Edit
Table

EditTable 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

Fourth printing

January 2009

Contents

How To Use This Manual

Overview

The INT EditTable Widget Library is designed for use by software developers and C or C++ programmers who want a fast, easy way to display and manipulate data in an X Window/Motif environment. Before you get started, read the following sections to acquaint yourself with manual organization and typographical conventions.This programming guide explains how to create application programs using the Xint EditTable Widget Library.

This chapter includes the following sections:

- **Organization of This Manual** on page x
- **Notation Conventions** on page x

Organization of This Manual

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 Edittable and related widgets.
- Use *Chapter 2—CompBase Widget Metaclass*, *Chapter 3—EditObject Widget Class*, *Chapter 4—EditTable Widget*, and *Chapter 5—Scroll Widget Class* to learn about specific callbacks, resources, actions, and functions.

  Note: Use the index to locate specific callback instructions.

- Use *Chapter 6—Examples* to learn how INT widget components are used in real-life programming situations to produce the desired effects.

You can use these chapters in any order, however, we recommend that you read them in sequence.

Chapter 6—Examples contains examples of programs that use the Xint EditTable Widget Library. We suggest that you work through the examples using the supplied example source code programs.

Notation Conventions

There are several conventions used in this programming guide to represent keyboard commands, menu commands, and widget-feature identifiers. You must become familiar with them in order to use this guide correctly.

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 pushbutton, 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.

Introduction

1

Overview

This chapter provides a general description of the INT EditTable Widget Library's architecture, contents and functionality, and includes the following sections:

EditTable Widget

The EditTable widget is a set of programming tools that help you create advanced spreadsheet and table-oriented data displays in the X Window/Motif environment. EditTable facilitates control of all aspects of a table display, from simple arrangement of titles, annotations, margins, color, and shading, to complex aspects of data formatting, validation, and hardcopy output.

Widget Components

EditTable is one of several widgets derived from the Motif XmManager widget class. Used together, these widgets provide not only advanced table features, but the ability to display and edit graphic objects, produce charts, and generate hardcopy in many popular formats. The EditTable widget has the following standard components:

- *Xint resources* let you control the widget's "look and feel."
- *Xint convenience functions* allow applications to directly populate, query, and control the data and format of the table.
- *Xint action routines* help control cursor and pointer movements and the selection, deletion, or saving of objects.
- *Xint callbacks* provide ways to modify data tables in response to user actions.

EditTable provides over 100 resources (in addition to Motif base class resources), over 50 action routines, over 100 functions, and many callbacks. All are designed to provide familiar tools you will recognize, without the time-consuming complexity of X/Motif programming.

EditTable Features

The EditTable widget is designed to provide the following specific features and benefits:

- Rapid development of portable applications.
- Full graphics editing capability in addition to graphics display capability.
- High-level building blocks reusable in a wide variety of applications.
- Built-in hardcopy support so an application programmer doesn't have to worry about hardcopy implementation.
- Object oriented programming design.
- Insulate application programmers from the complexity of Xlib.

- Decrease application design errors and programming errors.
- Decrease the cost of application enhancements and debugging.
- Allow inexperienced X/Motif programmers to successfully develop maintainable applications that meet user requirements.
- Combine broad widget functionality with ease of use.
- Provide all resources and functions needed to control any aspect of a object's behavior and appearance.
- Provide all resources, actions and dialog boxes needed by end users to control object behavior and appearance, thereby freeing the application programmer to spend time developing other aspects of the application.

Data Handling

The following sections describe the types of data handled by EditTable, and how data is formatted by EditTable.

Data Types

EditTable can handle a variety of data sources and data types. Each column has an associated data type set with resource **XmNcolumnDataTypeData** or function XintEditTableDefineColumnFormat. If no column type has been specified, the default applies. This is set with resource **XmNdefaultColumnDataType**.

Table data values

Data values in a table can be any of the following types:

- Integers (XintTYPE_INTEGER)
- Short integers (XintTYPE_SHORT)
- Long integers (XintTYPE_LONG)
- Floating point numbers (XintTYPE_FLOAT)
- Double-precision floating point numbers (XintTYPE_DOUBLE)
- Character strings (XintTYPE_STRING)

Note: For these data types, the data format must be uniform within a column. However, EditTable also supports the pointer data type which lets you use composite data in applications.

