **XmNgeometry** is specified as a pointer to a data structure of type XintGeometry, which takes the following form:

```
typedef struct {
   float x1;
   float y1;
   float x2;
   float y2;
} XintGeometry;
```

x1,y1	Coordinates of upper left corner of the chart.
x2,y2	Coordinates of lower right corner of the chart.

XmNpropagate

Specifies whether or not to propagate the resources set in a call to XtSetValues to the Chart's sub-objects (Plot, Axes, Legend, etc.). This resource is particularly useful to propagate resources controlling the edit mode such as **XmNsensitive**, **XmNmove** or **XmNshape**. It can also be used to propagate graphic attributes such as the background color or the font. This resource is automatically reset to False after it has been applied.

XmNshowLegend

Specifies whether a legend is displayed. If this resource is set to True, a Legend object is created. If you need to customize the legend, use function XintChartGetComponent to extract the ID of the Legend object.

XmNsymbolCount

Specifies the number of symbols defined in resource **XmNsymbolList**.

XmNsymbolList

Specifies a list of default symbols used in some plot types such as the scatter plot. The symbols in the list are used in turn (that is, the first series that requires a symbol will use the first symbol specified, the second series will used the second symbol, etc.). When the end of the list is reached, it is reset to the first symbol. The size of this array is specified using resource **XmNsymbolCount**.

The symbol list can contain any of the symbol constants listed in the following table:

Symbol Constant	Description
XintSYMBOL_X	X symbol.
XintSYMBOL_PLUS	+ symbol.
XintSYMBOL_SQUARE	Square symbol.
XintSYMBOL_CIRCLE	Circle symbol.
XintSYMBOL_TRIANGLE	Triangle symbol.
XintSYMBOL_DIAMOND	Diamond symbol.
XintSYMBOL_FILLED_SQUARE	Filled square symbol.
XintSYMBOL_FILLED_CIRCLE	Filled circle symbol.
XintSYMBOL_FILLED_TRIANGLE	Filled triangle symbol.
XintSYMBOL_FILLED_DIAMOND	Filled diamond symbol.

XmNtranspose

This resource controls whether or not to transpose DataSampled series. The transpose functionality is supported by several plot classes, including BarLine, Pie, XYPlot and all the 3D plot classes. See the Data section of each plot class for a description on how it handles transposed data.

XmNzValueColorRecord

Specifies a pointer to an opaque structure containing a list of color pixels that are used to display plot types that color their data series based on a Z value (or Y for 2D plots), including Surface3D, Bar3D and BarLine. If this resource is not specified, the colors specified using resource **XmNcolorList** will be used. See Chart function XintChartCreateColorRecord to create a color record structure from an array of pixel values.

Constraint Resources

The position of the Chart sub-objects is managed by the Chart. For example, the Chart normally will position the title above the Plot object and the legend to the right of the Plot object. Constraint resource **XmNconstraint** can be used to override the default placement of objects managed by the Chart, including any external objects inserted by the application. This resources does not apply to Plot objects, whose geometry is controlled by Chart resource **XmNchartMargins**. Resource **XmNconstraint** is specified using a pointer to a union of type XintConstraint which takes the following form:

```
typedef union  {
   struct {
      int type;
      int anchor;
   } placement;
} XintConstraint;
```

Member *type* should always be set to XintCONSTRAIN_PLACEMENT. For member *anchor*, use one of the following constants:

Defined Constants	Description
XintANCHOR_NORTH	Object placed above the Plot.
XintANCHOR_NORTH_EAST	Object placed next to upper right corner of Plot.
XintANCHOR_NORTH_WEST	Object placed next to upper left corner of Plot.
XintANCHOR_SOUTH	Object placed below the Plot.
XintANCHOR_SOUTH_EAST	Object placed next to lower right corner of Plot.
XintANCHOR_SOUTH_WEST	Object placed next to lower left corner of Plot.
XintANCHOR_EAST	Object placed right of the Plot.
XintANCHOR_WEST	Object placed left of the Plot.
XintANCHOR_CENTER	Object placed at the center inside the Plot.

Code

The following code sample shows how to position the title at the bottom of the plot.

```
Object chart;
Object title;
XintConstraint constraint;
...
title = XintChartGetComponent(chart,
                  XintCHART_COMPONENT_TITLE);
constraint.placement.type = XintCONSTRAIN_PLACEMENT;
constraint.placement.anchor = XintANCHOR_SOUTH;
XtVaSetValues((Object) title, XmNconstraint, &constraint,
                  NULL);
```

You can also use resource **XmNconstraint** to specify the placement of objects created by the application and inserted into the Chart using function XintChartInsertObject. If **XmNconstraint** is left NULL, the object will be positioned based on its own geometry resource, using the Chart coordinate system (0 to 100 in both directions with the origin located in the upper left corner).

Chart Callbacks

The following callbacks are defined by the Chart object class.

Name	Structure	Reason
XmNchartLayoutCallback	XintChartLayout-CallbackStruct	None
XmNplotLayoutCallback	XintPlotLayoutCall-backStruct	None
XmNverifyCallback	XintChartVerifyCall-backStruct	XintCR_OBJECT_MOVE XintCR_OBJECT_SHAPE XintCR_OBJECT_ADD_POINT XintCR_OBJECT_DELETE_POINT

XintChartLayoutCallbackStruct

The following ordered table lists the members of the callback structure, XintChartLayoutCallbackStruct associated with the callback XmNchartLayoutCallback.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
XintChartMargins *	chart_margins	The margins of the chart. The user should control it in the callback routine to maintain the aspect ratio he wants.
int	chart_height	Height of the chart (in pixels).
int	chart_width	Width of the chart (in pixels).
Boolean	doit	Set to False to prevent the chart margin from being recalculated.

XintPlotLayoutCallbackStruct

The following ordered table lists the members of the callback structure XintPlotLayoutCallbackStruct associated with the callback XmNplotLayoutCallback:

Data Type	Member	Description
int	reason	Indicates why the callback was invoked
XEvent *	event	Points to the XEvent that triggered the callback.
int	plot_height	Height of the plot (in pixels). [a]
int	plot_width	Width of the plot (in pixels).[a]
Boolean	doit	Set to False to prevent the plot dimensions from being recalculated.

a. The user should control it in the callback routine to maintain the aspect ratio he wants.

Functions

The following functions are defined for creating, updating or retrieving information regarding a particular Chart object.

Function Name	Description
XintCreateChart	Creates a Chart object.
XintChartAssociateData	Associates data with a chart.
XintChartCreateColorRecord	Creates a color record.
XintChartDisassociateAllData	Disassociates all data associated to a chart.
XintChartDisassociateData	Disassociates data from a chart.
XintChartFreezeUpdates	To freeze (or unfreeze) the chart while doing a series of changes.
XintChartGetComponent	Returns a particular component of the chart.
XintChartGetDataList	Returns the list of all the data groups that are connected to the specified Chart.
XintChartGetDataOfSeries	Returns the data object associated to the specified series in a chart.
XintChartGetSelectedComponent	Returns the selected component inside a chart.
XintChartGetSeriesOfData	Returns the object series of a Chart associated with the specified data element.
XintChartInitializeClassConverter	Initialized the ChartObject class converter.
XintChartInsertObject	To insert user defined object inside chart or plot.
XintChartIsTransposed	To tell if a Chart is transposed or not.
XintChartPick	To implement picking or data selection.
XintChartReadTemplate	Reads a list of Chart templates from a disk file.
XintChartSaveTemplate	Saves a list of Chart templates in a disk file.
XintChartZoom	To zoom a Chart object.

XintCreateChart

Creates a chart object and defines all the resources for it.

```
Object XintCreateChart (...)
```

Widget	parent	Parent of new chart object.
char *	name	Name of new chart object.
ArgList	arglist	List of resource settings for the new chart.
Cardinal	argcount	Number of resources set by arglist.

XintChartAssociateData

Associates a data group or a data item with a chart object. If you want multiple data group objects associated with a specified chart, you can call this function repeatedly.

```
void XintChartAssociateData (...)
```

Object	chart	ID of the chart object.
Object	data	ID of the data group or data object to associate with the chart.

XintChartCreateColorRecord

Creates a color record from an array of pixels that has been allocated by the application. A color record is a pointer to an opaque structure, XintColorRec, that can be used to specify Chart resource **XmNzValueColorRecord**.

```
XintColorRec *XintChartCreateColorRecord (...)
```

Pixel *	pixels	Array of pixels that has been allocated by the application.
int	pixel_count	Number of pixels in array *pixels*.

XintChartDisassociateAllData

Disassociates all data from a chart object.

```
void XintChartDisassociateAllData (Object chart)
```

XintChartDisassociateData

Disassociates a data group or a data item from a chart object. This function returns False if the data group or data item was not previously associated with the chart.

```
void XintChartDisassociateData (...)
```

Object	chart	ID of the chart object.
Object	data	ID of the data object or data group to disassociate from the chart.

XintChartFreezeUpdates

Freezes updates on a ChartObject before the application performs a series of changes on the Chart (argument *freeze* set to True). After the changes are made, the function should be called again with the argument *freeze* set to False to restore the Chart. If the argument *freeze* is set to True or False twice consecutively, the function will return False.

```
Boolean XintChartFreezeUpdates (...)
```

Object	chart	Name of the chart object to be frozen.
int	freeze	Controls whether to freeze or unfreeze the display of the Chart object. 1 - freeze 0 - unfreeze

XintChartGetComponent

This function returns the object ID of a specified component of the chart.

```
Object XintChartGetComponent (...)
```

Object	chart	ID of the chart object.
int	code	Code of the component to return.

The argument *code* can be one of the following constants:

Constant	Description
XintCHART_COMPONENT_PLOT	Code for the plot object.
XintCHART_COMPONENT_VERTICAL_AXIS	Code for the vertical axis object.
XintCHART_COMPONENT_HORIZONTAL_AXIS	Code for the horizontal axis object.
XintCHART_COMPONENT_LEGEND	Code for the legend object.
XintCHART_COMPONENT_TITLE	Code for the title object.
XintCHART_COMPONENT_FOOTER	Code for the footer object.

XintChartGetDataList

Returns the list of all the data groups or data items that have been associated with a particular Chart object using function XintChartAssociateData.

```
Object *XintChartGetDataList (...)
```

Object	chart	Chart object ID.
int *	count	Returns the data group count.

If the list returned is not NULL, you should free it after it is no longer needed.

XintChartGetDataOfSeries

Returns the list of data objects that are associated with the specified series component of a Chart. Most series represent one single data object component. Some series, like a HighLow series or a surface, represent more than one data object. Also, when a Chart is transposed (Chart resource **XmNtranspose** set to True) multiple series can be used to represent a data object.

```
Object *XintChartGetDataOfSeries (...)
```

Object	chart_or_plot	Chart or Plot object ID.
Object	series	ID of the series element.
int *	count	Returns the number of data objects associated with *series*.

You will need to free the object list returned after it is no longer needed.

XintChartGetSelectedComponent

When a chart or an object inside a chart is selected, the EditObject selection callback structure and function XintEditObjectSelectList always return the ID of the chart object. This convenience function can be used to retrieve the ID of the sub-component of the chart that was selected. It also returns a code indicating the type of the selected component.

```
Object XintChartGetSelectedComponent (...)
```

Object	chart	ID of the chart object.
int	*code	Code for the returned component.

The argument *code* will be one of the following constants:

Constant	**Description**
XintCHART_COMPONENT_CHART	Selected object is the chart.
XintCHART_COMPONENT_PLOT	Selected object is the plot.
XintCHART_COMPONENT_VERTICAL_AXIS	Selected object is the vertical axis.
XintCHART_COMPONENT_HORIZONTAL_AXIS	Selected object is the horizontal axis.
XintCHART_COMPONENT_LEGEND	Selected object is the legend object.
XintCHART_COMPONENT_TITLE	Selected object is the title object.
XintCHART_COMPONENT_FOOTER	Selected object is the footer object.
XintCHART_COMPONENT_SERIES	Selected object is a series.
XintCHART_COMPONENT_USER_OBJECT	Selected object is an object that was inserted using XintChartInsertObject.

XintChartGetSeriesOfData

Returns the list of object series used to represent a particular data element that is part of a Data Group associated with the chart. For most chart types, a single series object is used to represent a data object. For example, a DataSampled object is displayed using a BarSeries in a bar chart. If the Chart object is transposed (resource **XmNtranspose** set to True) and argument *data* is a DataSampled object, multiple series may be returned.

```
Object *XintChartGetSeriesOfData (...)
```

Object	chart_or_plot	Chart or Plot object ID.
Object	data	ID of the data element.
int *	count	Returns the number of series elements associated with *data*.

You will need to free the object list returned after it is no longer needed.

XintChartInitializeClassConverter

Installs the ChartObject class converter. This converter is used to convert the ascii object format and templates for the ChartObject from the string class name to the widget pointer class.

```
void XintChartInitializeClassConverter (void)
```

XintChartInsertObject

Allows the insertion of a graphic object, created by the application, inside a Chart or Plot2D object. User defined objects cannot be inserted inside a Plot3D object. If the object is inserted inside a Chart, its coordinate system is the Chart coordinate system which is between 0 and 100 in both the vertical and horizontal directions. If the object is inserted inside a Plot2D, the coordinate system is the one defined by the axes attached to the Plot2D object.

```
void XintChartInsertObject(...)
```

Object	chart_or_plot	ID of the Chart or the Plot in which to insert the object.
Object	object	ID of the object to be inserted.

XintChartIsTransposed

This convenience function returns True if resource **XmNtransposed** is True and False otherwise

```
Boolean XintChartIsTransposed (Object chart)
```

XintChartPick

Implements picking or data selection. Function XintChartPick converts an (X,Y) location inside a plot, specified as an event structure, into the corresponding data object and point index. If the location selected is outside the plot boundary or no data is selected, then the function returns NULL. Most applications requiring picking should use EditObject callback XmNselection Callback and call XintChartPick inside the callback routine. This function only works for 2D chart types.

```
Object XintChartPick(...)
```

Widget	edit_object	ID of the EditObject Widget.
XEvent	*event	Event to process (must be a button event of type ButtonPress or ButtonRelease).
int	*index	Returns the index of the data point selected.

XintChartReadTemplate

This function reads a template file that was created by the XintChartSaveTemplate function and restores the Chart or Charts that were saved in template form. Each Chart is associated with a data group by this function. However, the data are not required to be defined as formal DataGroup objects; any of the INT data object classes may be used. A list of the restored Charts is returned.

```
Object *XintChartReadTemplate (...)
```

Widget	parent	Widget ID of the parent of this Chart object.
char *	filename	Name of the file that contains the Chart templates.
Object *	data_group_list	List of data groups, one per Chart from the template file.
int	data_group_count	Number of data groups in the list.
int *	count	Returns the number of Chart objects read from disk and restored.

XintChartSaveTemplate

This function saves the listed Chart objects (and related objects) into a disk file. The Chart objects are saved in template format, meaning that the visual attributes of an object are saved, but the data associated with the object are not saved. This permits a Chart object to be read from disk by the XintChartReadTemplate function and used to display new data. Other, non-Chart, objects are entirely saved, including their data. All of the objects in the *object_list* must have the same parent.

```
Boolean XintChartSaveTemplate (...)
```

char *	filename	Name of the file where the Chart object templates will be stored.
Object *	object_list	List of Chart and non-Chart objects to be saved on disk in template format.
int	count	Number of objects in the list.

The function will return False if it cannot write to the specified file; otherwise it will return True.

XintChartZoom

This convenience function can be used to zoom a Chart object. The application must specify a rectangle in pixel coordinates defining the area to zoom. The Chart object viewport will be changed so that it displays only the intersection of the plot area with the specified rectangle. This function is normally called from a callback procedure attached to callback XmNareaSelectionCallback (see EditObject Reference section). The geometry of the selected rectangle is provided inside the callback structure. The function returns NULL and does not change the Chart object if the selected rectangle does not intersect with the plot area.

```
Boolean XintChartZoom (...)
```

Object	chart	ID of the Chart object to Zoom.
int	x,y	Pixel coordinates of the upper left corner of the area to zoom.
int	width, height	Size in pixels of the area to zoom.

Macros

Macro XintIsChart returns True if the specified *object* is a Chart object and False otherwise.

```
Boolean XintIsChart (Object object)
```

ChartWidget Widget Class

The ChartWidget class is a convenience widget class that automatically creates a Chart object inside an EditObject widget. The main purpose of this widget class is to be used inside GUI builders so that one can create an application requiring a chart interactively. This widget class can also be used directly to create a Chart object and an EditObject widget in one step.

Inherited Behavior and Resources

The ChartWidget class inherits behavior and resources from the *Core, Composite*, *Constraint*, *Manager*, *CompBase* and *EditObject* classes:

- Class pointer is *xintChartWidgetClass*
- Class name is *XintChartWidget*
- Header file is included as `<Xint/ChartW.h>`

The ChartWidget class does not define any new resources. You can however, specify any resources defined or inherited by the Chart object class. These resources will be applied directly to the Chart object created.

Functions

The following functions can be used to create a ChartWidget and to obtain the ID of the Chart object created by a ChartWidget widget.

XintChartWidgetGetObject

Returns the ID of the Chart Object created by a ChartWidget widget.

```
Object XintChartWidgetGetObject (Widget chart_widget)
```

where *chart_widget* is the ID of a ChartWidget widget.

XintCreateChartWidget

Creates an unmanaged ChartWidget widget.

```
Widget XintCreateChartWidget (...)
```

Widget	parent	Parent of new ChartWidget widget.
char *	name	Name of new ChartWidget widget.
ArgList	arglist	List of resource/value items.
Cardinal	argcount	Number of items in arglist.

AxisObject Object Class

A Chart object automatically creates several AxisObject objects. AxisObject objects are used both to provide annotation to the plot and to provide a coordinate system to map user coordinates to and from device coordinates.

The Chart object creates two additional sets of axes (two vertical axes and two horizontal axes) that are used to provide annotation for the plot and to define the user coordinate system. Function XintChartGetComponent lets you access the ID of one of the horizontal axes and one of the vertical axes. The axes in each set are synchronized, so when one is modified the other one is automatically updated. See Plot2D resources **XmNxAxisPlacement** and **XmNyAxisPlacement** to control which axes are visible.

The resources listed below can be used to customize the appearance of the axis objects.

Axis Limits and Increments

The limits and increments of the horizontal and vertical AxisObject associated with a 2D plot are controlled using Plot2D resources **XmNxLimits**, **XmNyLimits**, **XmNxIncrements** and **XmNyIncrements**. See section Plot2D below for more information on how to set those resources. If you don’t specify those resources, the axis will be automatically scaled based on the content of the data displayed in the Plot2D object.

Resources

The AxisObject object class inherits behavior and resources from the *Xt Object* and *Graphic* object classes:

- Class pointer is *xintAxisObjectClass*
- Class name is *XintAxisObject*
- Header file is included as `<Xint/AxisObject.h>`

Resources

The following resources are defined by the AxisObject object class:

Name	Type Default	Access
XmNannotationAngle	int 0	CSG
XmNannotationFont	String “*-Helvetica*-120-*”	CSG
XmNannotationFormat	String “%g”	CSG
XmNannotationPlacement	int XintANNOTATION_OUTSIDE	CSG
XmNaxisLineThickness	int 2	CSG
XmNendPoints	XintEndPoints * {10., 10., 90., 90.}	CSG
XmNincrements	XintIncrements * {20., 10.}	CSG
XmNlabel	String NULL	CSG
XmNlabelFont	String “*-Helvetica*-140-*”	CSG
XmNlimits	XintLimits * {0., 100.}	CSG
XmNlogScale	Boolean False	CSG
XmNmajorGridLineStyle	int XintLINE_NONE	CSG
XmNminimumLabelSpacing	int 0	CSG
XmNminorGridLineStyle	int XintLINE_NONE	CSG
XmNreverseOrder	Boolean False	CSG
XmNtickPlacement	int XintTICK_OUTSIDE	CSG

XmNannotationAngle

Specifies the angle of rotation in degrees for the axis annotation with respect to the axis base line. Best results are obtained for angle values between -45 and 90.

XmNannotationFont

Specifies the font used to draw the axis annotation.

XmNannotationFormat

Specifies the format used to draw the annotation. Use any C format descriptor (e.g. “%5.2f”) suitable for displaying floating point data.

XmNannotationPlacement

Specifies where to locate the annotation with respect to the axis line. You can specify one of the following constants:

Resource Value	Description
XintANNOTATION_NONE	No annotation is displayed.
XintANNOTATION_INSIDE	Annotation is displayed inside the plot area.
XintANNOTATION_OUTSIDE (default)	Annotation is displayed outside the plot area.
XintANNOTATION_CROSS	Annotation is displayed on both sides.

XmNaxisLineThickness

Specifies the axis line thickness in pixels.

XmNendPoints

Specifies the axis geometry which is defined by specifying the two end points of the axis bounding box in the parent coordinate system. This resource is specified as a pointer to a data structure of type XintEndPoints which takes the following form:

```
typedef struct {
    float x1, y1;
    float x2, y2;
} XintEndPoints;
```

This resource is automatically set for an Axis object that is used inside a Chart object.

XmNincrements

Specifies the major and minor increment for the axis annotation. This resource is specified as a pointer to a data structure of type XintIncrements which takes the following form:

```
typedef struct {
    float major_increment;
    float minor_increment;
} XintIncrements;
```

This resource should not be set directly for an Axis object created inside a Chart object. Use Plot2D resources **XmNxIncrements** and **XmNyIncrements** instead.

XmNlabel

Specifies a string that is used to draw the axis title label. Specify NULL if you don't want an axis title label.

XmNlabelFont

Specifies the font used to draw the axis title label.

XmNlimits

Specifies the limits for the axis widget. This resource is specified as a pointer to a data structure of type XintLimits which takes the following form:

```
typedef struct {
    float minimum;
    float maximum;
} XintLimits;
```

This resource should not be set directly for an Axis object created inside a Chart object. Use Plot2D resources **XmNxLimits** and **XmNyLimits** instead.

XmNlogScale

Specifies whether to use a linear scale (False) or a logarithmic scale (True). If you specify a logarithmic scale, make sure that the data range is strictly positive.

XmNmajorGridLineStyle
XmNminorGridLineStyle

Specifies how to draw the major (minor) grid lines associated with the AxisObject. For a vertical axis, these resources control the horizontal grid lines; for a horizontal axis, these resources control the vertical grid lines. You can specify one of the following constants:

Resource Value	Description
XintLINE_NONE (default)	No grid lines are drawn.
XintLINE_SOLID	Grid lines are drawn using a solid line.
XintLINE_ON_OFF_DASH	Grid lines are drawn using a on-off dash pattern
XintLINE_DOUBLE_DASH	Grid lines are drawn using a double dashed patterns.

XmNminimumLabelSpacing

Specifies the minimum distance, in pixel, between any adjacent label annotation.

XmNreverseOrder

Specifies whether to invert the axis or not. By default (False), the axis places the minimum value on the left (horizontal axis) or at the bottom (vertical axis). If you specify True, the axis will place the minimum value on the right (horizontal axis) or at the top (vertical axis).

XmNtickPlacement

Specifies where to locate the major and minor tick marks with respect to the axis line. You can specify one of the following constants:

Resource Value	Description
XintTICK_NONE	No tick marks are displayed.
XintTICK_INSIDE	Tick marks are displayed inside the plot area.
XintTICK_OUTSIDE (default)	Tick marks are displayed outside the plot area.
XintTICK_CROSS	Tick marks are displayed on both sides.

Inherited Resources

AxisObject inherits behavior and resources from the Graphic class. The set of resources that can be accessed along with the proper defaults is listed below.

Name	Type Default	Access
XmNcolor	Pixel foreground	CSG
XmNdashList	char * NULL	CSG
XmNfont	char * "Helvetica*120*"	CSG
XmNhighlightMode	int XintHIGHLIGHT_HANDLE	CSG
XmNlineStyle	int XintLINE_SOLID	CSG
XmNlineThickness	int 1	CSG
XmNmove	Boolean True	CSG
XmNsensitive	Boolean True	CSG
XmNshape	Boolean True	CSG
XmNstippleColor	Pixel foreground	CSG

AxisObject Callbacks

The following callback is defined by the AxisObject object class.

Name	Structure	Reason
XmNcolorLabelCallback	XintAxisObjectColorLabelCallbackStruct	None

XintAxisObjectColorLabelCallbackStruct

The following ordered table lists the members of the callback structure, XintAxisObjectColorLabelCallbackStruct associated with the callback XmNcolorLabelCallback.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the XEvent that triggered the callback.
int	index	Index of the label for which the color is being changed.
float	position	Position of the label.
String	label	The character string content of the label.
Pixel	color	The color of the label (can be changed).
Object	object	The object id of the AxisObject.
int	doit	Set it to False to prevent the color from being changed.

Functions

The following functions can be used to do conversion between pixel coordinates and axis object coordinates.

XintAxisObjectUserToPixel

Converts a user coordinate expressed in the Axis object coordinate system into a pixel location. The function returns False if the coordinate is outside the axis range.

```
Boolean XintAxisObjectUserToPixel (...)
```

Object	axis	ID of the Axis object.
double	user_coordinate	User coordinate to convert to a pixel location.
int *	pixel	Returns the pixel location corresponding to *user_coordinate*.

XintAxisObjectPixelToUser

Converts a pixel location into a user coordinate expressed in the Axis object coordinate system. The function returns False if the coordinate returned is outside the axis range.

```
Boolean XintAxisObjectPixelToUser (...)
```

Object	axis	ID of the Axis object.
int	pixel	Pixel coordinate to convert.
float *	user_coordinate	Returns the user coordinate corresponding to *pixel*.

Macros

The XintIsAxisObject macro returns True if the specified *object* is an AxisObject object. It returns False otherwise.

```
Boolean XintIsAxisObject (Object object)
```

Legend Object Class

A Chart Object creates a Legend object when Chart resource **XmNshowLegend** is set to True. The following section describes the resources of the Legend object that you can use to customize its appearance. The labels used to annotate the legend correspond to the name of each series data object. Figure 35 shows a bar chart with a legend.

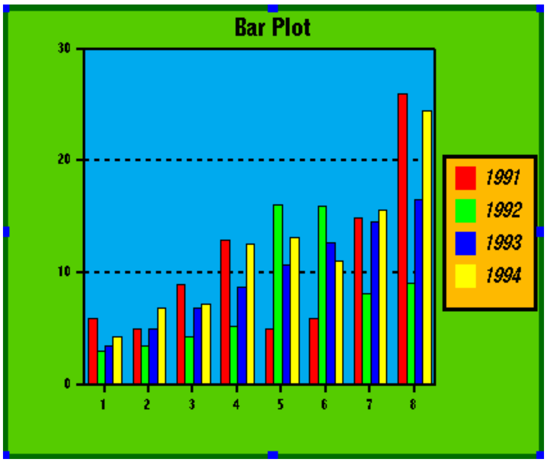


Figure 35. Plot With a Legend

Inherited Behavior and Resources

The Legend object class inherits behavior and resources from the *Xt Object* and *Graphic* object classes:

- Class pointer is *xintLegendObjectClass*
- Class name is *XintLegend*
- Header file is included as `<Xint/Legend.h>`

Resources

The following resources are defined by the Legend object class:

Name	Type Default	Access
XmNiconWidth	int 15	CSG
XmNcolumns	int 1	CSG
XmNlegendLocation	XintLocation * NULL	CSG
XmNlegendTitle	String NULL	CSG
XmNlineStyle	int XintLINE_NONE	CSG
XmNmarginHeight	int 8	CSG
XmNmarginWidth	int 8	CSG
XmNvisibleEntryCount	int 0	CSG

XmNiconWidth

Specifies the size in pixels of the icon used to draw the series symbol.

XmNcolumns

Specifies the number of columns to use for the legend layout.

XmNlegendLocation

This resource is used to specify the position of the legend inside the chart. Do not use this resource directly but rather use the Chart constraint resource **XmNconstraint** to move the legend elsewhere. This resource is used to get values.

This resource is specified as a pointer to a data structure of type XintLocation which takes the following form:

```
typedef struct {
   float x;
   float y;
} XintLocation;
```

member	Description
x, y	Specify the location of the center of the legend in the chart coordinate system (0 to 100 in both X and Y directions).

XmNlegendTitle

Specifies a title for the legend.

XmNlineStyle

Controls the line style for the rectangle that is drawn around the legend. The line color and line thickness are controlled by the Graphic resources **XmNcolor** and **XmNlineThickness**. You can specify one of the following constants for **XmNlineStyle**:

Resource Value	Description
XintLINE_NONE (default)	No rectangle is drawn.
XintLINE_SOLID	The rectangle is drawn with a solid line.
XintLINE_ON_OFF_DASH	The rectangle is drawn with an on-off dash pattern.
XintLINE_DOUBLE_DASH	The rectangle is drawn with a double dash pattern.
XintSHADOW_IN	Shadow is drawn so that legend appears inset.
XintSHADOW_OUT	Shadow is drawn so that legend appears outset.

XmNmarginHeight

Specifies the vertical margin for the legend in pixels.

XmNmarginWidth

Specifies the horizontal margin for the legend in pixels.

XmNvisibleEntryCount

Specifies the maximum number of entries that can be displayed by the legend. Specify 0 to have no limits.

Resources

Legend inherits behavior and resources from the Graphic class.The following table lists the resources that can be accessed:

Name	Type Default	Access
XmNcolor	Pixel foreground	CSG
XmNdashList	char * NULL	CSG
XmNfillColor	Pixel background	CSG
XmNfillFilename	char * NULL	CSG
XmNfillPixmap	Pixmap NULL	CSG
XmNfillStyle	int XintFILL_NONE	CSG
XmNfont	char * “*Helvetica*-120-*”	CSG
XmNhighlightMode	int XintHIGHLIGHT_HANDLE	CSG
XmNlineStyle	int XintLINE_SOLID	CSG
XmNlineThickness	int 1	CSG
XmNmove	Boolean True	CSG
XmNsensitive	Boolean True	CSG
XmNshape	Boolean True	CSG
XmNstippleColor	Pixel foreground	CSG

Macros

The XintIsLegend macro returns True if the specified *object* is a Legend object. Otherwise, returns False otherwise.

```
Boolean XintIsLegend (Object object)
```

Plot2D Object Metaclass

Plot2D is the base class for all the 2D plot objects that can be created by a Chart object. This class is a metaclass and it will never be instantiated. Its only purpose is to define the set of resources common to all 2D plot objects.

Inherited Behavior and Resources

The Plot2D object class inherits behavior and resources from the *Xt Object*, *Graphic* and *Group*.

Object class resources

The following resources are defined by the Plot2D object class:

Name	Type Default	Access
XmNdrawFrame	Boolean True	CSG
XmNpropagate	Boolean False	CSG
XmNxAutoRangeMode	int XintROUND_MIN_MAX	CSG
XmNxAxisPlacement	int XintPLACEMENT_BOTTOM	CSG
XmNxIncrements	XintIncrements * NULL	CSG
XmNxInsidePlacement	float * NULL	CSG
XmNxLimits	XintLimits * NULL	CSG
XmNyAutoRangeMode	int XintROUND_MIN_MAX	CSG
XmNyAxisPlacement	int XintPLACEMENT_LEFT	CSG
XmNyIncrements	XintIncrements * NULL	CSG
XmNyInsidePlacement	float * NULL	CSG
XmNyLimits	XintLimits * NULL	CSG

XmNdrawFrame

Specifies whether or not to draw a frame around the plot area. The thickness of the frame is set to be the maximum of the horizontal and vertical thickness. The frame color is set to the color of the axis, horizontal or vertical, that was last set.

XmNpropagate

Specifies whether or not to propagate the resources set in a call to XtSetValues to the Plot2D's sub-objects (series object). This resource is particularly useful to propagate resources controlling the edit mode such as **XmNsensitive**, **XmNmove** or **XmNshape**. For example, to disable editing on all the series objects created inside a plot, apply resources **XmNpropagate** (set to True) and **XmNshape** (set to False) for the Plot object. Resource **XmNpropagate** is reset to False automatically after being used.

XmNxAutoRangeMode

When resource **XmNxLimits** is NULL, the auto range is activated for the horizontal axis. When auto range is active, the limits of the axis are calculated automatically based on the data range in the axis direction. Resource **XmNxAutoRangeMode** specifies how those limits are calculated. You can specify one of the following constants:

Constant	Description
XintROUND_MIN_MAX (default)	Axis end values obtained by rounding off minimum and maximum data values to multiples of major increment.
XintUSE_MIN_MAX	Axis end values set to minimum and maximum data values.

XmNxAxisPlacement

Specifies where to locate the X axis, according to the following constants:

Resource Value	Description
XintPLACEMENT_TOP	Axis is placed at top of plot area.
XintPLACEMENT_BOTTOM (default)	Axis is placed at bottom of plot area.
XintPLACEMENT_NONE	No axis is displayed.
XintPLACEMENT_TOP_BOTTOM	Axis placed at top and bottom of plot area.
XintPLACEMENT_INSIDE	Axis placed inside plot area at Y location specified by resource **XmNxInsidePlacement**

XmN[xy]Increments

Specifies the major and minor increment for the axis (horizontal or vertical) associated with the Plot2D object. If you don't specify this resource, the axis major and minor increments are calculated automatically based on the range of the data displayed in the plot.

These resources are specified as a pointer to a data structure of type XintIncrements which takes the following form:

```
typedef struct {
   float major_increment;
   float minor_increment;
} XintIncrements;
```

XmN[xy]InsidePlacement

Specifies the location of the X (Y) axis inside the Plot object if resource **XmNxAxisPlacement** (**XmNyaxisPlacement**) is set to XintPLACEMENT_INSIDE. The location should be specified using the coordinate system of the other direction, Y (X).

XmN[xy]Limits

Specifies the limits for the axis (horizontal or vertical) associated with the Plot2D object. If you don't specify these resources, the limits of the horizontal axis and vertical axis associated with a Plot2D are automatically calculated based on the range of the data displayed in the plot. This resource can be used to overwrite the default limits or to zoom in on a portion of the plot. If you only want the auto-range option to apply to the minimum or the maximum but not both, specify constant XintUNDEFINED_FLOAT for the field which you want to be calculated automatically.

These resources are specified as a pointer to a data structure of type XintLimits which takes the following form:

```
typedef struct {
   float minimum;
   float maximum;
} XintLimits;
```

XmNyAutoRangeMode

When resource **XmNyLimits** is NULL, the auto range is activated for the vertical axis. When auto range is active, the limits of the axis are calculated automatically based on the data range in the axis direction. Resource **XmNyAutoRangeMode** specifies how those limits are calculated. You can specify one of the following constants:

Constant	Description
XintROUND_MIN_MAX (default)	Axis end values are obtained by rounding off minimum and maximum data values to multiples of the major increment.
XintUSE_MIN_MAX	Axis end values are set to the minimum and maximum data values.