Data Formatting

The formatting of the cell data into a string is done automatically based on the data type specified for each column. A C type format specifier is used to control the formatting (see resources **XmNdefaultColumnDataFormat** on page 96 and **XmNcolumnDataFormatData** on page 94).

The application should make sure that the data format specifier and data type are compatible for each column. The application can also use the callback XmNformatCellCallback to specify non standard formats (for example to represent negative numbers inside brackets).

EditTable also supports the storage of pointer (XintTYPE_POINTER) data where the formatting of the cell is done via the callback XmNformatCellCallback only.

The table can be populated on a per cell basis, using function XintEditTableFillCell or on a column basis using function XintEditTableFillColumn.

Flexible Data Handling

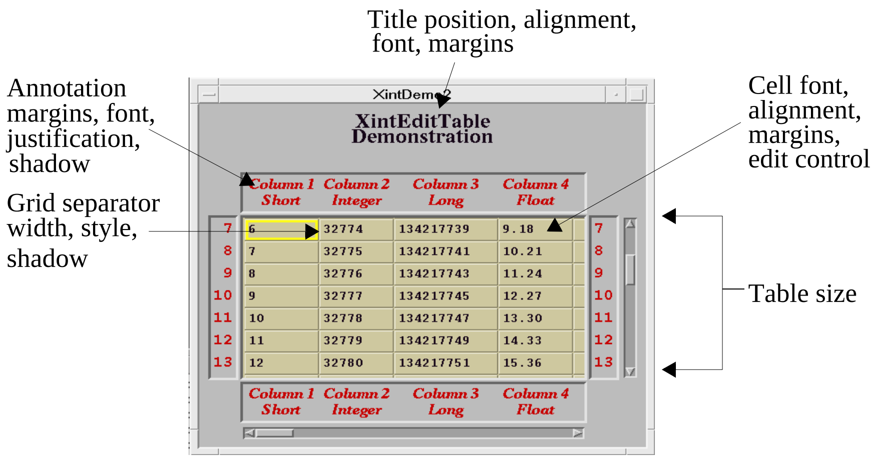
By default, EditTable makes a copy of the data. This behavior can be disabled by setting the resource **XmNuseOriginalData** to True at widget creation time. For example, you could have a table directly access the same memory area as a displayed plot, so that any changes to the plot would immediately update the table and vice versa.

Dataless Tables

EditTable table supports a column type XintTYPE_NONE where the table will request its data as a string using the callback XmNformatCellCallback. In this last case, changes in the data should be notified to the EditTable using the function XintEditTableUpdateDataDisplay.

Basic Features

This section discusses the basic features provided by EditTable and its related widgets, as shown in Figure 1. Refer to *Chapter 4—EditTable Widget* for more detailed information.


Title position, alignment, font, margins
Annotation margins, font, justification, shadow
Grid separator width, style, shadow
Cell font, alignment, margins, edit control
Table size
XintDemo2

XintEditTable Demonstration

	Column 1 Short	Column 2 Integer	Column 3 Long	Column 4 Float	
7	6	32774	134217739	9.18	7
8	7	32775	134217741	10.21	8
9	8	32776	134217743	11.24	9
10	9	32777	134217745	12.27	10
11	10	32778	134217747	13.30	11
12	11	32779	134217749	14.33	12
13	12	32780	134217751	15.36	13
	Column 1 Short	Column 2 Integer	Column 3 Long	Column 4 Float	

Figure 1. Understanding the Basic Features

Margins and Table Dimensions

You can control all aspects of table size, margins, and dimensions. EditTable allows you to perform the following actions, dynamically:

- Add columns or rows.
- Resize column width or row height.
- Set precise margins for titles, annotation areas, or cells.

Margins can be sized automatically or based on resources provided by either the XmManager widget or the Scroll widget (XintScroll). Total table size is limited only by the size of available memory.

Text Alignment, Size, and Font Type

EditTable gives you complete control over the positioning, size, color and font of all text components, including:

- Table title
- Cells
- Row/column annotation

For instance, the title can be at any location on the table (top/bottom/left/right). Annotations can be on either side or both sides of the table, with any desired justification inside the annotation area.