XmNyAxisPlacement

Specifies where to locate the Y axis. You can use one of the following constants:

Constant	Description
XintPLACEMENT_LEFT (default)	Axis is placed to the left of the plot area.
XintPLACEMENT_RIGHT	Axis is placed to the right of the plot area.
XintPLACEMENT_NONE	No axis is displayed.
XintPLACEMENT_LEFT_RIGHT	Axis is placed both left and right of the plot area.
XintPLACEMENT_INSIDE	Axis is placed inside the plot area at X location specified by resource **XmNyInsidePlacement**

Graphic class resources

Plot2D inherits behavior and resources from the Graphic class. The following table lists the rsources that can be accessed:

Name	Type Default	Access
XmNcolor	Pixel foreground	CSG
XmNdashList	char * NULL	CSG
XmNfillColor	Pixel “gray”	CSG
XmNfillFilename	char * NULL	CSG
XmNfillPixmap	Pixmap NULL	CSG
XmNfillStyle	int XintFILL_SOLID	CSG
XmNfont	char * “*Helvetica*-120-*”	CSG
XmNhighlightMode	int XintHIGHLIGHT_HANDLE	CSG
XmNlineStyle	int XintLINE_NONE	CSG
XmNlineThickness	int 1	CSG
XmNmove	Boolean True	CSG
XmNsensitive	Boolean True	CSG
XmNshape	Boolean True	CSG
XmNstippleColor	Pixel foreground	CSG

Functions

The following function is defined to insert objects into plots.

XintPlotInsertObject

Inserts a graphic object created by the application inside a Plot2D object. The function performs an optimized insertion which is faster and smoother than the Chart convenience function XintChartInsertObject. The inserted object's coordinate system is the one defined by the axis attached to the Plot2D object.

```
    Boolean XintPlotInsertObject (...)
```

Object	chart or plot	Name of the Chart or the Plot in which to insert the object.
Object	child	Name of the child object to be inserted.

Plot3D Object Metaclass

Plot3D is the base class for all the 3D plot objects that can be created by a Chart object. This class is a metaclass and it will never be instantiated. Its only purpose is to define the set of resources common to all 3D plot objects.

Rotation, Translation and Scaling

Objects based on the Plot3D class can be rotation, scaled and translated interactively. The EditObject widget class supports a set of three actions, Transform3DStart, Transform3D and Transform3DEnd that are used to specify how to interact with 3D objects. The default translation associated with those actions is as followed:

Event Sequence	Action	Description
Ctrl <Btn3Down>	Transform3DStart(scale)	Start scaling 3D object.
Shift <Btn3Down>	Transform3DStart(shift)	Start translating 3D object.
None <Btn3Down>	Transform3DStart(rotate)	Start rotating 3D object.
<Btn3Motion>	Transform3D()	Execute transformation while mouse is moving.
<Btn3Up>	Transform3DEnd()	End transformation and redraw object.

Refer to *“EditObject Widget Class”* on page 81 for a complete description of those actions.

Inherited Behavior and Resources

The Plot3D object class inherits behavior and resources from the *Xt Object*, *Graphic* and *Group*.

Object class resources

The following resources are defined by the Plot3D object class:

Name	Type Default	Access
XmN3dPerspectiveDepth	int 0	CSG
XmN3dRotation	Xint3DRotation * {45., 0., 45.}	CS
XmN3dScale	Xint3DScale {1., 1., 1.}	CSG
XmNannotationStrokeFont	int XintSIMPLEX_ROMAN	CSG
XmNannotationStrokeFont-Size	int 40	CSG
XmNlabelStrokeFont	int XintCOMPLEX_ROMAN	CSG
XmNlabelStrokeFontSize	int 50	CSG
XmNviewScale	int 100	CSG
XmNxAnnotation	Boolean True	CSG
XmNyAnnotation	Boolean True	CSG
XmNzAnnotation	Boolean True	CSG
XmNxAnnotationFormat	char * “%.0f”	CSG
XmNyAnnotationFormat	char * “%.0f”	CSG
XmNzAnnotationFormat	char * “%.0f”	CSG
XmNxAxisOffset	int 0	CSG

Name (continued)	**Type** **Default**	**Access**
XmNyAxisOffset	int 0	CSG
XmNzAxisOffset	int 0	CSG
XmNxGridLines	int XintXY_PLANE+XintXZ_PLANE	CSG
XmNyGridLines	int XintXY_PLANE+XintYZ_PLANE	CSG
XmNzGridLines	int XintXZ_PLANE+XintYZ_PLANE	CSG
XmNxIncrements	XintIncrements * NULL	CSG
XmNyIncrements	XintIncrements * NULL	CSG
XmNzIncrements	XintIncrements * NULL	CSG
XmNxLabel	String NULL	CSG
XmNyLabel	String NULL	CSG
XmNzLabel	String NULL	CSG
XmNxLimits	XintLimits * NULL	CSG
XmNyLimits	XintLimits * NULL	CSG
XmNzLimits	XintLimits * NULL	CSG
XmNxTranslation	int 0	CSG
XmNyTranslation	int 0	CSG
XmNwallColor	Pixel “light grey”	CSG

XmN3dPerspectiveDepth

Specifies the distance from the eye to the object as a percentage of the object size. The distance must be a positive integer greater or equal to 200, which corresponds to a distance equal to twice the object size. You can also specify 0 to obtain a parallel projection (distance from object is infinite).

XmN3dRotation

Specifies the clockwise rotation in degrees along the X, Y and Z directions. This resource is specified as a pointer to a data structure of type Xint3DRotation which takes the following form:

```
typedef struct {
   float x_angle;
   float y_angle;
   float z_angle;
} Xint3DRotation;
```

XmN3dScale

Specifies the scale to apply along the X, Y and Z directions. This resource is specified as a pointer to a data structure of type Xint3DScale which takes the following form:

```
typedef struct {
   float x_scale;
   float y_scale;
   float z_scale;
} Xint3DScale;
```

XmNannotationStrokeFont

Specifies the font used to draw the annotation. You can specify one of the following constants:

Resource Value	Description
XintHERSHEY_SIMPLEX	Use a simplex stroke font.
XintHERSHEY_TRIPLEX	Use a triplex stroke font.

XmNannotationStrokeFontSize

Specifies the size of the font used to draw the axis annotation. The size is measured in thousandths of the unit cube size.

XmNlabelStrokeFont

Specifies the font used to draw the axis labels. You can specify one of the following constants:

Resource Value	Description
XintHERSHEY_SIMPLEX	Use a simplex stroke font.
XintHERSHEY_TRIPLEX	Use a triplex stroke font.

XmNlabelStrokeFontSize

Specifies the size of the font used to draw the axis labels. The size is measured in thousandths of the unit cube size.

XmNviewScale

Specifies the scale to apply to the 3D plot as a percentage of the initial size. You must specify a positive integer value.

XmN[xyz]Annotation

Specifies whether or not to display the axis annotation.

XmN[xyz]AnnotationFormat

Specifies the format used to draw the annotation. Use any C format descriptor (e.g. “%2.5f”) suitable for displaying floating point data. Resource **XmNzAnnotationFormat** is also used to specify the legend annotation format for the Surface3D subclass.

XmN[xyz]AxisOffset

Specifies an offset corresponding to a space to leave on either side of the axis in the specified direction. The offset size is measured in thousandths of the unit cube size.

XmN[xyz]GridLines

Specifies which grid lines to draw. You can specify one or a combination of 2 (by adding them together) of the following constants:

Resource Value	Description
XintXY_PLANE	Draw grid lines in the XY plane.
XintXZ_PLANE	Draw grid lines in the XZ plane.
XintYZ_PLANE	Draw grid lines in the YZ plane.

For example, to have all X grid lines drawn, specify XintXY_PLANE+XintXZ_PLANE for resource **XmNxGridLines**.

XmN[xyz]Increments

Specifies the major and minor increment for the axis along the X (Y or Z) direction. If you don't specify this resource, the axis major and minor increment are calculated automatically based on the range of the data.

These resources are specified as a pointer to a data structure of type XintIncrements which takes the following form:

```
typedef struct {
   float major_increment;
   float minor_increment;
} XintIncrements;
```

XmN[xyz]Label

Specifies the X (Y or Z) label displayed next to the corresponding axis. Specify NULL to have no label displayed.

XmN[xyz]Limits

Specifies the axis range along the specified direction. If you don't specify anything for this value, the axis range will be set to the minimum and maximum in the specified direction. This resource is specified as a pointer to a data structure of type XintLimits which takes the following form:

```
typedef struct {
   float minimum;
   float maximum;
} XintLimits;
```

XmN[xy]Translation

Specifies the translation in the X (Y) direction in pixel units.

XmNwallColor

Specifies the color, as a pixel value, used to draw the wall behind 3D plots such as Surface or Bar3D.

CellArray Object Class

A ChartObject creates a Plot object of type CellArray when the chart type resource is set to XintCHART_TYPE_CELL_ARRAY, as shown in Figure 36. The CellArray object displays data as an array of colored cells, with colors specified as a list of color pixels using Chart resource **XmNzValueColorRecord**. If this resource is NULL, the colors specified using Chart resource XmNcolorList are used. Colors are assigned to each cell according to the cell value, using a linear interpolation scheme based on the data minimum and maximum values.

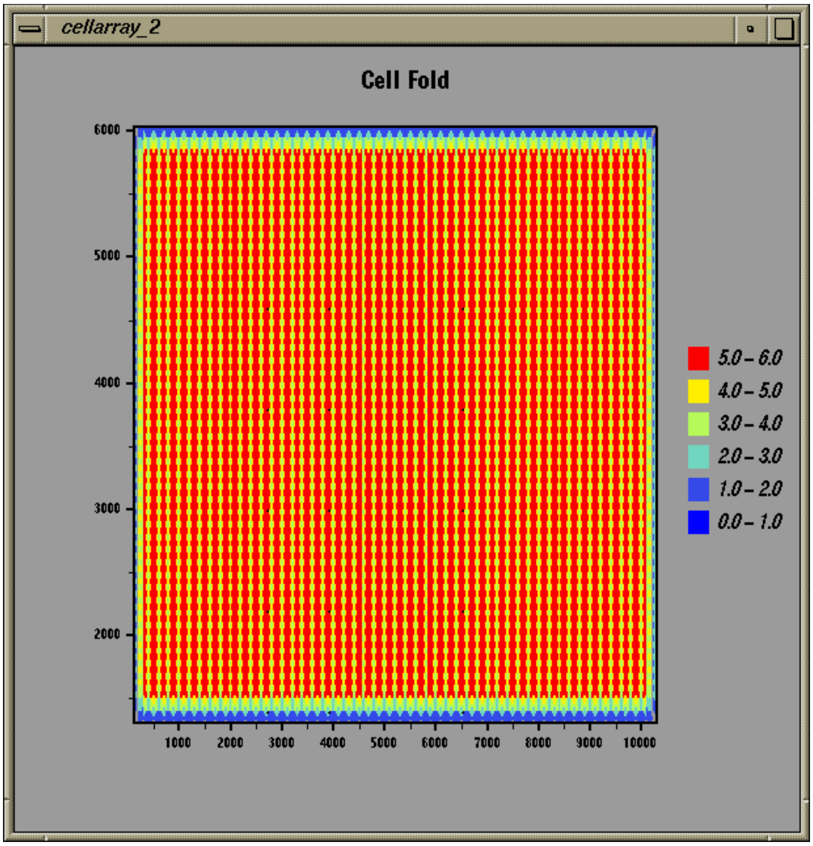


Figure 36. CellArray Chart Type

Data

The CellArray class handles data objects of type DataGrid or DataSampled. It first searches for a DataGrid object, if none is found, it searches for DataSampled objects. All the DataSampled objects should have the same increment, but can have different starting values (see resource **XmNsampledRange** to specify a non default start and increment). Missing and undefined values in the data are represented as a hole.

Inherited Behavior and Resources

The CellArray object class inherits behavior and resources from the *Xt Object*, *Graphic*, *Group* and *Plot2D* object classes:

- Class pointer is *xintCellArrayObjectClass*
- Class name is *XintCellArray*
- Header file is included as `<Xint/CellArray.h>`

Resources

The following resources are defined by the CellArray object class:

Name	Type Default	Access
XmNdisplayPatterns	Boolean True	CSG
XmNlegendAnnotationFormat	char * “%.0f”	CSG
XmNpatternList	Pixmap * NULL	CSG

XmNdisplayPatterns

Specifies whether or not to display patterns defined with **XmNpatternList**.

XmNlegendAnnotationFormat

Specifies format used to print the data values in the legend. Use any C format descriptor suitable for displaying floating point data.

XmNpatternList

Specifies list of bitmaps (pixmaps of depth 1) used as patterns when drawing each cell content. The list should be terminated using constant XmUNSPECIFIED_PIXMAP. Each pattern is assigned to a color in the colormap used by the CellArray in the order specified (first pattern is associated with first color, etc.).To assign a pattern to each cell, specify as many patterns as there are colors in the colormap.

Inherited Resources

Refer to *"Plot2D Object Metaclass"* on page 235 for a complete listing of the inherited resources for the CellArray object class.

Macros

Macro XintIsCellArray returns True if specified *object* is a CellArray object. Otherwise returns False.

```
Boolean XintIsCellArray (Object object)
```

ComboPlot Object Class

A Chart Object creates a Plot object of type ComboPlot when the chart type resource is set to XintCHART_TYPE_COMBO_PLOT. This is a special plot type that can create and manage other plot types to build composite plots.

Resources

The ComboPlot object class inherits behavior and resources from the *Xt Object*, *Graphic* and *Group* object classes:

- Class pointer is *xintComboPlotObjectClass*
- Class name is *XintComboPlot*
- Header file is included as `<Xint/ComboPlot.h>`

Note: The ComboPlot object class does not define any new resources.

Inherited Resources

ComboPlot inherits behavior and resources from the Graphic class:

Name	Type Default	Access
XmNcolor	Pixel foreground	CSG
XmNdashList	char * NULL	CSG
XmNfillColor	Pixel “gray”	CSG
XmNfillFilename	char * NULL	CSG
XmNfillPixmap	Pixmap NULL	CSG
XmNfillStyle	int XintFILL_SOLID	CSG
XmNfont	char * “*Helvetica*-120-*”	CSG
XmNhighlightMode	int XintHIGHLIGHT_HANDLE	CSG

Name (continued)	**Type** **Default**	**Access**
XmNlineStyle	int XintLINE_NONE	CSG
XmNlineThickness	int 1	CSG
XmNmove	Boolean True	CSG
XmNsensitive	Boolean True	CSG
XmNshape	Boolean True	CSG
XmNstippleColor	Pixel foreground	CSG

Functions

The following functions are defined for creating new plot types and retrieving plot components.

Function Name	**Description**
XintComboPlotCreateNewPlot	Creates a new plot type inside the combo plot.
XintComboPlotGetComponent	Retrieves a component from a combo plot.

XintComboPlotCreateNewPlot

Creates a new plot object inside the ComboPlot object. You can use function XintChartAssociateData to attach data to this new plot object. The function returns the ID of the plot object that was created.

```
Object XintComboPlotCreateNewPlot (...)
```

Object	combo_plot	ID of the ComboPlot object.
int	plot_type	Type of plot to create.

Argument *plot_type* can be one of the following constants:

Constant	Description
XintPLOT_TYPE_BAR	Create a BarLine plot.
XintPLOT_TYPE_PIE	Create a Pie plot.
XintPLOT_TYPE_SURFACE_3D	Create a Surface3D plot.
XintPLOT_TYPE_HIGH_LOW	Create a HighLow plot.
XintPLOT_TYPE_HISTOGRAM	Create a Histogram plot.
XintPLOT_TYPE_AREA	Create a XYPlot configured as an area plot.
XintPLOT_TYPE_LINE	Create a XYPlot configured as a line plot.
XintPLOT_TYPE_SCATTERED	Create a XYPlot configured as a scattered plot.
XintPLOT_TYPE_BAR_3D	Create a Bar3D plot.

XintComboPlotGetComponent

Returns object ID of specified component of ComboPlot. Similar to XintChartGetComponent, except that it provides one more argument to take into account that multiple plots can be managed by a ComboPlot.

```
Object XintComboPlotGetComponent (...)
```

Object	combo_plot	ID of the ComboPlot object.
int	code	Code for the component to return.
int	index	The index of the plot of interest (starts at 0).

Argument *code* can be one of the following constants:

Constant	Description
XintCHART_COMPONENT_PLOT	Code for plot object.
XintCHART_COMPONENT_VERTICAL_AXIS	Code for vertical axis object.
XintCHART_COMPONENT_HORIZONTAL_ AXIS	Code for horizontal axis object.

Returns NULL if argument *index* is out of range.

Macros

Macro XintIsComboPlot returns True if specified *object* is a ComboPlot object. Otherwise returns False.

```
Boolean XintIsComboPlot (Object object)
```

BarLine Object Class

A Chart Object creates a Plot object of type BarLine when the chart type resource is set to XintCHART_TYPE_BAR. This plot type displays its data as horizontal or vertical bars or lines, as illustrated in Figure 37. A BarLine creates BarSeries and LineSeries objects to display the bars or the lines. See section BarLine Series for a complete list of the applicable resources.

.

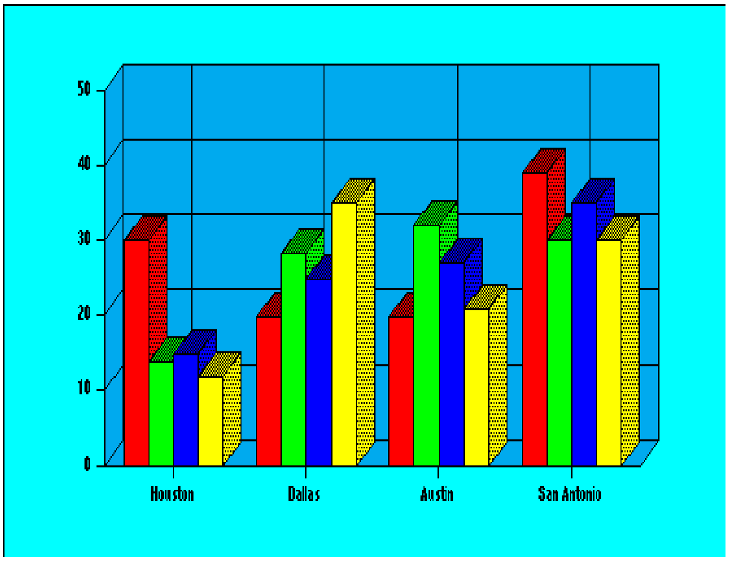


Figure 37. BarLine Chart Type

Data

The BarLine class only handles data objects of type DataSampled. It creates a BarSeries for each DataSampled object, with as many bars as the number of samples in the data object. No bars are created for missing or null values. All the DataSampled objects should have the same increment, but can have different starting values (see resource **XmNsampledRange** to specify a non default start and increment).

The BarLine class supports transposition. In this case a BarSeries is created at each sampled location, with as many bars as there are DataSampled objects.

Inherited Behavior and Resources

The BarLine object class inherits behavior and resources from the *Xt Object*, *Graphic*, *Group* and *Plot2D* object classes:

- Class pointer is *xintBarLineObjectClass*
- Class name is *XintBarLine*
- Header file is included as `<Xint/BarLine.h>`

The following resources are defined by the BarLine object class:

Name	Type Default	Access
XmNbarOrientation	int XintVERTICAL	CSG
XmNbarStyle	int XintADJACENT	CSG
XmNclusterWidth	int 75	CSG
XmNdrawShadow	Boolean False	CSG
XmNinclination	int 0	CSG
XmNperspectiveDepth	int 25	CSG
XmNproportional	Boolean False	CSG
XmNrotation	int 0	CSG

XmNbarOrientation

Specifies if the bars are oriented vertically (XintVERTICAL) or horizontally (XintHORIZONTAL).

XmNbarStyle

Specifies the bar style. You can use one of the following constants:

Resource Value	Description
XintADJACENT	Bars are adjacent.
XintOVERLAPPED	Bars overlap.
XintSTACKED	Bars are stacked.

XmNclusterWidth

Specifies, as a percentage between 0 and 100, the size of a bar cluster (all the bars assigned to the same sampled value). The difference between the specified size and 100 is the size allocated for the space between bar clusters. Specify 100 to have no space between bar clusters.

XmNdrawShadow

If True, and if either resource **XmNinclination** or **XmNdepth** is not 0, the side of the bars will drawn using a darker color which simulates a shadow.

XmNinclination

Specifies the inclination in degrees along the X axis (vertical bars) or the Y axis (horizontal bars) of the bars. This value must be between 0 and 45.

XmNperspectiveDepth

This resource specifies the depth of a bar as a percentage of the plot width. This resource is ignored if **XmNinclination** and **XmNrotation** are both 0.

XmNproportional

Specifies that the stacked bars should be made proportional to and plotted against an axis ranging between 0 and 100.

XmNrotation

Specifies the rotation clockwise in degrees along the Y axis (vertical bars) or the X axis (horizontal bars) of the bars. This value must be between 0 and 45.

Inherited Resources

Refer to *“Plot2D Object Metaclass”* on page 235 for a complete listing of the inherited resources for the BarLine object class.

Macros

The XintIsBarLine macro returns True if the specified *object* is a BarLine object. It returns False otherwise.

```
Boolean XintIsBarLine (Object object)
```

BarLine Series

The BarLine object class displays its data using BarSeries or Polyline Series objects. These object classes inherit their resources from the Graphic class.

BarSeries resources

The following table lists resources for BarSeries object classes (refer to *Chapter 3—Graphic Object Reference* for a complete description of these resources):

Name	Type Default	Access
XmNcolor	Pixel “black”	CSG
XmNdashList	char * NULL	CSG
XmNfillColor	Pixel assigned from color list	CSG
XmNfillFilename	char * NULL	CSG
XmNfillPixmap	Pixmap NULL	CSG
XmNfillStyle	int XintFILL_SOLID	CSG
XmNlineStyle	int XintLINE_SOLID	CSG
XmNlineThickness	int 1	CSG
XmNsensitive	Boolean True	CSG
XmNstippleColor	Pixel “black”	CSG

Polyline series resources

The following table lists resources for Polyline Series object classes:

Name	Type Default	Access
XmNcolor	Pixel assigned from color list	CSG
XmNdashList	char * NULL	CSG
XmNfillColor	Pixel assigned from color list	CSG
XmNfillFilename	char * NULL	CSG
XmNfillPixmap	Pixmap NULL	CSG
XmNfillStyle	int dynamic	CSG
XmNlineStyle	int dynamic	CSG
XmNlineThickness	int 1	CSG
XmNsensitive	Boolean True	CSG
XmNstippleColor	Pixel “black”	CSG
XmNsymbol	Boolean dynamic	CSG
XmNsymbolColor	Pixel assigned from color list	CSG
XmNsymbolSize	int8 8	CSG
XmNsymbolType	int assigned from symbol list	CSG

Bar3D Object

A Chart Object creates a Plot object of type Bar3D when the chart type resource is set to XintCHART_TYPE_BAR_3D. This plot type displays its data as a 3D bar plot. Figure 38 shows a sample 3D bar plot.

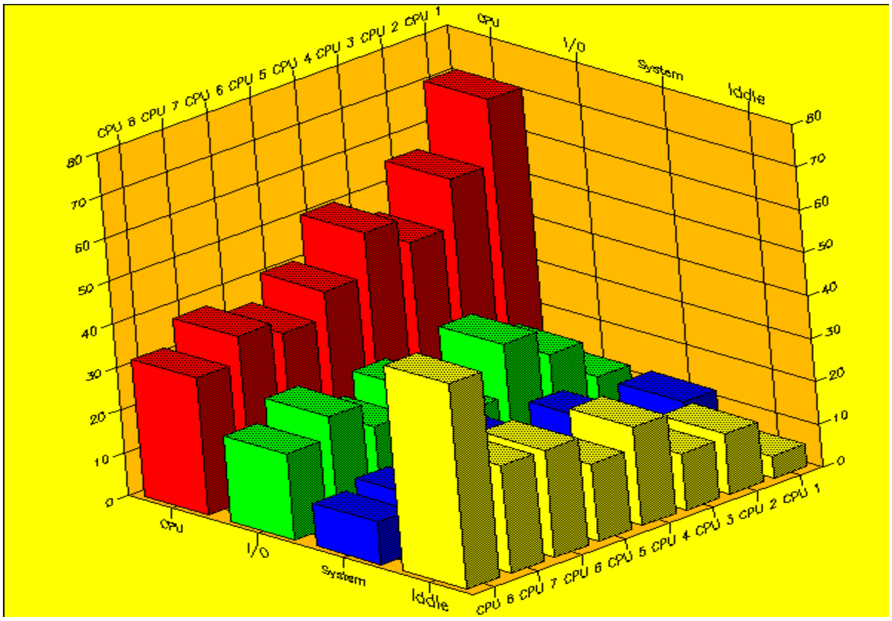


Figure 38. Bar3D Chart

Data

The Bar3D class accepts data objects of type Grid or DataSampled. It first looks for a Grid data object, if one is not found, it searches for DataSampled objects. All the DataSampled objects should have the same increment, but can have different starting values (see resource **XmNsampledRange** to specify a non default start and increment).

Transposition is supported for the case where a Bar3D object displays a set of DataSampled objects. If Chart resource **XmNtranspose** is False, DataSampled objects are plotted along the X direction, with a different color for each data object. If **XmNtranspose** it True, DataSampled objects are plotted along the Y direction. with a different color for each sample (color is uniform along the Y direction).

Inherited Behavior and Resources

The Bar3D object class inherits behavior and resources from the *Xt Object*, *Graphic*, *Group* and *Plot3D* object classes:

- Class pointer is *xintBar3DObjectClass*
- Class name is *XintBar3D*
- Header file is included as `<Xint/Bar3D.h>`

The following resources are defined by the Bar3D object class.

Name	Type Default	Access
XmNbarColorMode	int XintCOLOR_LIST_MODE	CSG
XmNbarLineColor	Pixel "black"	CSG
XmNbarLineThickness	int 1	CSG
XmNclusterHeight	int 75	CSG
XmNclusterWidth	int 75	CSG
XmNdrawShadow	Boolean True	CSG

XmNbarColorMode

Specifies how to assign the bar colors. You can use one of the following constants:

Resource Value	Description
XintCOLOR_LIST_MODE (default)	Bar color is uniform in the X direction (or Y if transposed) and colors are taken from the list specified in Chart resource **XmNcolorList**.
XintZ_VALUE_MODE	Bar color is based on the Z value and colors are assigned from the colors specified in Chart resource **XmNzValueColorRecord**.

XmNbarLineColor

Specifies the color pixel used to draw the bar outline.

XmNbarLineThickness

Specifies the bar line thickness in pixels.

XmNclusterHeight

Specifies the size of the bars along the Y direction as a percentage. The resource value must be an integer value between 0 and 100. The difference between the resource value and 100 corresponds to the space left between bars.

XmNclusterWidth

Specifies the size of the bars along the X direction as a percentage. The resource value must be an integer value between 0 and 100. The difference between the resource value and 100 corresponds to the space left between bars.

XmNdrawShadow

Specifies whether or not to draw the side of the bars using a darker color which simulates a shadow.

Inherited Resources

See section Plot3D Object Class for a complete listing of the inherited resources for the Bar3D object class.

Macros

Macro XintIsBar3D returns True if the specified *object* is a Bar3D object.

```
Boolean XintIsBar3D (Object object)
```

HighLow Object Class

A Chart Object creates a Plot object of type HighLow when the chart type resource is set to XintCHART_TYPE_HIGH_LOW. This plot type is often used for stock prices, to display high/low/open/close information. This type of plot can also be used for scientific data, such as to indicate temperature ranges. Resource **XmNdrawCandlestick** is available to draw the HighLow chart as a Candlestick plot. Figure 39 shows a typical HighLow chart.

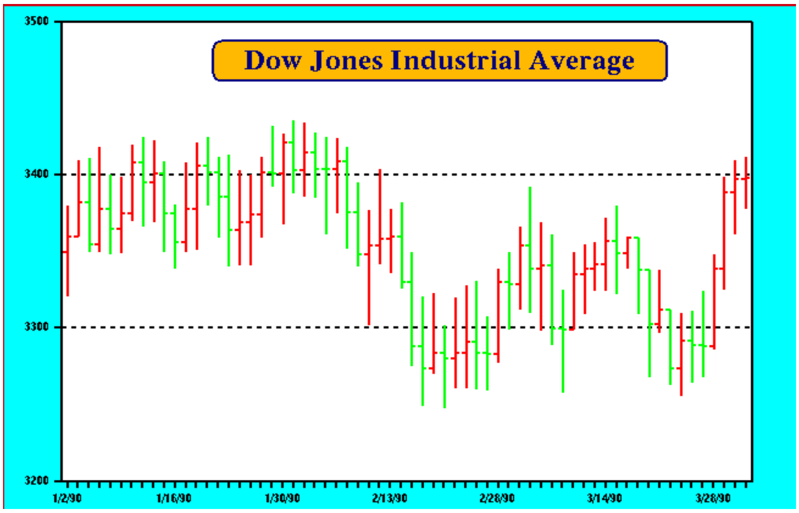


Figure 39. HighLow Chart type

Data

The HighLow class accepts only DataSampled series for data. You must pass DataGroup objects containing DataSampled objects in the following order: high, low, open and close. You need to pass a DataGroup containing at least two DataSampled series (for the high and low information). Open and close information is optional.

Note: Transposition is not supported for this plot object class.

The HighLow object class creates a HighLowSeries object to display a high-low-open-close curve. A maximum of one curve is created for each DataGroup associated with the chart. To display multiple curves you will need to associate additional DataGroup objects, each containing at least two DataSampled objects.

Inherited Behavior and Resources

The HighLow object class inherits behavior and resources from the *Xt Object*, *Graphic*, *Group* and *Plot2D* object classes:

- Class pointer is *xintHighLowObjectClass*
- Class name is *XintHighLow*
- Header file is included as `<Xint/HighLow.h>`

The following resources are defined by the HighLow object class:

Name	Type Default	Access
XmNdrawCandlestick	Boolean False	CSG
XmNshowClose	Boolean True	CSG
XmNshowGainLoss	Boolean False	CSG
XmNshowOpen	Boolean True	CSG
XmNtickLength	int 75	CSG

XmNdrawCandlestick

Specifies whether or not to draw the chart as a candlestick chart. In a candlestick chart, a filled rectangle is used to draw the area between the open and the close. The width of the rectangle is set by resource **XmNtickLength**. Resources **XmNshowClose**, **XmNshowOpen** are ignored when resource **XmNdrawCandlestick** is set to True.

XmNshowClose

Specifies whether or not the close indicator is displayed.

XmNshowGainLoss

Specifies whether samples corresponding to a loss are drawn using the same color as the samples corresponding to a gain. See section HighLow Series for a description of the resources to use to set the HighLowSeries colors.

XmNshowOpen

Specifies whether or not the open indicator is displayed.

XmNtickLength

Specifies the length of the open and close indicators as a percentage between 0 and 100. Specify 100 to have the close from one sample connected to the open from the next sample.

Inherited Resources

See section Plot2D Object Class for a complete listing of the inherited resources for the HighLow object class.

Macros

Macro XintIsHighLow returns True if the specified *object* is a HighLow object.

```
Boolean XintIsHighLow (Object object)
```

HighLow Series

The HighLow object class displays its data using HighLowSeries objects. This resource class inherits its resources from the Graphic class. The following is a list of the applicable resources for the HighLow object class. See class Graphic for a complete description of these resources.

Name	Type Default	Access
XmNcolor	Pixel assigned from color list	CSG
XmNdashList	char * NULL	CSG
XmNgainColor	Pixel “white”	CSG
XmNlineStyle	int XintLINE_SOLID	CSG
XmNlineThickness	int 1	CSG
XmNlossColor	Pixel “black”	CSG
XmNsensitive	Boolean True	CSG

For a HighLow plot, resource **XmNcolor** controls the color of the lines when resource **XmNshowGainLoss** is False. When resource **XmNshowGainLoss** is True, resources **XmNgainColor** and **XmNlossColor** are used to draw the samples corresponding to a gain or a loss respectively.

For a Candlestick plot, resource **XmNcolor** is used to draw the wick. The color used to draw the body of the candle is specified by resources **XmNgainColor** or **XmNlossColor**.

Histogram Object

A Chart Object creates a Plot object of type Histogram when the chart type resource is set to XintCHART_TYPE_HISTOGRAM. The histogram is used to display the distribution of a set of data, and optionally a cumulative curve. Figure 40 shows a typical Histogram chart.

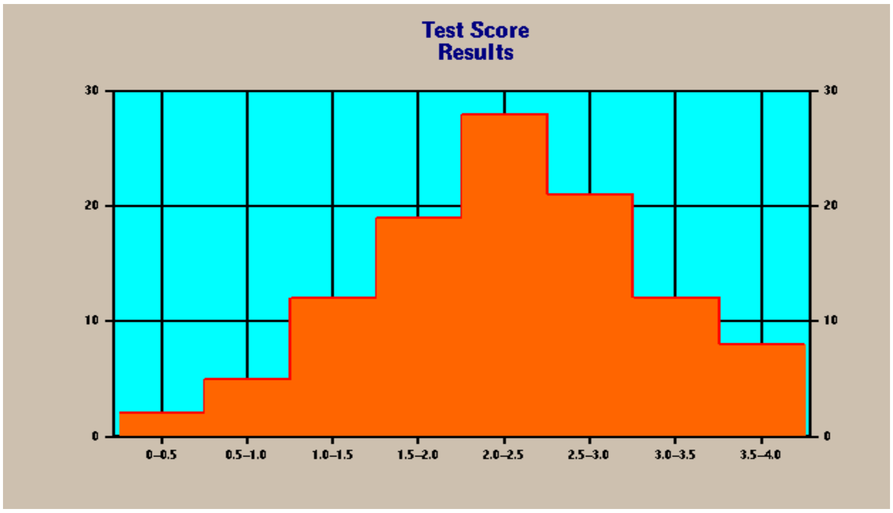


Figure 40. Histogram Chart Type

Data

The Histogram accepts only DataSampled series for data. For each DataSampled object it creates a HistogramSeries object (distribution curve) and a Line Series (cumulative curve). See section Histogram Series for a complete description of the resources applicable to the HistogramSeries class. Transposition is not supported for this plot class.

Inherited Behavior and Resources

The Histogram object class inherits behavior and resources from the *Xt Object*, *Graphic*, *Group* and *Plot2D* object classes:

- Class pointer is *xintHistogramObjectClass*
- Class name is *XintHistogram*
- Header file is included as `<Xint/Histogram.h>`

Resources The following resources are defined by the Histogram object class:

Name	Type Default	Access
XmNbarLength	int 100	CSG
XmNshowCumulative	Boolean False	CSG
XmNshowDistribution	Boolean True	CSG

XmNbarLength

Specifies the width of the bars used to display the distribution as a percentage between 0 and 100.

XmNshowCumulative

Specifies whether or not to display a curve representing the cumulative sum of the distribution.

XmNshowDistribution

Specifies whether or not to display the distribution curve.

Inherited Resources

See section Plot2D Object Class for a complete listing of the inherited resources for the Histogram object class.