Text Editing

The EditTable widget supports a large number of editing operations that can either be controlled by the application using INT functions or performed by the end user via action routines. Users of applications built on EditTable can edit cells or row/column annotations directly in a full text editing mode (i.e., click, shade, backspace, retype). The widget supports the following features:

- Insert/delete rows or columns
- Undelete previously deleted rows or columns
- Reverse row/column sequence
- Cut, paste, or copy rows, columns, or ranges of cells
- Change all selected cells to a specified value

Any deleted or copied rows or columns are stored on a clipboard that can simultaneously hold both columns and rows. When another copy or delete operation occurs, the data copied or deleted replaces the same type of data (row or column) currently on the clipboard. Data on the clipboard can be pasted into the table using convenience functions and action routines.

Column/Row Formatting

A column can be modified by changing any of the following:

- Format of data in the column
- Width of the column
- Number of rows
- Text font
- Foreground/background colors

EditTable allows format changes by the application or the user. For example, the user may want prices to be displayed using dollar signs, commas, and decimals, even though the price data is stored without these elements.

Cursor Movement

Users can traverse a table using the keyboard (Tab, cursor keys, etc.) or by using the mouse to click directly on a cell. The application can control keyboard traversal through a callback so that the cursor skips over certain parts of the table that are irrelevant to the current operation. You can also build dialogs that let end users “go to” a specific row/column in the table or to a specified percent of the width/height of the table.

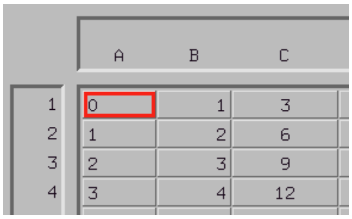
Advanced Features

This following sections discuss several advanced features provided by the EditTable and related widgets.

User-defined or Automatic Annotation

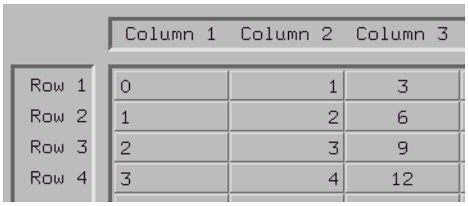
Row/column annotations can be user-defined or automatically labelled with numeric or alphabetic sequences. Columns can be annotated in spreadsheet fashion with character strings that are automatically generated in alphabetical order (A, B, C,..., AA, AB, etc.). Rows can be annotated automatically with sequential integers or any of the other supported data types (float, character, and so forth).

The application can apply the annotation value for a column or row as it is created or after it is created. Figure 2 shows an example of a table with auto-annotation and an example of a table with user-defined annotation:



	A	B	C
1	0	1	3
2	1	2	6
3	2	3	9
4	3	4	12

Table With Auto Annotation



	Column 1	Column 2	Column 3
Row 1	0	1	3
Row 2	1	2	6
Row 3	2	3	9
Row 4	3	4	12

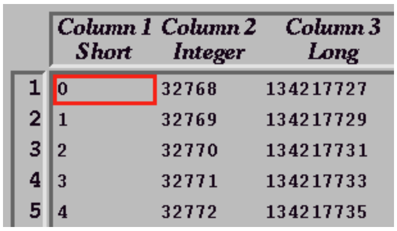
Table With User-defined Annotation

Figure 2. Table Annotation Options

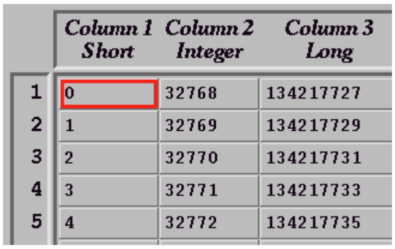
Inter-cell Grid Separators

For more visually sophisticated displays, EditTable allows you to add grid line separators (Figure 3) with specifiable features such as:

- Line thickness (in pixels)
- Line color and style (solid, dashed, etc.)
- 3D effect (shadow in/shadow out)
- Selectable orientation (row, column, or both)



	Column 1 Short	Column 2 Integer	Column 3 Long
1	0	32768	134217727
2	1	32769	134217729
3	2	32770	134217731
4	3	32771	134217733
5	4	32772	134217735