Macros

Macro XintIsHistogram returns True if the specified *object* is an histogram object.

```
Boolean XintIsHistogram (Object object)
```

Histogram Series

The Histogram object class displays its data using HistogramSeries objects (distribution curve) and Polyline series objects (cumulative curve).

Graphic class resources

This object classes inherits its resources from the Graphic class. The following table lists resources for this class (refer to *Chapter 3—Graphic Object Reference* for a complete description of these resources):

Name	Type Default	Access
XmNcolor	Pixel assigned from color list	CSG
XmNdashList	char * NULL	CSG
XmNfillColor	Pixel assigned from color list	CSG
XmNfillFilename	char * NULL	CSG
XmNfillPixmap	Pixmap NULL	CSG
XmNfillStyle	int XintFILL_SOLID	CSG
XmNlineStyle	int XintLINE_SOLID	CSG
XmNlineThickness	int 1	CSG
XmNsensitive	Boolean True	CSG
XmNstippleColor	Pixel “black”	CSG

PolyLine class resources

The following table lists PolyLine class resources:

Name	Type Default	Access
XmNcolor	Pixel assigned from color list	CSG
XmNdashList	char * NULL	CSG
XmNlineStyle	int dynamic	CSG
XmNlineThickness	int 1	CSG
XmNsensitive	Boolean True	CSG
XmNstippleColor	Pixel “black”	CSG
XmNsymbol	Boolean False	CSG
XmNsymbolColor	Pixel Foreground	CSG
XmNsymbolSize	int8 8	CSG
XmNsymbolType	int XintSYMBOL_PLUS	CSG

Pie Object Class

A Chart Object creates a Plot object of type Pie when the chart type resource is set to XintCHART_TYPE_PIE. The Pie chart graphs data as a proportional slice of a circular pie. The resulting chart is useful in the comparison of the relative contribution of parts to a whole. Figure 41 shows a sample Pie chart.

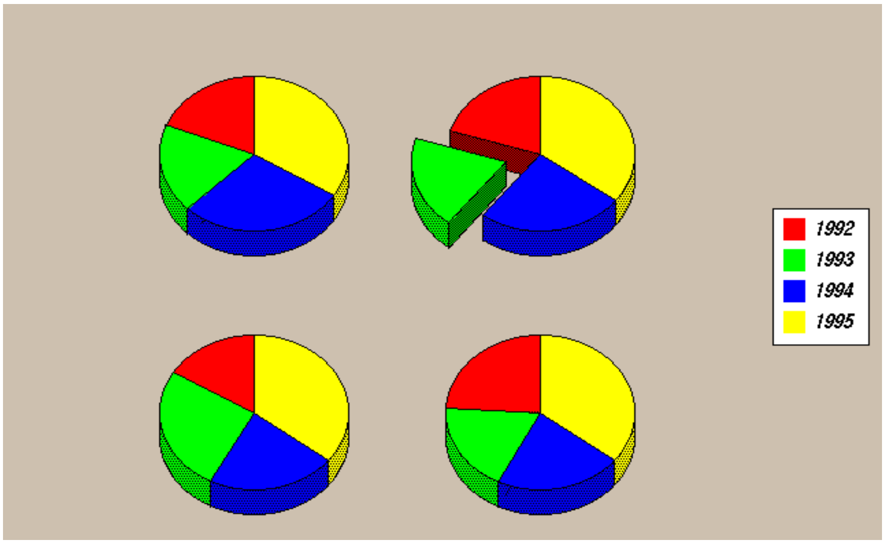


Figure 41. Pie Chart Type

Data

The Pie class only handles data objects of type DataSampled. It creates a WedgeSeries for each DataSampled, with as many Wedges as the number of samples in the data object. No wedges are created for missing or null values. All the DataSampled objects should have the same increment, but can have different starting values (see resource **XmNsampledRange** to specify a non default start and increment).

The Wedge class supports transposition. If Chart resource **XmNtranspose** is False, each wedge of a Wedge Series is placed in a different pie. If **XmNtranspose** is True, all the wedges of a WedgeSeries are placed in one pie (A DataSampled object defines an entire pie).

Inherited Behavior and Resources

The Pie object class inherits behavior and resources from the *Xt Object*, *Graphic*, *Group* and *Plot2D* object classes:

- Class pointer is *xintPieObjectClass*
- Class name is *XintPie*
- Header file is included as `<Xint/Pie.h>`

Resources The following resources are defined by the Pie object class:

Name	Type Default	Access
XmNdrawShadow	Boolean False	CSG
XmNinclination	int 0	CSG
XmNpieSize	int 75	CSG
XmNperspectiveDepth	int 25	CSG
XmNshowPieName	Boolean False	CSG
XmNshowWedgeLabels	Boolean False	CSG

XmNdrawShadow

Specifies whether or not to draw the side of the pie with a darker color.

XmNinclination

Specifies the inclination of the pie in degrees. You must specify an integer value between 0 and 45 for this resource

XmNpieSize

Each pie is centered in a square whose size is based on the number of pies and the width and height of the plot area. Resource **XmNpieSize** specifies how much of the square the pie occupies, as a percentage between 0 and 100.

XmNperspectiveDepth

Specifies the depth of the pie as a percentage of the Plot width. This resource has no effect if resource **XmNinclination** is 0.

XmNshowPieName

Specifies whether or not to display the name of each pie.

XmNshowWedgeLabels

Specifies whether or not to display the name of the series next to each wedge. The name of the series corresponds to the name of the DataSampled object to which each wedge corresponds.

Inherited Resources

Refer to *"Plot2D Object Metaclass"* on page 235 for a complete listing of the inherited resources for the Pie object class.

Functions

Function XintPieExplodeWedge is used to explode a wedge in a pie.

```
Boolean XintPieExplodeWedge (...)
```

Object	plot	ID of the Pie object.
Object	data	ID of the DataSampled object that contains the wedge.
int	sample_index	Index of the sample inside the DataSampled object (starts at 0).
int	series_index	Not used.
int	percent	Distance of explosion as a percentage of the pie radius.

Macros

Macro XintIsPie returns True if the specified *object* is a Pie object.

```
Boolean XintIsPie (Object object)
```

Pie Series

The Pie object class displays its data using WedgeSeries objects. This object class inherits its resources from the Graphic class. The following table lists the resources for this class (refer to *Chapter 3—Graphic Object Reference* for a complete description of these resources):

Name	Type Default	Access
XmNcolor	Pixel “black”	CSG
XmNdashList	char * NULL	CSG
XmNfillColor	Pixel assigned from color list	CSG
XmNfillFilename	char * NULL	CSG
XmNfillPixmap	Pixmap NULL	CSG
XmNfillStyle	int XintFILL_OPAQUE_STIPPLED	CSG
XmNlineStyle	int XintLINE_SOLID	CSG
XmNlineThickness	int 1	CSG
XmNsensitive	Boolean True	CSG
XmNstippleColor	Pixel “black”	CSG

Surface3D Object

A Chart Object creates a Plot object of type Surface3D when the chart type resource is set to XintCHART_TYPE_SURFACE_3D. This plot type displays its data as a 3D surface. Figure 42 shows a typical Surface3D chart:

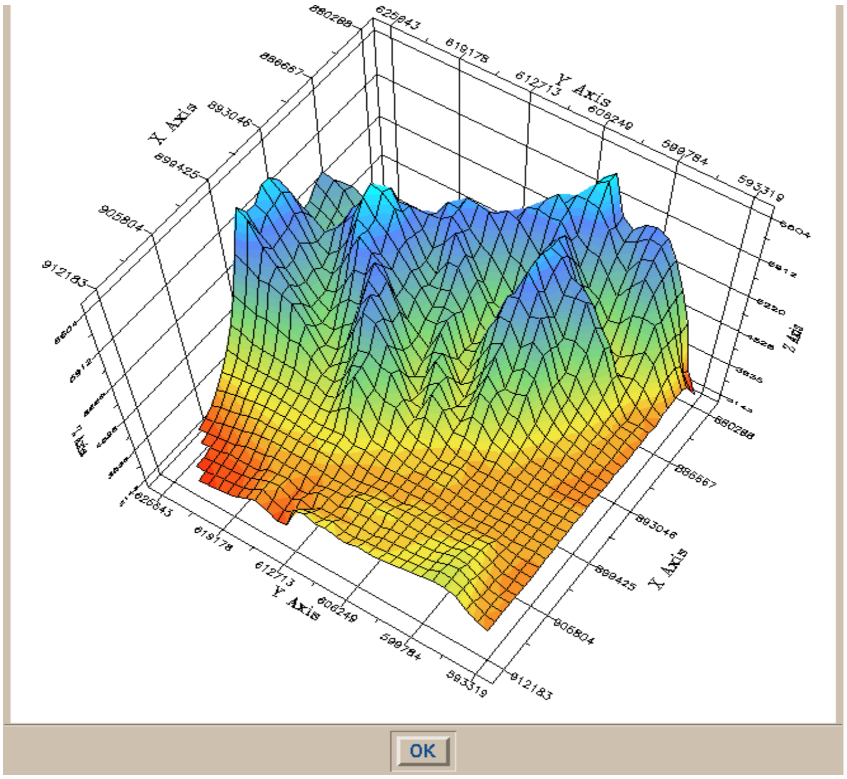


Figure 42. Surface3D Chart Type

Data

The Surface3D class accepts data objects of type Grid or DataSampled. It first looks for a Grid data object, if one is not found, it searches for DataSampled objects. All the DataSampled objects should have the same increment, but can have different starting values (see resource **XmNsampledRange** to specify a non default start and increment).

Transposition is supported for the case where a Surface3D object displays a set of DataSampled objects. If Chart resource **XmNtranspose** is False, DataSampled objects are plotted along the X direction. If **XmNtranspose** it True, DataSampled objects are plotted along the Y direction.

Inherited Behavior and Resources

The Surface3D object class inherits behavior and resources from the *Xt Object*, *Graphic*, *Group* and *Plot3D* object classes:

- Class pointer is *xintSurface3DObjectClass*
- Class name is *XintSurface3D*
- Header file is included as `<Xint/Surface3D.h>`

Resources The following resources are defined by the Surface3D object class;

Name	**Type** **Default**	**Access**
XmNcontourLineColor	Pixel black	CSG
XmNcontourLineMode	int XintNO_CONTOUR_LINE	CSG
XmNcontourLineThickness	int 1	CSG
XmNfillMode	int XintCONTOUR_FILL	CSG
XmNmeshColor	Pixel black	CSG
XmNmeshDrawX	Boolean True	CSG
XmNmeshDrawY	Boolean True	CSG

Name (continued)	**Type** **Default**	**Access**
XmNmeshMode	int XintMONOCHROME_MESH	CSG
XmNmeshLineThickness	int 1	CSG
XmNnumGridX	int 16	CSG
XmNnumGridY	int 16	CSG
XmNnumGridZ	int 16	CSG
XmNsolidSurface	Boolean False	CSG
XmNsurfaceBottomColor	Pixel gray	CSG
XmNsurfaceTopColor	Pixel light grey	CSG

XmNcontourLineColor

Specifies the color used to draw contour lines when resource **XmNcontourLineMode** is set to XintCONTOUR_LINE.

XmNcontourLineMode

Specifies what type of contour lines to draw. You can use one of the following constants:

Resource Value	**Description**
XintNO_CONTOUR_LINE (default)	No contour line is drawn.
XintCONTOUR_LINE	Draw the contour lines using color specified in resource XmNcontourLineColor.
XintCONTOUR_COLORED_LINE	Draw the contour lines using the color based on the Z value.

XmNcontourLineThickness

Specifies the line thickness in pixels used to draw the contour lines.

XmNfillMode

Specifies the surface fill mode. You can specify one of the following constants:

Resource Value	Description
XintNO_FILL	No fill.
XintSHADED_FILL	Surface is drawn using two colors. One for the top of the surface (**XmNsurfaceTopColor**) and one for the bottom of the surface (**XmNsurfaceBottomColor**).
XintMOSAIC_FILL	The surface is filled using a uniform color inside each grid cell. The color used is based on the average Z value for each cell.
XintCONTOUR_FILL	The surface is filled along the contour lines. The color used is based on the contour value.

XmNmeshColor

Specifies the color used to draw the grid mesh.

XmNmeshDrawX
XmNmeshDrawY

Specifies whether the grid mesh is drawn along the X (Y) direction.

XmNmeshMode

Specifies mode used to draw the mesh according to the following constants:

Resource Value	Description
XintNO_MESH	No grid mesh is drawn.
XintCONTOUR_MESH	Grid mesh is drawn using colors based on the Z values.
XintMONOCHROME_MESH (default)	Mesh is drawn with color specified in resource **XmNmeshColor**.

XmNmeshLineThickness

Specifies the thickness in pixels of the line used to draw the grid mesh.

XmNnumGridX
XmNnumGridY

Specifies the number of grid lines in the X (Y) direction.

XmNnumGridZ

Specifies the number of contour levels.

XmNsolidSurface

When this resource is TRUE, the surface is rendered as a solid object by adding “sides” to the surface. The sides drop from the surface boundaries to the minimum Z value. The mesh and surface bottom colors are used when drawing the “sides”.

XmNsurfaceBottomColor
XmNsurfaceTopColor

Specifies the pixel color used to draw the bottom (top) of the surface when resource **XmNfillMode** is set to XintSHADED_FILL

Inherited Resources

Refer to *“Plot3D Object Metaclass”* on page 241 for a complete listing of the inherited resources for the Surface3D object class.

Macros

Macro XintIsSurface3D returns True if the specified *object* is a Surface3D object.

```
Boolean XintIsSurface3D (Object object)
```

XYPlot Object Class

A Chart Object creates a Plot object of type XYPlot when the chart type resource is set to XintCHART_TYPE_AREA, XintCHART_TYPE_LINE or XintCHART_TYPE_SCATTERED. Figure 43 shows a sample XYPlot chart:

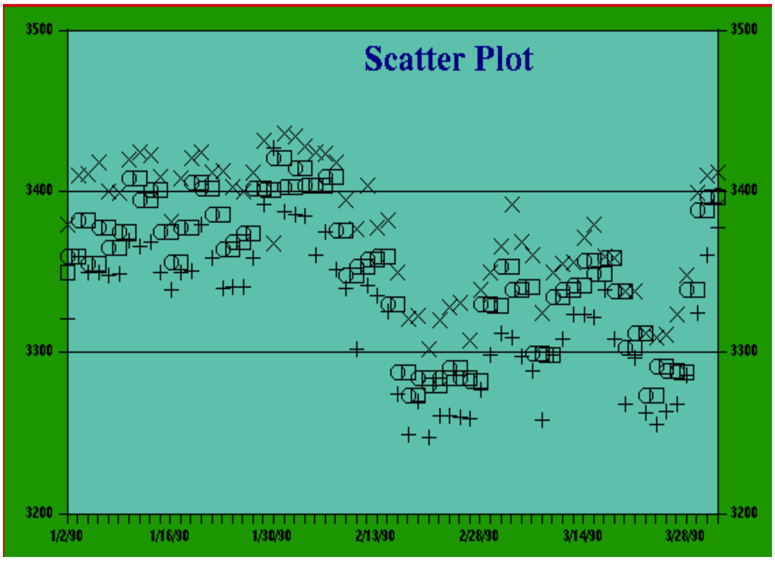


Figure 43. Scattered Plot Chart Type

Data

The XYPlot object class accepts DataSampled or DataSeries objects for data. DataSampled objects are plotted along the X axis.

If Chart resource **XmNtranspose** is False, a Polyline series is created for each DataSampled object. If **XmNtranspose** if True, a Polyline series is created for each sample (number of points in the Polyline series is equal to the number of DataSamPled objects). Transposition has no effect on DataSeries objects.

Inherited Behavior and Resources

The XYPlot object class inherits behavior and resources from the *Xt Object*, *Graphic*, *Group* and *Plot2D* object classes:

- Class pointer is *xintXYPlotObjectClass*
- Class name is *XintXYPlot*
- Header file is included as `<Xint/XYPlot.h>`

Resources

The following resources are defined by the XYPlot object class:

Name	Type Default	Access
XmNplotOrientation	int XintVERTICAL	CSG
XmNsymbol	Boolean dynamic	CSG
XmNsymbolSize	int 8	CSG
XmNstackingOrder	int XintFRONT_TO_BACK	CSG

XmNplotOrientation

Specifies the orientation of the plot.

Resource Value	Description
XintVERTICAL (default)	The X axis is plotted along the horizontal direction and the Y axis along the vertical direction.
XintHORIZONTAL	The X axis is plotted along the vertical direction and the Y axis is along the horizontal direction.

XmNsymbol

Specifies whether or not a symbol is displayed at each point location. The default for this resource is True for scattered plots and False otherwise. See Chart resource **XmNsymbolList** to see how symbols are assigned by default.

XmNsymbolSize

Specifies the size of the symbol in pixels.

XmNstackingOrder

Specifies the stacking order if multiple series are displayed. This resource is useful for area plots where the order in which the series are drawn is important. See also EditObject functions XintEditObjectRaise, XintEditObjectLower, XintEditObjectBack and XintEditObjectFront to control the stacking order on a object by object basis. You can specify the following constants for this resource:

Resource Value	Description
XintFRONT_TO_BACK (default)	Series are painted from first one specified to the last one.
XintBACK_TO_FRONT	Series are painted from the last one specified to the first one.

Inherited Resources

Refer to *"Plot2D Object Metaclass"* on page 235 for a complete listing of the inherited resources for the XYPlot object class.