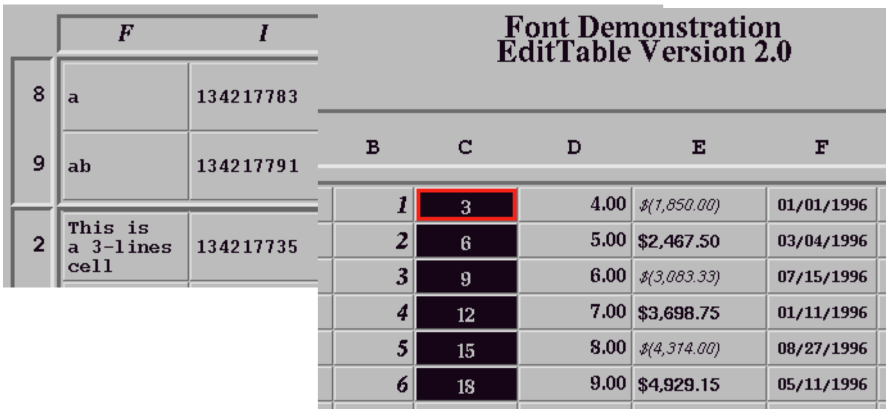


	Column 1 Short	Column 2 Integer	Column 3 Long
1	0	32768	134217727
2	1	32769	134217729
3	2	32770	134217731
4	3	32771	134217733
5	4	32772	134217735

Figure 3. Cells With/Without Grid Line Separators

Micro-formatting

EditTable supports color or text font changes at the row level, column level, or cell-by-cell, as shown in Figure 4. This provides a way to highlight important data such as totals or constraints so that it stands out in the table. Individual cells can contain multiple lines of data, if desired.



	F	I
8	a	134217783
9	ab	134217791
2	This is a 3-lines cell	134217735

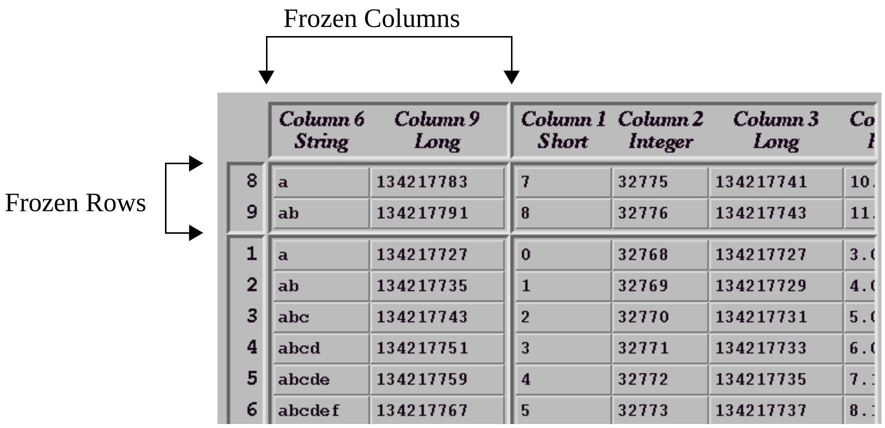
Font Demonstration
EditTable Version 2.0

B	C	D	E	F
1	3	4.00	$(1,850.00)	01/01/1996
2	6	5.00	$2,467.50	03/04/1996
3	9	6.00	$(3,083.33)	07/15/1996
4	12	7.00	$3,698.75	01/11/1996
5	15	8.00	$(4,314.00)	08/27/1996
6	18	9.00	$4,929.15	05/11/1996

Figure 4. Varied Text Fonts and Multi-line Cells

Row/Column Freezing

EditTable allows freezing of any sequential or non-sequential group of rows and/or columns, as shown in Figure 5. Freezing causes the rows or columns to remain fixed in the same position so that they do not scroll with the rest of the table. Frozen rows can be placed at the top or bottom of the table; frozen columns at the left or right. Cells in a frozen column can still be selected and edited, if desired. Frozen rows or columns can be unfrozen when desired.

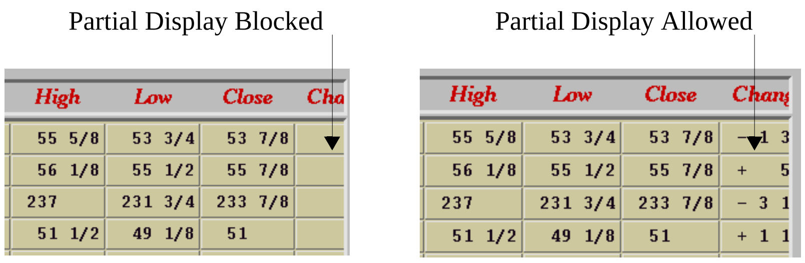


	Column 6 String	Column 9 Long	Column 1 Short	Column 2 Integer	Column 3 Long	Co
8	a	134217783	7	32775	134217741	10.
9	ab	134217791	8	32776	134217743	11.
1	a	134217727	0	32768	134217727	3.
2	ab	134217735	1	32769	134217729	4.
3	abc	134217743	2	32770	134217731	5.
4	abcd	134217751	3	32771	134217733	6.
5	abcde	134217759	4	32772	134217735	7.
6	abcdef	134217767	5	32773	134217737	8.

Figure 5. Example of Frozen Columns and/or Rows

Row/Column Visibility and Partial Displays

To provide additional control over data presentation, EditTable allows the visibility of any column or row to be selectively turned on or off (Figure 6). Columns or rows that are turned off disappear completely from the display, but remain active in the background until turned on again. EditTable also allows you to decide whether or not to display data in columns that are only partially visible in the viewing area. This feature is valuable for financial displays or other applications where a partial display of prices or other data could lead to misquotes or other potential misinterpretation.



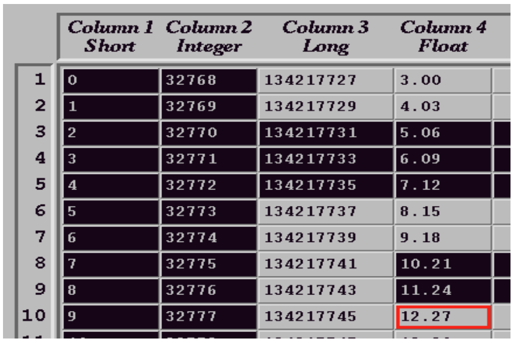
High	Low	Close	Cha
55 5/8	53 3/4	53 7/8	
56 1/8	55 1/2	55 7/8	
237	231 3/4	233 7/8	
51 1/2	49 1/8	51	

High	Low	Close	Chang
55 5/8	53 3/4	53 7/8	−1 3
56 1/8	55 1/2	55 7/8	+ 5
237	231 3/4	233 7/8	− 3 1
51 1/2	49 1/8	51	+ 1 1

Figure 6. Columns With/Without Displayed Data

Free-style Selection

EditTable allows users to select any combination of columns, rows, or individual cells, whether or not the selected components are adjacent (Figure 7). You can extend selections indefinitely to add rows, columns, or cells to the initially selected area. Selected rows, columns, or cells can be set off by a special background color and/or a highlighted border of a specified thickness and color, and cut, copied, or pasted as a group.



	Column 1 Short	Column 2 Integer	Column 3 Long	Column 4 Float
1	0	32768	134217727	3.00
2	1	32769	134217729	4.03
3	2	32770	134217731	5.06
4	3	32771	134217733	6.09
5	4	32772	134217735	7.12
6	5	32773	134217737	8.15
7	6	32774	134217739	9.18
8	7	32775	134217741	10.21
9	8	32776	134217743	11.24
10	9	32777	134217745	12.27

Figure 7. Row, Column, and Random Selections

Horizontal Columns

The columns in a table can be oriented vertically or horizontally. Vertical is the standard orientation, but in some applications a horizontal orientation may be useful. When the columns are oriented horizontally, the cells in a column are drawn from left to right and the column annotation appears on the left or the right of the column. All column and row operations work the same, regardless of column orientation.