Macros

Macro XintIsXYPlot returns True if the specified *object* is a XYPlot object.

```
Boolean XintIsXYPlot (Object object)
```

PolySeries

The XYPlot object class displays its data using PolySeries objects. The following table lists the resources for this class (refer to *“MultiPoint Object Metaclass”* on page 142 and *“Polyline Object Class”* on page 146 for a complete description of these resources):

Name	Type Default	Access
XmNcolor	Pixel assigned from color list	CSG
XmNdashList	char * NULL	CSG
XmNdrawSymbolCallback	XtCallbackList NULL	C
XmNfillColor	Pixel assigned from color list	CSG
XmNfillFilename	char * NULL	CSG
XmNfillPixmap	Pixmap NULL	CSG
XmNfillStyle	int dynamic	CSG
XmNlineStyle	int dynamic	CSG
XmNlineThickness	int 1	CSG
XmNsensitive	Boolean True	CSG
XmNstippleColor	Pixel “black”	CSG
XmNsymbol	Boolean dynamic	CSG
XmNsymbolColor	Pixel assigned from color list	CSG
XmNsymbolSize	int8 8	CSG
XmNsymbolType	int assigned from symbol list	CSG

Index

Actions

ChangeCursorMask 89
DrawCursor 89
EndAreaSelection 90
EndDrawCursor 89
ExtendAreaSelection 90
Increment 89
InitAreaSelection 89
InitDrawCursor 89
Locator 89
MotifDragStart 89
NextTabGroup 89
ObjectCancel 91
ObjectEdit 90
ObjectEditEnd 90
ObjectEditStart 90
ObjectPointAdd 91
ObjectPointDelete 91
ObjectSelect 90
Page 89
PreviousTabGroup 89
ResourceDialog 91
SelectionCallback 89
Transform3D 91
Transform3DEnd 91
Transform3DStart 91
TraverseCurrent 89

Callback Reasons

XintCR_AREA_SELECTION 93
XintCR_COPY 85, 93
XintCR_CUT 86, 93
XintCR_DATA_BATCH 170, 178, 185, 192, 200
XintCR_DATA_EXTEND 192, 200
XintCR_DATA_GROUP_DELETE_CHILD 170
XintCR_DATA_GROUP_INSERT_CHILD 170
XintCR_DATA_GROUP_UPDATE_CHILD 170
XintCR_DATA_REPLACE 178, 192, 200
XintCR_DATA_SHIFT 192, 200
XintCR_DATA_UPDATE 178, 185, 192, 200
XintCR_DRAG 93
XintCR_DROP 93
XintCR_INSERT_OBJECT 93
XintCR_LOCATOR 93
XintCR_MANAGE_RESOURCE_DIALOG 93
XintCR_OBJECT_ADD_POINT 130
XintCR_OBJECT_DELETE_POINT 130
XintCR_OBJECT_DESELECTION 93
XintCR_OBJECT_EDIT 93
XintCR_OBJECT_EDIT_END 93
XintCR_OBJECT_EDIT_START 93
XintCR_OBJECT_MOVE 130
XintCR_OBJECT_SELECTION 87, 93
XintCR_OBJECT_SHAPE 130
XintCR_PASTE 88, 93
XintCR_RUBBERBAND 93
XintCR_RUBBERBAND_END 93
XintCR_RUBBERBAND_START 93
XintCR_SELECTION 88, 93

Data Structures

Xint3DRotation 244
Xint3DScale 244
XintChartMargins 208
XintConstraint 212
XintDataGridUpdateCallbackStruct 177
XintDataGroupUpdateCallbackStruct 169

Index

Data Structures (continued)

XintDataLabelUpdateCallbackStruct 185
XintDataSampledUpdateCallbackStruct 191
XintDataSeriesUpdateCallbackStruct 199
XintEditObjectAreaSelectionCallbackStruct 94
XintEditObjectCallbackStruct 94
XintEditObjectDragDropCallbackStruct 95
XintEditObjectEditCallbackStruct 95
XintEditObjectInsertCallbackStruct 96
XintEditObjectLocatorCallbackStruct 97
XintEditObjectResourceDialogCallbackStruct 97
XintEditObjectRubberbandCallbackStruct 98
XintEditObjectSelectionCallbackStruct 96
XintGeometry 210
XintGraphicVerifyCallbackStruct 130
XintIncrements 237, 246
XintLimits 168, 227, 237, 246
XintLine 140
XintLineVerifyCallbackStruct 141
XintLocation 233
XintPolylineDrawSymbolCallbackStruct 148
XintPolylinePoint 147
XintRange 177, 184, 191
XintRectangle 150
XintRectangleVerifyCallbackStruct 145, 151
XintSymbolLocation 155
XintSymbolScale 155
XintTextLocation 162
XintTextScale 162
XintTextVerifyCallbackStruct 163

Defined Constants

EastGravity 154
NorthGravity 154
SouthGravity 154
WestGravity 154
XC_crosshair 84
XintACTION_EDITOR 113
XintADJACENT 254, 255
XintANCHOR_CENTER 212
XintANCHOR_EAST 212
XintANCHOR_NORTH 212
XintANCHOR_NORTH_EAST 212
XintANCHOR_NORTH_WEST 212
XintANCHOR_SOUTH 212
XintANCHOR_SOUTH_EAST 212
XintANCHOR_SOUTH_WEST 212
XintANCHOR_WEST 212
XintANNOTATION_CROSS 226
XintANNOTATION_INSIDE 226
XintANNOTATION_NONE 226
XintANNOTATION_OUTSIDE 225, 226
XintATTRIBUTE_EDITOR 113
XintBACK 111
XintBACK_TO_FRONT 280
XintCGM_VDC_TYPE_INTEGER 74
XintCGM_VDC_TYPE_REAL 74
XintCGM_WIDTH_MODE_ABSOLUTE 73, 74
XintCGM_WIDTH_MODE_SCALED 73, 74
XintCHART_COMPONENT_CHART 219
XintCHART_COMPONENT_FOOTER 218, 219
XintCHART_COMPONENT_HORIZONTAL_AXIS 218, 219, 252
XintCHART_COMPONENT_LEGEND 218, 219
XintCHART_COMPONENT_PLOT 218, 219, 252
XintCHART_COMPONENT_SERIES 219
XintCHART_COMPONENT_TITLE 218, 219
XintCHART_COMPONENT_USER_OBJECT 219
XintCHART_COMPONENT_VERTICAL_AXIS 218, 219, 252
XintCHART_TYPE_AREA 209
XintCHART_TYPE_BAR 207, 209
XintCHART_TYPE_BAR_3D 209
XintCHART_TYPE_CELL_ARRAY 209
XintCHART_TYPE_COMBINATION 209
XintCHART_TYPE_HIGH_LOW 209
XintCHART_TYPE_HISTOGRAM 209
XintCHART_TYPE_LINE 209
XintCHART_TYPE_PIE 209
XintCHART_TYPE_SCATTERED 209
XintCHART_TYPE_SURFACE_3D 209

XintCOLOR 77, 78
XintCOLOR_EDITOR 113
XintCOLOR_LIST_MODE 259
XintCOMPLEX_ROMAN 242
XintCONSTRAIN_PLACEMENT 212
XintCONTOUR_COLORED_LINE 275
XintCONTOUR_FILL 274, 276
XintCONTOUR_LINE 275
XintCONTOUR_MESH 276
XintCOPY 111
XintCOURIER 161
XintCROSS_HAIR_CURSOR 85, 100
XintCUT 111
XintDASH_TYPE 112
XintDATA_TYPE_DOUBLE 175, 190, 198
XintDATA_TYPE_FLOAT 174, 175, 189, 190, 197, 198
XintDATA_TYPE_INTEGER 175, 190, 198
XintDATA_TYPE_LONG 175, 190, 198
XintDATA_TYPE_SHORT 175, 190, 198
XintDEFAULT 161
XintDOUBLE_ARROW 140
XintEDIT_ADJUST 87
XintEDIT_INSERT 87, 103
XintEDIT_MOVE 87
XintEDIT_NONE 84, 87
XintEDIT_RUBBERBAND 87, 88
XintEDIT_SHAPE 87
XintEDIT_SIZE 87
XintEND_ARROW 140
XintFILL_COLOR 112
XintFILL_NONE 124, 126, 234
XintFILL_OPAQUE_STIPPLED 126, 272
XintFILL_SOLID 126, 239, 250, 256, 267
XintFILL_STIPPLED 126
XintFILL_STYLE 112
XintFILLED 139
XintFONT_SIZE_DEFAULT 161
XintFRONT 111
XintFRONT_TO_BACK 279, 280
XintGROUP 111
XintHALIGN_CENTER 161
XintHALIGN_LEFT 160, 161
XintHALIGN_RIGHT 161
XintHANDLE_BOUNDS 143
XintHANDLE_POINTS 143
XintHELVETICA 160, 161
XintHERSHEY_SIMPLEX 245
XintHERSHEY_TRIPLEX 244
XintHIGHLIGHT_COLOR 127
XintHIGHLIGHT_HANDLE 124, 127, 143, 229, 234, 239, 250
XintHIGHLIGHT_NONE 127
XintHORIZONTAL 110, 254, 279
XintIMAGE_FIXED 136
XintIMAGE_INTERPOLATE 135, 136
XintLABEL_X 182, 183
XintLABEL_Y 183
XintLABEL_Z 183
XintLINE_COLOR 112
XintLINE_DOUBLE_DASH 125, 127, 228, 233
XintLINE_ENDS 112
XintLINE_NONE 127, 225, 228, 232, 233, 239, 251
XintLINE_ON_OFF_DASH 125, 127, 228, 233
XintLINE_SOLID 124, 127, 228, 229, 233, 234, 256, 264, 267, 272
XintLINE_STYLE 112
XintLINE_WIDTH 112
XintLOWER 111
XintMONOCHROME 77, 78
XintMONOCHROME_MESH 275, 276
XintMOSAIC_FILL 276
XintMOVE_ANY 125, 128
XintMOVE_HORIZONTAL 128
XintMOVE_VERTICAL 128
XintNEW_CENTURY_SCHOOLBOOK 161
XintNO_ARROW 139, 140
XintNO_CONTOUR_LINE 274, 275

Defined Constants (continued)

XintNO_FILL 80, 276
XintNO_MESH 276

Index

XintOBJECT_EDITOR 113
XintORIENTATION _AUTOMATIC 78, 79
XintORIENTATION _LANDSCAPE 78, 79
XintORIENTATION_PORTRAIT 78, 79
XintOVERLAPPED 255
XintPASTE 111
XintPATTERN_COLOR 112
XintPIXMAP_EDITOR 113
XintPLACEMENT_BOTTOM 235, 236
XintPLACEMENT_INSIDE 236, 237, 238
XintPLACEMENT_LEFT 235, 238
XintPLACEMENT_LEFT_RIGHT 238
XintPLACEMENT_NONE 236, 238
XintPLACEMENT_RIGHT 238
XintPLACEMENT_TOP 236
XintPLACEMENT_TOP_BOTTOM 236
XintPLOT_TYPE_AREA 252
XintPLOT_TYPE_BAR 252
XintPLOT_TYPE_BAR_3D 252
XintPLOT_TYPE_HIGH_LOW 252
XintPLOT_TYPE_HISTOGRAM 252
XintPLOT_TYPE_LINE 252
XintPLOT_TYPE_PIE 252
XintPLOT_TYPE_SCATTERED 252
XintPLOT_TYPE_SURFACE_3D 252
XintRAISE 111
XintROUND_MIN_MAX 235, 236, 238
XintSHADED_FILL 276
XintSHADOW_IN 233
XintSHADOW_OUT 233
XintSIMPLEX_ROMAN 242
XintSLANT_DEFAULT 161
XintSLANT_OBLIQUE 161
XintSLANT_REGULAR 160, 161
XintSTACKED 255
XintSTART_ARROW 140
XintSYMBOL 161
XintSYMBOL_CIRCLE 144, 148, 211
XintSYMBOL_DATA 144
XintSYMBOL_DIAMOND 144, 148, 211
XintSYMBOL_FILLED_CIRCLE 144, 148, 211
XintSYMBOL_FILLED_DIAMOND 144, 148, 211
XintSYMBOL_FILLED_SQUARE 144, 148, 211
XintSYMBOL_FILLED_TRIANGLE 144, 148, 211
XintSYMBOL_PLUS 143, 144, 148, 211, 268
XintSYMBOL_SQUARE 144, 148, 211
XintSYMBOL_TRIANGLE 144, 148, 211
XintSYMBOL_X 144, 148, 211
XintTICK_CROSS 228
XintTICK_INSIDE 228
XintTICK_NONE 228
XintTICK_OUTSIDE 225, 228
XintTIME_DAY 204
XintTIME_HOUR 204
XintTIME_MINUTE 204
XintTIME_MONTH 204
XintTIME_SECOND 203, 204
XintTIME_YEAR 204
XintTIMES 161
XintUNDEFINED_DOUBLE 14
XintUNDEFINED_FLOAT 14, 237
XintUNDEFINED_INTEGER 14
XintUNDEFINED_LONG 14
XintUNDEFINED_SHORT 14
XintUNGROUP 111
XintUSE_MIN_MAX 236, 238
XintVALIGN_BOTTOM 162
XintVALIGN_CENTER 162
XintVALIGN_TOP 160, 162
XintVERTICAL 110, 111, 254, 279
XintWARNING_NONE 71
XintWARNING_POST 70, 71
XintWARNING_PRINT 71
XintWEIGHT_BOLD 161
XintWEIGHT_DEFAULT 161
XintWEIGHT_MEDIUM 160, 161
XintX_VECTOR 174, 176
XintXY_PLANE 243, 245
XintXZ_PLANE 243, 245
XintY_VECTOR 176
XintYZ_PLANE 243, 245
XintZ_VALUE_MODE 259

XmUNSPECIFIED_PIXMAP 135, 249

Functions

XintAxisObjectPixelToUser 231
XintAxisObjectUserToPixel 230, 240
XintCGMDrawBox 72
XintCGMGetDimensions 73
XintCGMPixelToInch 73
XintCGMSetEdgeWidthMode 73
XintCGMSetLineWidthMode 74
XintCGMSetVDCType 74
XintChartAssociateData 216
XintChartCreateColorRecord 216
XintChartDisassociateAllData 216
XintChartDisassociateData 217
XintChartFreezeUpdates 217
XintChartGetComponent 217
XintChartGetDataList 218
XintChartGetDataOfSeries 218
XintChartGetSelectedComponent 219
XintChartGetSeriesOfData 220
XintChartInitializeClassConverter 220
XintChartInsertObject 220
XintChartIsTransposed 221
XintChartPick 221
XintChartReadTemplate 221
XintChartSaveTemplate 222
XintChartWidgetGetObject 223
XintChartZoom 59, 222
XintComboPlotCreateNewPlot 251
XintComboPlotGetComponent 252
XintCreateChart 216
XintCreateChartWidget 223
XintCreateDataGrid 179
XintCreateDataGroup 171
XintCreateDataLabel 186
XintCreateDataSampled 193
XintCreateDataSeries 200
XintCreateEditObject 100
XintCreateGroup 134
XintCreateImageObject 137
XintCreateLine 141
XintCreateObjectEditor 116
XintCreateOval 145
XintCreatePolyline 149
XintCreateRectangle 151
XintCreateSymbol 158
XintCreateText 163
XintDataBatchUpdate 171
XintDataGridDataReplace 180
XintDataGridGetGridArray 179
XintDataGroupFind 172
XintDataGroupIterate 172
XintDataLabelExtend 186
XintDataLabelReplace 187
XintDataLabelShift 187
XintDataRangeX 172
XintDataRangeY 172
XintDataRangeZ 172
XintDataSampledDataExtend 193
XintDataSampledDataReplace 194
XintDataSampledDataShift 195
XintDataSampledGetDataArray 194
XintDataSampledGetSampledArray 194
XintDataSeriesDataExtend 201
XintDataSeriesDataReplace 201
XintDataSeriesDataShift 202
XintDataSeriesGetXArray 202
XintDataSeriesGetYArray 202
XintDrawCursorFromData 100
XintEditObjectBack 100
XintEditObjectCopy 100
XintEditObjectCurrent 101
XintEditObjectCut 101
XintEditObjectDeselectAll 101
XintEditObjectDeselectObject 101
XintEditObjectDestroyObject 101
XintEditObjectFreeze 102
XintEditObjectFront 102

Functions (continued)

XintEditObjectGetIntersectList 102
XintEditObjectGetList 103

Index

XintEditObjectGroup 103
XintEditObjectInsert 103
XintEditObjectLower 104
XintEditObjectManageResourceDialog 104
XintEditObjectMove 104
XintEditObjectNew 105
XintEditObjectOpen 105
XintEditObjectPaste 105
XintEditObjectRaise 105
XintEditObjectReadFile 105, 108
XintEditObjectSave 106
XintEditObjectSaveAs 106
XintEditObjectSelectAll 106
XintEditObjectSelectList 106
XintEditObjectSelectObject 106
XintEditObjectSetEditMode 107
XintEditObjectSize 107
XintEditObjectUngroup 107
XintEditObjectWriteFile 107
XintGetWidgetSize 75
XintGraphicGetViewPortList 131
XintGraphicUnmanageDialog 131
XintHorizontalPixelToUser 75
XintHorizontalUserToPixel 75
XintIsAxisObject 231
XintIsBar3D 260
XintIsBarLine 256
XintIsCellArray 249
XintIsChart 222
XintIsComboPlot 252
XintIsDataGrid 180
XintIsDataGroup 172
XintIsDataLabel 187
XintIsDataSampled 195
XintIsDataSeries 202
XintIsGraphic 131
XintIsGroup 134
XintIsHighLow 263
XintIsHistogram 266
XintIsImageObject 137
XintIsLegend 234
XintIsLine 141
XintIsOval 145
XintIsPie 271
XintIsPolyline 149
XintIsRectangle 151
XintIsSurface3D 277
XintIsSymbol 158
XintIsText 163
XintIsXYPlot 280
XintObjectEditorAddEditObjectToList 119
XintObjectEditorGetDefinedPixmap 116
XintObjectEditorRemoveEditObjectFromList 119
XintOutputCGM 76
XintOutputMontageCGM 76
XintOutputMontagePostscript 77
XintOutputPostscript 78
XintPieExplodeWedge 271
XintPostscriptGetDefaults 79
XintPostscriptSetBackground 80
XintPostscriptSetDefaults 80
XintResEditGetBox 36
XintResEditGetLabel 37
XintResEditGetMenuButton 37
XintSymbolCreate 158
XintSymbolFree 158
XintVerticalPixelToUser 80
XintVerticalUserToPixel 80