Icon Selection and Display

EditTable lets you superimpose an icon or pixmap as the background for any range of cells. This can include company logos or special pictorial elements that can be more easily interpreted than raw data. An XWD image, X bitmap, or other bitmapped symbol can be converted to a pixmap and inserted into a cell. The application can dynamically control the use of bitmapped symbols as background for table cells, as shown in Figure 8:

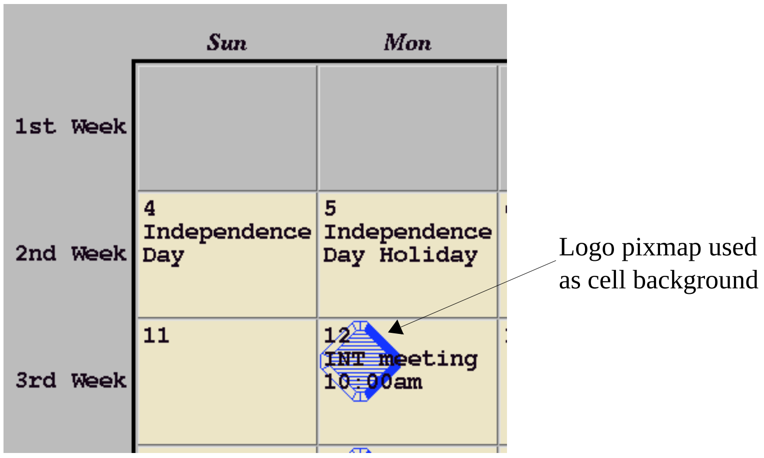


Figure 8. Example of Pixmap Background

Cell Validation and Locking

EditTable includes advanced input validation features. For example, you could trap input that does not match the default data type and display a warning dialog for users. Additionally, you can check that data is within specified ranges. Validation can occur immediately on a cell-by-cell basis or for the entire table. The application can allow the change, prohibit the change, or substitute its own value.

EditTable also supports read-only protection of cells, so that the user can be locked out of certain rows, columns, or individual cells. Since the widget notifies the application of cursor movement, it can flag any situation where the user tries to edit read-only cells.

Cell Attributes

Attributes such as cell foreground and background, pixmap and font can be set on a cell basis. A set of functions, such as XintEditTableSetCellForeground or XintEditTableSetCellBackground is provided to set those attributes for a cell or a block of cells. This information is stored inside the table. For large tables or when memory usage is a concern, the callback XmNcellAttributesCallback can be used to provide cell attributes directly at cell repaint time. For example this feature can be used to set the cell color when it is a function of the data, row or column number.

Note: To make a cell flash for a specified period of time, use function XintEditTableCellFlash.

The function XintEditTableSetCellDisplayAttributes can be used to update the foreground, background and data content of a range of cells in one atomic operation.

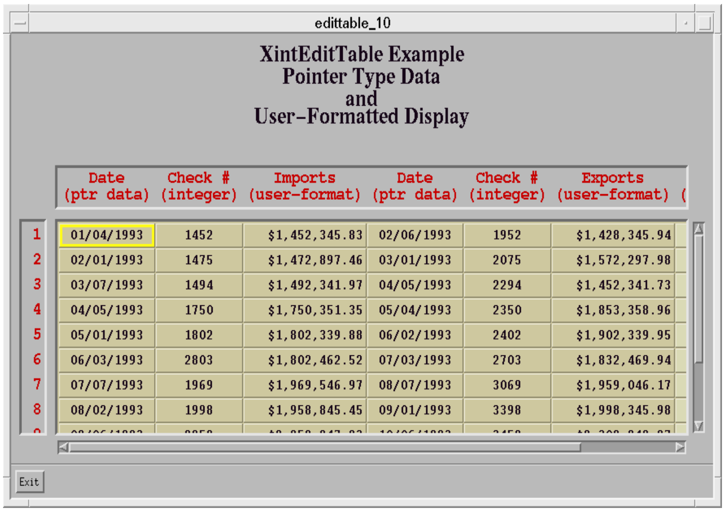
Displaying Composite Data (Pointer Data Type)

EditTable provides a *pointer* data type that lets you set up specific columns in a table so they receive application-defined data (structures, arrays, or other tables). This allows you to display composite items such as date and time or a value calculated as the result of operations on another table. For example, the date may be derived from an array {1993, Dec, 15} but displayed in an alternate format (12-15-93). The application uses a callback to handle the formatting process.

User-defined Data Formatting

EditTable provides another callback feature that lets users select their own custom formatting for data columns. For example, the application might store the data

123456.78 as data type “double,” but display it with dollar signs and commas so that the data reads: $123,456.78, as shown in Figure 9:



XintEditTable Example
Pointer Type Data
and
User–Formatted Display

	Date (ptr data)	Check # (integer)	Imports (user-format)	Date (ptr data)	Check # (integer)	Exports (user-format)
1	01/04/1993	1452	$1,452,345.83	02/06/1993	1952	$1,428,345.94
2	02/01/1993	1475	$1,472,897.46	03/01/1993	2075	$1,572,297.98
3	03/07/1993	1494	$1,492,341.97	04/05/1993	2294	$1,452,341.73
4	04/05/1993	1750	$1,750,351.35	05/04/1993	2350	$1,853,358.96
5	05/01/1993	1802	$1,802,339.88	06/02/1993	2402	$1,902,339.95
6	06/03/1993	2803	$1,802,462.52	07/03/1993	2703	$1,832,469.94
7	07/07/1993	1969	$1,969,546.97	08/07/1993	3069	$1,959,046.17
8	08/02/1993	1998	$1,958,845.45	09/01/1993	3398	$1,998,345.98

Figure 9. Pointer Type Data in User-formatted Display

Cell Spanning

EditTable provides a method to allow cells to span more than one row/column location in both the row and column directions. This span can be applied to all of the cells in the table, or a subset of the table locations.

Importing Data from ASCII Files

EditTable provides a mechanism for populating tables with data contained in ASCII files. The widget provides both a resource and a function for reading delimited ASCII files into a table. Both tab-separated and comma-separated (CSV) files can be read directly into the table. Each line in the ASCII file corresponds to a row in the table. The cell contents of a row correspond to the file entries between delimiters. Incoming data for a cell is converted to the specified column data type as the file is being read; if this conversion fails, the cell is left unchanged. Controls are provided for specifying alternative field delimiters. It is also possible to cause EditTable to use the first column of the file as the row annotation; similarly the first row of the file can be used as the column annotation.

Interactive Move, Copy, and Resize

EditTable provides a set of actions for interactively moving, copying and resizing rows, columns and cells. The move and copy actions let users reposition rows and columns within the table by selecting them and dragging them to the desired location using a mouse. Resizing is similar in that the user drags the row, column, or cell boundary until it becomes the desired size. EditTable supplies functions for inquiring the cell height and width after interactive resizing.

You can specify the cursor type displayed during these operation. Additionally, you can show either the *contents* of the cells being moved or just an *outline* of the cells. In either case, you can specify the line style and the color of the outline of the cells which the user is dragging. The application can also specify a callback to be invoked after the cells have been resized or dragged to their new location.

Multiple Output Formats

EditTable provides several hardcopy and exchange formats for output You can output an entire table or a range of cells in these formats:

- **ASCII**
  This is a plain text format often used to transfer data between applications. Columns may be separated by spaces or by a delimiter character such as a comma or tab.
- **SYLK**
  This is a spreadsheet exchange format that offers a variety of font options, for specially designed tables that need to match a specific presentation environment.
- **PostScript**
  EditTable provides color or monochrome PostScript output of the entire table or any sub-table. You can specify an output scale that controls the size of final output, plus specify resolution (pixels/in), page width (in), and page height (in).
- **CGM**
  EditTable provides binary CGM output, a common scalable format used in presentations and desktop publishing.

Motif Drag and Drop

EditTable 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 into another table, from a text widget into a table, etc. If you are using ChartObject, it is also possible to drag and drop data from a table to a chart and from a chart to a table.

EditObject action MotifStartDrag controls the drag operation, and connects with the following translation:

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

Note: No specific action is 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. This callback can also be used to modify the type of drag and drop operation (copy/move/link) and modify the index or count of the cells being dragged.