Resources

XintHERSHEY_SIMPLEX 244
XintHERSHEY_TRIPLEX 245
XmN3dPerspectiveDepth 242
XmN3dRotation 242
XmN3dScale 242
XmNactionList 110
XmNactionNumColumns 110
XmNactionOrientation 110
XmNallowDrag 84
XmNallowDrop 84
XmNannotationAngle 225
XmNannotationFont 225
XmNannotationFormat 225

XmNannotationPlacement 225
XmNannotationStrokeFont 242
XmNannotationStrokeFontSize 242
XmNareaSelectionCallback 84, 93, 94
XmNarrowLength 139
XmNarrowStyle 139
XmNattributeList 110
XmNattributeNumColumns 110
XmNattributeOrientation 110
XmNaxisLineThickness 225
XmNaxisSpacing 207
XmNbarColorMode 259
XmNbarLength 266
XmNbarLineColor 259
XmNbarLineThickness 259
XmNbarOrientation 254
XmNbarStyle 254
XmNbaseAngle 139
XmNchartFooter 207
XmNchartMargins 207
XmNchartTitle 207
XmNchartType 207
XmNclipGrid 124
XmNcloseEndPoints 147
XmNclusterHeight 259
XmNclusterWidth 254, 259
XmNcolor 124, 229, 234, 239, 250, 257, 264, 267, 272, 281
XmNcolorAttributeList 110
XmNcolorList 110, 207
XmNcolorNumColumns 110
XmNcolorOrientation 110
XmNcolumns 232
XmNconstraint 208, 213
XmNcontourLineColor 274
XmNcontourLineMode 274
XmNcontourLineThickness 274
XmNcopyCallback 84, 93, 95
XmNcopyData 174, 182, 189, 197
XmNcount 189, 197
XmNcursorType 84
XmNcutCallback 84, 93, 95
XmNdashList 124, 229, 234, 239, 250, 257, 264, 267, 272, 281
XmNdataArray 189
XmNdataGroup 174, 182, 189, 197
XmNdataType 174, 189, 197
XmNdisplayName 124
XmNdisplayPatterns 248
XmNdoubleBuffer 207
XmNdragDropCallback 84, 93, 95
XmNdrawCandlestick 262
XmNdrawFrame 235
XmNdrawShadow 254, 259, 270
XmNdrawSymbolCallback 147, 148, 281
XmNeditable 174, 189, 197
XmNeditList 110
XmNeditObjectCallback 84, 93, 94
XmNeditObjects 110
XmNendPoints 225
XmNfillColor 124, 234, 239, 250, 257, 267, 272, 281
XmNfillFilename 124, 234, 239, 250, 257, 267, 272, 281
XmNfillMode 274
XmNfillPixmap 124, 234, 239, 250, 257, 267, 272, 281
XmNfillStyle 124, 234, 239, 250, 257, 267, 272, 281
XmNflip 84
XmNfont 124, 229, 234, 239, 250
XmNfontFamily 160
XmNfontPath 70
XmNfontSize 160
XmNfontSlant 160
XmNfontWeight 160
XmNfreePixmap 135
XmNgainColor 264
XmNgeometry 207

Resources (continued)

XmNgridArray 174
XmNgridOrder 174
XmNgroup 124

Index

XmNhandleColor 84
XmNhandleMode 143
XmNhandleSize 84
XmNhighlightMode 124, 229, 234, 239, 250
XmNhistoryLength 167
XmNhorizontalAxis 124
XmNhorizontalTextAlignment 160
XmNiconWidth 232
XmNimageColorRecord 135
XmNimageDisplayMode 135
XmNimagePixmap 135
XmNinclination 254, 270
XmNincrements 225
XmNinsertObjectCallback 84, 93, 96
XmNlabel 225
XmNlabelCount 182
XmNlabelFont 225
XmNlabelGravity 154
XmNlabelOrientation 182
XmNlabelPositionArray 182
XmNlabelSpacing 154
XmNlabelStrings 182
XmNlabelStrokeFont 242
XmNlabelStrokeFontSize 242
XmNlastViewDestroy 167, 174, 182, 189, 197
XmNlegendAnnotationFormat 248
XmNlegendLocation 232
XmNlegendTitle 232
XmNlimits 225
XmNlimitsX 167
XmNlimitsY 167
XmNlimitsZ 167
XmNline 139
XmNlineEnd 139
XmNlineStyle 124, 229, 232, 234, 239, 251, 257, 264, 267, 272, 281
XmNlineThickness 125, 229, 234, 239, 251, 257, 264, 267, 272, 281
XmNlist 134
XmNlistCount 134
XmNlocatorCallback 84, 93, 97
XmNlogScale 225
XmNlossColor 264
XmNmajorGridLineStyle 225
XmNmarginHeight 160, 232
XmNmarginWidth 160, 232
XmNmeshColor 274
XmNmeshDrawX 274
XmNmeshDrawY 274
XmNmeshLineThickness 275
XmNmeshMode 275
XmNminimumLabelSpacing 225
XmNminorGridLineStyle 225
XmNmove 125, 229, 234, 239, 251
XmNmoveDirection 125
XmNname 125
XmNnullValue 147
XmNnumColumns 110
XmNnumEditObjects 110
XmNnumGridX 275
XmNnumGridY 275
XmNnumGridZ 275
XmNobjectClassList 110
XmNobjectDeselectionCallback 84, 93, 96
XmNobjectEditMode 84
XmNobjectSelectionCallback 84, 93, 96
XmNorientation 110
XmNpasteCallback 84, 93, 95
XmNpatternList 248
XmNperspectiveDepth 254, 270
XmNpieSize 270
XmNpixmapList 111
XmNpixmapNumColumns 111
XmNpixmapOrientation 111
XmNplotOrientation 279
XmNpointArray 147
XmNpointCount 147
XmNpointSelectionTolerance 85
XmNpropagate 134, 207, 235
XmNproportional 254
XmNrectangle 150
XmNresourceDialog 125
XmNresourceDialogCallback 85, 93, 97
XmNreverseOrder 225

XmNrotateAngle 150, 160
XmNrotation 254
XmNroundEdge 150
XmNrubberbandCallback 85, 93, 98
XmNsampledRange 182, 189
XmNselectionCallback 85, 93, 94
XmNsensitive 125, 229, 234, 239, 251, 257, 264, 267, 272, 281
XmNshape 125, 229, 234, 239, 251
XmNshowAttributeLabels 111
XmNshowClose 262
XmNshowCumulative 266
XmNshowDistribution 266
XmNshowGainLoss 262
XmNshowLegend 207
XmNshowOpen 262
XmNshowPieName 270
XmNshowWedgeLabels 270
XmNsolidSurface 275
XmNstackingOrder 279
XmNstippleColor 125, 229, 234, 239, 251, 257, 267, 272, 281
XmNsurfaceBottomColor 275
XmNsurfaceTopColor 275
XmNsymbol 143, 257, 268, 279, 281
XmNsymbolColor 143, 257, 268, 281
XmNsymbolCount 207
XmNsymbolData 143, 154
XmNsymbolHeight 154
XmNsymbolList 207
XmNsymbolLocation 154
XmNsymbolScale 154
XmNsymbolSize 143, 257, 268, 279, 281
XmNsymbolType 143, 257, 268, 281
XmNsymbolWidth 154
XmNtextLocation 160
XmNtextScale 160
XmNtextString 160
XmNtickLength 262
XmNtickPlacement 225
XmNtimeBase 203
XmNtimeLabelFormat 203
XmNtimeMeasure 203
XmNtipAngle 139
XmNtranspose 207
XmNupdateCallback 174, 182, 189, 197
XmNuserData 125
XmNverifyCallback 125, 130, 145
XmNverticalAxis 125
XmNverticalTextAlignment 160
XmNviewScale 242
XmNvisible 125
XmNvisibleEntryCount 232
XmNwallColor 243
XmNwarning 70
XmNxAnnotation 242
XmNxAnnotationFormat 242
XmNxArray 197
XmNxAutoRangeMode 235
XmNxAxisOffset 242
XmNxAxisPlacement 235
XmNxCount 174
XmNxGridLines 243
XmNxIncrements 235, 243
XmNxInsidePlacement 235
XmNxLabel 243
XmNxLimits 235, 243
XmNxRange 174
XmNxTranslation 243
XmNyAnnotation 242
XmNyAnnotationFormat 242
XmNyArray 197
XmNyAutoRangeMode 235
XmNyAxisOffset 243
XmNyAxisPlacement 235
XmNyCount 174
XmNyGridLines 243
XmNyIncrements 235, 243
XmNyInsidePlacement 235

Resources (continued)

XmNyLabel 243
XmNyLimits 235, 243
XmNyRange 174

Index

XmNyTranslation 243
XmNzAnnotation 242
XmNzAnnotationFormat 242
XmNzAxisOffset 243
XmNzGridLines 243
XmNzIncrements 243
XmNzLabel 243
XmNzLimits 243
XmNzValueColorRecord 207

Index

Index

Callback Reasons
XintCR_AREA_SELECTION 93
XintCR_COPY 85, 93
XintCR_CUT 86, 93
XintCR_DATA_BATCH 170, 178, 185, 192, 200
XintCR_DATA_EXTEND 192, 200
XintCR_DATA_GROUP_DELETE_CHILD 170
XintCR_DATA_GROUP_INSERT_CHILD 170
XintCR_DATA_GROUP_UPDATE_CHILD 170
XintCR_DATA_REPLACE 178, 192, 200
XintCR_DATA_SHIFT 192, 200
XintCR_DATA_UPDATE 178, 185, 192, 200
XintCR_DRAG 93
XintCR_DROP 93
XintCR_INSERT_OBJECT 93
XintCR_LOCATOR 93
XintCR_MANAGE_RESOURCE_DIALOG 93
XintCR_OBJECT_ADD_POINT 130
XintCR_OBJECT_DELETE_POINT 130
XintCR_OBJECT_DESELECTION 93
XintCR_OBJECT_EDIT 93
XintCR_OBJECT_EDIT_END 93
XintCR_OBJECT_EDIT_START 93
XintCR_OBJECT_MOVE 130
XintCR_OBJECT_SELECTION 87, 93
XintCR_OBJECT_SHAPE 130
XintCR_PASTE 88, 93
XintCR_RUBBERBAND 93
XintCR_RUBBERBAND_END 93
XintCR_RUBBERBAND_START 93
XintCR_SELECTION 88, 93

Index

Index

Data Structures
Xint3DRotation 244
Xint3DScale 244
XintChartMargins 208
XintConstraint 212
XintDataGridUpdateCallbackStruct 177
XintDataGroupUpdateCallbackStruct 169
XintDataLabelUpdateCallbackStruct 185
XintDataSampledUpdateCallbackStruct 191
XintDataSeriesUpdateCallbackStruct 199
XintEditObjectAreaSelectionCallbackStruct 94
XintEditObjectCallbackStruct 94
XintEditObjectDragDropCallbackStruct 95
XintEditObjectEditCallbackStruct 95
XintEditObjectInsertCallbackStruct 96
XintEditObjectLocatorCallbackStruct 97
XintEditObjectResourceDialogCallbackStruct 97
XintEditObjectRubberbandCallbackStruct 98
XintEditObjectSelectionCallbackStruct 96
XintGeometry 210
XintGraphicVerifyCallbackStruct 130
XintIncrements 237, 246
XintLimits 168, 227, 237, 246
XintLine 140
XintLineVerifyCallbackStruct 141
XintLocation 233
XintPolylineDrawSymbolCallbackStruct 148
XintPolylinePoint 147
XintRange 177, 184, 191
XintRectangle 150
XintRectangleVerifyCallbackStruct 145, 151
XintSymbolLocation 155
XintSymbolScale 155
XintTextLocation 162
XintTextScale 162
XintTextVerifyCallbackStruct 163

Index

Index

Defined Constants

E

EastGravity 154
NorthGravity 154
SouthGravity 154
WestGravity 154
XC_crosshair 84
XintACTION_EDITOR 113
XintADJACENT 254, 255
XintANCHOR_CENTER 212
XintANCHOR_EAST 212
XintANCHOR_NORTH 212
XintANCHOR_NORTH_EAST 212
XintANCHOR_NORTH_WEST 212
XintANCHOR_SOUTH 212
XintANCHOR_SOUTH_EAST 212
XintANCHOR_SOUTH_WEST 212
XintANCHOR_WEST 212
XintANNOTATION_CROSS 226
XintANNOTATION_INSIDE 226
XintANNOTATION_NONE 226
XintANNOTATION_OUTSIDE 225, 226
XintATTRIBUTE_EDITOR 113
XintBACK 111
XintBACK_TO_FRONT 280
XintCGM_VDC_TYPE_INTEGER 74
XintCGM_VDC_TYPE_REAL 74
XintCGM_WIDTH_MODE_ABSOLUTE 73, 74
XintCGM_WIDTH_MODE_SCALED 73, 74
XintCHART_COMPONENT_CHART 219
XintCHART_COMPONENT_FOOTER 218, 219
XintCHART_COMPONENT_HORIZONTAL_AXIS 218, 219, 252
XintCHART_COMPONENT_LEGEND 218, 219
XintCHART_COMPONENT_PLOT 218, 219, 252
XintCHART_COMPONENT_SERIES 219
XintCHART_COMPONENT_TITLE 218, 219
XintCHART_COMPONENT_USER_OBJECT 219
XintCHART_COMPONENT_VERTICAL_AXIS 218, 219, 252
XintCHART_TYPE_AREA 209
XintCHART_TYPE_BAR 207, 209
XintCHART_TYPE_BAR_3D 209
XintCHART_TYPE_CELL_ARRAY 209
XintCHART_TYPE_COMBINATION 209
XintCHART_TYPE_HIGH_LOW 209
XintCHART_TYPE_HISTOGRAM 209
XintCHART_TYPE_LINE 209
XintCHART_TYPE_PIE 209
XintCHART_TYPE_SCATTERED 209
XintCHART_TYPE_SURFACE_3D 209
XintCOLOR 77, 78
XintCOLOR_EDITOR 113
XintCOLOR_LIST_MODE 259
XintCOMPLEX_ROMAN 242
XintCONSTRAIN_PLACEMENT 212
XintCONTOUR_COLORED_LINE 275
XintCONTOUR_FILL 274, 276
XintCONTOUR_LINE 275
XintCONTOUR_MESH 276
XintCOPY 111
XintCOURIER 161
XintCROSS_HAIR_CURSOR 85, 100
XintCUT 111
XintDASH_TYPE 112
XintDATA_TYPE_DOUBLE 175, 190, 198
XintDATA_TYPE_FLOAT 174, 175, 189, 190, 197, 198
XintDATA_TYPE_INTEGER 175, 190, 198
XintDATA_TYPE_LONG 175, 190, 198
XintDATA_TYPE_SHORT 175, 190, 198
XintDEFAULT 161
XintDOUBLE_ARROW 140
XintEDIT_ADJUST 87
XintEDIT_INSERT 87, 103
XintEDIT_MOVE 87

XintEDIT_NONE 84, 87
XintEDIT_RUBBERBAND 87, 88
XintEDIT_SHAPE 87
XintEDIT_SIZE 87
XintEND_ARROW 140
XintFILL_COLOR 112
XintFILL_NONE 124, 126, 234
XintFILL_OPAQUE_STIPPLED 126, 272
XintFILL_SOLID 126, 239, 250, 256, 267
XintFILL_STIPPLED 126
XintFILL_STYLE 112
XintFILLED 139
XintFONT_SIZE_DEFAULT 161
XintFRONT 111
XintFRONT_TO_BACK 279, 280
XintGROUP 111
XintHALIGN_CENTER 161
XintHALIGN_LEFT 160, 161
XintHALIGN_RIGHT 161
XintHANDLE_BOUNDS 143
XintHANDLE_POINTS 143
XintHELVETICA 160, 161
XintHERSHEY_SIMPLEX 245
XintHERSHEY_TRIPLEX 244
XintHIGHLIGHT_COLOR 127
XintHIGHLIGHT_HANDLE 124, 127, 143, 229, 234, 239, 250
XintHIGHLIGHT_NONE 127
XintHORIZONTAL 110, 254, 279
XintIMAGE_FIXED 136
XintIMAGE_INTERPOLATE 135, 136
XintLABEL_X 182, 183
XintLABEL_Y 183
XintLABEL_Z 183
XintLINE_COLOR 112
XintLINE_DOUBLE_DASH 125, 127, 228, 233
XintLINE_ENDS 112
XintLINE_NONE 127, 225, 228, 232, 233, 239, 251
XintLINE_ON_OFF_DASH 125, 127, 228, 233
XintLINE_SOLID 124, 127, 228, 229, 233, 234, 256, 264, 267, 272
XintLINE_STYLE 112
XintLINE_WIDTH 112
XintLOWER 111
XintMONOCHROME 77, 78
XintMONOCHROME_MESH 275, 276
XintMOSAIC_FILL 276
XintMOVE_ANY 125, 128
XintMOVE_HORIZONTAL 128
XintMOVE_VERTICAL 128
XintNEW_CENTURY_SCHOOLBOOK 161
XintNO_ARROW 139, 140
XintNO_CONTOUR_LINE 274, 275
XintNO_FILL 80, 276
XintNO_MESH 276
XintOBJECT_EDITOR 113
XintORIENTATION _AUTOMATIC 78, 79
XintORIENTATION _LANDSCAPE 78, 79
XintORIENTATION_PORTRAIT 78, 79
XintOVERLAPPED 255
XintPASTE 111
XintPATTERN_COLOR 112
XintPIXMAP_EDITOR 113
XintPLACEMENT_BOTTOM 235, 236
XintPLACEMENT_INSIDE 236, 237, 238
XintPLACEMENT_LEFT 235, 238
XintPLACEMENT_LEFT_RIGHT 238
XintPLACEMENT_NONE 236, 238
XintPLACEMENT_RIGHT 238
XintPLACEMENT_TOP 236
XintPLACEMENT_TOP_BOTTOM 236
XintPLOT_TYPE_AREA 252
XintPLOT_TYPE_BAR 252
XintPLOT_TYPE_BAR_3D 252
XintPLOT_TYPE_HIGH_LOW 252
XintPLOT_TYPE_HISTOGRAM 252
XintPLOT_TYPE_LINE 252

Index

XintPLOT_TYPE_PIE 252
XintPLOT_TYPE_SCATTERED 252
XintPLOT_TYPE_SURFACE_3D 252
XintRAISE 111
XintROUND_MIN_MAX 235, 236, 238
XintSHADED_FILL 276
XintSHADOW_IN 233
XintSHADOW_OUT 233
XintSIMPLEX_ROMAN 242
XintSLANT_DEFAULT 161
XintSLANT_OBLIQUE 161
XintSLANT_REGULAR 160, 161
XintSTACKED 255
XintSTART_ARROW 140
XintSYMBOL 161
XintSYMBOL_CIRCLE 144, 148, 211
XintSYMBOL_DATA 144
XintSYMBOL_DIAMOND 144, 148, 211
XintSYMBOL_FILLED_CIRCLE 144, 148, 211
XintSYMBOL_FILLED_DIAMOND 144, 148, 211
XintSYMBOL_FILLED_SQUARE 144, 148, 211
XintSYMBOL_FILLED_TRIANGLE 144, 148, 211
XintSYMBOL_PLUS 143, 144, 148, 211, 268
XintSYMBOL_SQUARE 144, 148, 211
XintSYMBOL_TRIANGLE 144, 148, 211
XintSYMBOL_X 144, 148, 211
XintTICK_CROSS 228
XintTICK_INSIDE 228
XintTICK_NONE 228
XintTICK_OUTSIDE 225, 228
XintTIME_DAY 204
XintTIME_HOUR 204
XintTIME_MINUTE 204
XintTIME_MONTH 204
XintTIME_SECOND 203, 204
XintTIME_YEAR 204
XintTIMES 161
XintUNDEFINED_DOUBLE 14
XintUNDEFINED_FLOAT 14, 237
XintUNDEFINED_INTEGER 14
XintUNDEFINED_LONG 14
XintUNDEFINED_SHORT 14
XintUNGROUP 111
XintUSE_MIN_MAX 236, 238
XintVALIGN_BOTTOM 162
XintVALIGN_CENTER 162
XintVALIGN_TOP 160, 162
XintVERTICAL 110, 111, 254, 279
XintWARNING_NONE 71
XintWARNING_POST 70, 71
XintWARNING_PRINT 71
XintWEIGHT_BOLD 161
XintWEIGHT_DEFAULT 161
XintWEIGHT_MEDIUM 160, 161
XintX_VECTOR 174, 176
XintXY_PLANE 243, 245
XintXZ_PLANE 243, 245
XintY_VECTOR 176
XintYZ_PLANE 243, 245
XintZ_VALUE_MODE 259
XmUNSPECIFIED_PIXMAP 135, 249

Index

Index

Functions

XintAxisObjectPixelToUser 231
XintAxisObjectUserToPixel 230, 240
XintCGMDrawBox 72
XintCGMGetDimensions 73
XintCGMPixelToInch 73
XintCGMSetEdgeWidthMode 73
XintCGMSetLineWidthMode 74
XintCGMSetVDCType 74
XintChartAssociateData 216
XintChartCreateColorRecord 216
XintChartDisassociateAllData 216
XintChartDisassociateData 217
XintChartFreezeUpdates 217
XintChartGetComponent 217
XintChartGetDataList 218
XintChartGetDataOfSeries 218
XintChartGetSelectedComponent 219
XintChartGetSeriesOfData 220
XintChartInitializeClassConverter 220
XintChartInsertObject 220
XintChartIsTransposed 221
XintChartPick 221
XintChartReadTemplate 221
XintChartSaveTemplate 222
XintChartWidgetGetObject 223
XintChartZoom 59, 222
XintComboPlotCreateNewPlot 251
XintComboPlotGetComponent 252
XintCreateChart 216
XintCreateChartWidget 223
XintCreateDataGrid 179
XintCreateDataGroup 171
XintCreateDataLabel 186
XintCreateDataSampled 193
XintCreateDataSeries 200
XintCreateEditObject 100
XintCreateGroup 134
XintCreateImageObject 137
XintCreateLine 141
XintCreateObjectEditor 116
XintCreateOval 145
XintCreatePolyline 149
XintCreateRectangle 151
XintCreateSymbol 158
XintCreateText 163
XintDataBatchUpdate 171
XintDataGridDataReplace 180
XintDataGridGetGridArray 179
XintDataGroupFind 172
XintDataGroupIterate 172
XintDataLabelExtend 186
XintDataLabelReplace 187
XintDataLabelShift 187
XintDataRangeX 172
XintDataRangeY 172
XintDataRangeZ 172
XintDataSampledDataExtend 193
XintDataSampledDataReplace 194
XintDataSampledDataShift 195
XintDataSampledGetDataArray 194
XintDataSampledGetSampledArray 194
XintDataSeriesDataExtend 201
XintDataSeriesDataReplace 201
XintDataSeriesDataShift 202
XintDataSeriesGetXArray 202
XintDataSeriesGetYArray 202
XintDrawCursorFromData 100
XintEditObjectBack 100
XintEditObjectCopy 100
XintEditObjectCurrent 101
XintEditObjectCut 101
XintEditObjectDeselectAll 101
XintEditObjectDeselectObject 101
XintEditObjectDestroyObject 101
XintEditObjectFreeze 102
XintEditObjectFront 102
XintEditObjectGetIntersectList 102
XintEditObjectGetList 103
XintEditObjectGroup 103

XintEditObjectInsert 103
XintEditObjectLower 104
XintEditObjectManageResourceDialog 104
XintEditObjectMove 104
XintEditObjectNew 105
XintEditObjectOpen 105
XintEditObjectPaste 105
XintEditObjectRaise 105
XintEditObjectReadFile 105, 108
XintEditObjectSave 106
XintEditObjectSaveAs 106
XintEditObjectSelectAll 106
XintEditObjectSelectList 106
XintEditObjectSelectObject 106
XintEditObjectSetEditMode 107
XintEditObjectSize 107
XintEditObjectUngroup 107
XintEditObjectWriteFile 107
XintGetWidgetSize 75
XintGraphicGetViewPortList 131
XintGraphicUnmanageDialog 131
XintHorizontalPixelToUser 75
XintHorizontalUserToPixel 75
XintIsAxisObject 231
XintIsBar3D 260
XintIsBarLine 256
XintIsCellArray 249
XintIsChart 222
XintIsComboPlot 252
XintIsDataGrid 180
XintIsDataGroup 172
XintIsDataLabel 187
XintIsDataSampled 195
XintIsDataSeries 202
XintIsGraphic 131
XintIsGroup 134
XintIsHighLow 263
XintIsHistogram 266
XintIsImageObject 137
XintIsLegend 234
XintIsLine 141
XintIsOval 145
XintIsPie 271
XintIsPolyline 149
XintIsRectangle 151
XintIsSurface3D 277
XintIsSymbol 158
XintIsText 163
XintIsXYPlot 280
XintObjectEditorAddEditObjectToList 119
XintObjectEditorGetDefinedPixmap 116
XintObjectEditorRemoveEditObjectFromList 119
XintOutputCGM 76
XintOutputMontageCGM 76
XintOutputMontagePostscript 77
XintOutputPostscript 78
XintPieExplodeWedge 271
XintPostscriptGetDefaults 79
XintPostscriptSetBackground 80
XintPostscriptSetDefaults 80
XintResEditGetBox 36
XintResEditGetLabel 37
XintResEditGetMenuButton 37
XintSymbolCreate 158
XintSymbolFree 158
XintVerticalPixelToUser 80
XintVerticalUserToPixel 80

Index

Resources
XintHERSHEY_SIMPLEX 244
XintHERSHEY_TRIPLEX 245
XmN3dPerspectiveDepth 242
XmN3dRotation 242
XmN3dScale 242
XmNactionList 110
XmNactionNumColumns 110
XmNactionOrientation 110
XmNallowDrag 84
XmNallowDrop 84
XmNannotationAngle 225
XmNannotationFont 225
XmNannotationFormat 225
XmNannotationPlacement 225
XmNannotationStrokeFont 242
XmNannotationStrokeFontSize 242
XmNareaSelectionCallback 84, 93, 94
XmNarrowLength 139
XmNarrowStyle 139
XmNattributeList 110
XmNattributeNumColumns 110
XmNattributeOrientation 110
XmNaxisLineThickness 225
XmNaxisSpacing 207
XmNbarColorMode 259
XmNbarLength 266
XmNbarLineColor 259
XmNbarLineThickness 259
XmNbarOrientation 254
XmNbarStyle 254
XmNbaseAngle 139
XmNchartFooter 207
XmNchartMargins 207
XmNchartTitle 207
XmNchartType 207
XmNclipGrid 124
XmNcloseEndPoints 147
XmNclusterHeight 259
XmNclusterWidth 254, 259
XmNcolor 124, 229, 234, 239, 250, 257, 264, 267, 272, 281
XmNcolorAttributeList 110
XmNcolorList 110, 207
XmNcolorNumColumns 110
XmNcolorOrientation 110
XmNcolumns 232
XmNconstraint 208, 213
XmNcontourLineColor 274
XmNcontourLineMode 274
XmNcontourLineThickness 274
XmNcopyCallback 84, 93, 95
XmNcopyData 174, 182, 189, 197
XmNcount 189, 197
XmNcursorType 84
XmNcutCallback 84, 93, 95
XmNdashList 124, 229, 234, 239, 250, 257, 264, 267, 272, 281
XmNdataArray 189
XmNdataGroup 174, 182, 189, 197
XmNdataType 174, 189, 197
XmNdisplayName 124
XmNdisplayPatterns 248
XmNdoubleBuffer 207
XmNdragDropCallback 84, 93, 95
XmNdrawCandlestick 262
XmNdrawFrame 235
XmNdrawShadow 254, 259, 270
XmNdrawSymbolCallback 147, 148, 281
XmNeditable 174, 189, 197
XmNeditList 110
XmNeditObjectCallback 84, 93, 94
XmNeditObjects 110
XmNendPoints 225
XmNfillColor 124, 234, 239, 250, 257, 267, 272, 281
XmNfillFilename 124, 234, 239, 250, 257, 267, 272, 281
XmNfillMode 274
XmNfillPixmap 124, 234, 239, 250, 257,

267, 272, 281
XmNfillStyle 124, 234, 239, 250, 257, 267, 272, 281
XmNflip 84
XmNfont 124, 229, 234, 239, 250
XmNfontFamily 160
XmNfontPath 70
XmNfontSize 160
XmNfontSlant 160
XmNfontWeight 160
XmNfreePixmap 135
XmNgainColor 264
XmNgeometry 207
XmNgridArray 174
XmNgridOrder 174
XmNgroup 124
XmNhandleColor 84
XmNhandleMode 143
XmNhandleSize 84
XmNhighlightMode 124, 229, 234, 239, 250
XmNhistoryLength 167
XmNhorizontalAxis 124
XmNhorizontalTextAlignment 160
XmNiconWidth 232
XmNimageColorRecord 135
XmNimageDisplayMode 135
XmNimagePixmap 135
XmNinclination 254, 270
XmNincrements 225
XmNinsertObjectCallback 84, 93, 96
XmNlabel 225
XmNlabelCount 182
XmNlabelFont 225
XmNlabelGravity 154
XmNlabelOrientation 182
XmNlabelPositionArray 182
XmNlabelSpacing 154
XmNlabelStrings 182
XmNlabelStrokeFont 242
XmNlabelStrokeFontSize 242
XmNlastViewDestroy 167, 174, 182, 189, 197
XmNlegendAnnotationFormat 248
XmNlegendLocation 232
XmNlegendTitle 232
XmNlimits 225
XmNlimitsX 167
XmNlimitsY 167
XmNlimitsZ 167
XmNline 139
XmNlineEnd 139
XmNlineStyle 124, 229, 232, 234, 239, 251, 257, 264, 267, 272, 281
XmNlineThickness 125, 229, 234, 239, 251, 257, 264, 267, 272, 281
XmNlist 134
XmNlistCount 134
XmNlocatorCallback 84, 93, 97
XmNlogScale 225
XmNlossColor 264
XmNmajorGridLineStyle 225
XmNmarginHeight 160, 232
XmNmarginWidth 160, 232
XmNmeshColor 274
XmNmeshDrawX 274
XmNmeshDrawY 274
XmNmeshLineThickness 275
XmNmeshMode 275
XmNminimumLabelSpacing 225
XmNminorGridLineStyle 225
XmNmove 125, 229, 234, 239, 251
XmNmoveDirection 125
XmNname 125
XmNnullValue 147
XmNnumColumns 110
XmNnumEditObjects 110
XmNnumGridX 275
XmNnumGridY 275
XmNnumGridZ 275
XmNobjectClassList 110
XmNobjectDeselectionCallback 84, 93, 96

Index

XmNobjectEditMode 84
XmNobjectSelectionCallback 84, 93, 96
XmNorientation 110
XmNpasteCallback 84, 93, 95
XmNpatternList 248
XmNperspectiveDepth 254, 270
XmNpieSize 270
XmNpixmapList 111
XmNpixmapNumColumns 111
XmNpixmapOrientation 111
XmNplotOrientation 279
XmNpointArray 147
XmNpointCount 147
XmNpointSelectionTolerance 85
XmNpropagate 134, 207, 235
XmNproportional 254
XmNrectangle 150
XmNresourceDialog 125
XmNresourceDialogCallback 85, 93, 97
XmNreverseOrder 225
XmNrotateAngle 150, 160
XmNrotation 254
XmNroundEdge 150
XmNrubberbandCallback 85, 93, 98
XmNsampledRange 182, 189
XmNselectionCallback 85, 93, 94
XmNsensitive 125, 229, 234, 239, 251, 257, 264, 267, 272, 281
XmNshape 125, 229, 234, 239, 251
XmNshowAttributeLabels 111
XmNshowClose 262
XmNshowCumulative 266
XmNshowDistribution 266
XmNshowGainLoss 262
XmNshowLegend 207
XmNshowOpen 262
XmNshowPieName 270
XmNshowWedgeLabels 270
XmNsolidSurface 275
XmNstackingOrder 279
XmNstippleColor 125, 229, 234, 239, 251, 257, 267, 272, 281
XmNsurfaceBottomColor 275
XmNsurfaceTopColor 275
XmNsymbol 143, 257, 268, 279, 281
XmNsymbolColor 143, 257, 268, 281
XmNsymbolCount 207
XmNsymbolData 143, 154
XmNsymbolHeight 154
XmNsymbolList 207
XmNsymbolLocation 154
XmNsymbolScale 154
XmNsymbolSize 143, 257, 268, 279, 281
XmNsymbolType 143, 257, 268, 281
XmNsymbolWidth 154
XmNtextLocation 160
XmNtextScale 160
XmNtextString 160
XmNtickLength 262
XmNtickPlacement 225
XmNtimeBase 203
XmNtimeLabelFormat 203
XmNtimeMeasure 203
XmNtipAngle 139
XmNtranspose 207
XmNupdateCallback 174, 182, 189, 197
XmNuserData 125
XmNverifyCallback 125, 130, 145
XmNverticalAxis 125
XmNverticalTextAlignment 160
XmNviewScale 242
XmNvisible 125
XmNvisibleEntryCount 232
XmNwallColor 243
XmNwarning 70
XmNxAnnotation 242
XmNxAnnotationFormat 242
XmNxArray 197
XmNxAutoRangeMode 235
XmNxAxisOffset 242

XmNxAxisPlacement 235
XmNxCount 174
XmNxGridLines 243
XmNxIncrements 235, 243
XmNxInsidePlacement 235
XmNxLabel 243
XmNxLimits 235, 243
XmNxRange 174
XmNxTranslation 243
XmNyAnnotation 242
XmNyAnnotationFormat 242
XmNyArray 197
XmNyAutoRangeMode 235
XmNyAxisOffset 243
XmNyAxisPlacement 235
XmNyCount 174
XmNyGridLines 243
XmNyIncrements 235, 243
XmNyInsidePlacement 235
XmNyLabel 243
XmNyLimits 235, 243
XmNyRange 174
XmNyTranslation 243
XmNzAnnotation 242
XmNzAnnotationFormat 242
XmNzAxisOffset 243
XmNzGridLines 243
XmNzIncrements 243
XmNzLabel 243
XmNzLimits 243
XmNzValueColorRecord 207