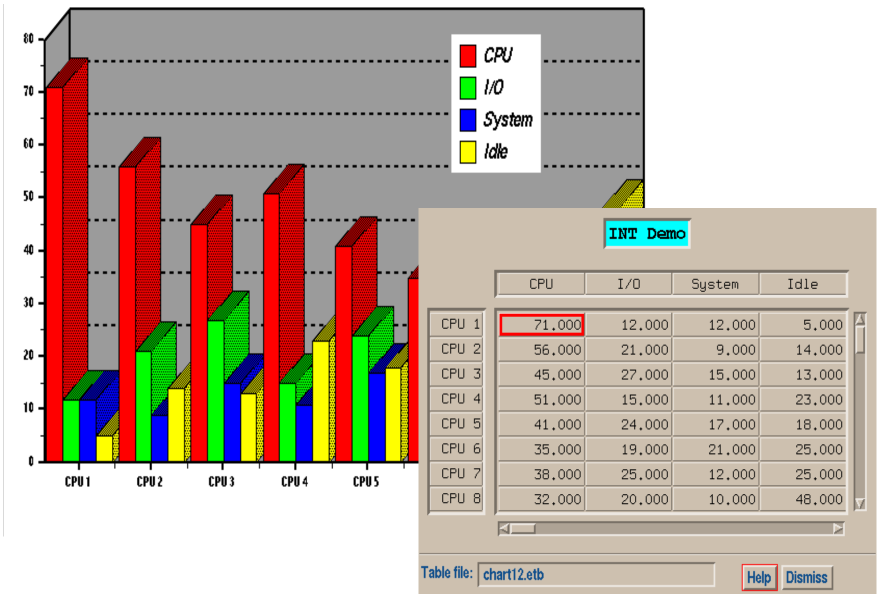
Default behavior for drag and drop

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>	EditTable	EditTable	Copies the selected cells from the source table to the destination table.
<Btn2Down>	EditTable	Chart object	The data contained in the selected cells is copied into the chart object (move option is disabled for the EditTable widget).
Ctrl <Btn2Down>	EditTable	Chart object	The data contained in the selected cells is copied into the chart object.
Ctrl Shift <Btn2Down>	EditTable	Chart object	The data contained in the selected cells is copied into the chart object and a link is made between the table and the chart.
<Btn2Down>	Chart object	EditTable	The data contained in the chart is moved to the table at the location specified by the pointer.
Ctrl <Btn2Down>	Chart object	EditTable	The data contained in the chart is copied to the table at the location specified by the pointer.
Ctrl Shift <Btn2Down>	Chart object	EditTable	The data contained in the chart is copied to the table at the location specified by the pointer. A link is established between the chart and the table.
<Btn2Down>	Motif Text	EditTable	Content of the Text widget is copied into the specified table cell.

DataObject Connection

The ChartObject library provides a library of data objects called DataObject library. This library defines the *model* component of the ChartObject Model View Controller architecture. When used in conjunction with ChartObject, EditTable can become another view of the data by associating a Data object to the EditTable widget using function XintEditTableAssociateData. Figure 10 illustrates an example where a chart object and an EditTable widget are both associated with the same data object:

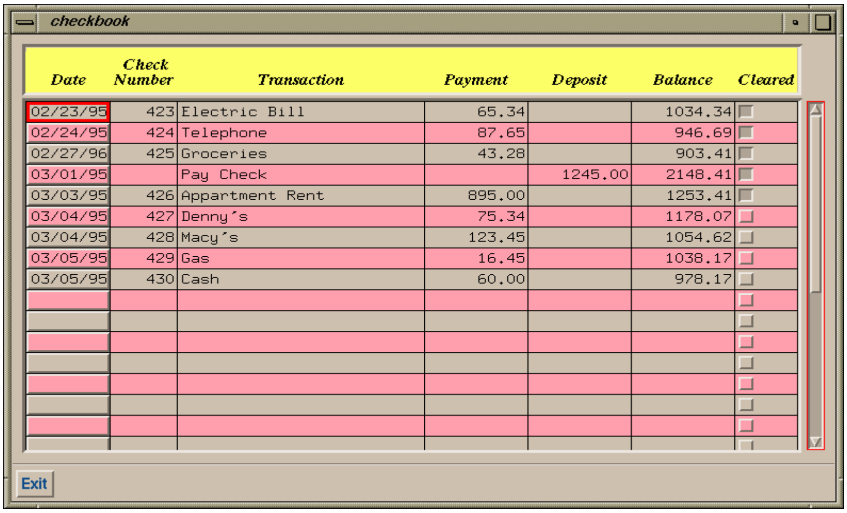


	CPU	I/O	System	Idle
CPU 1	71.000	12.000	12.000	5.000
CPU 2	56.000	21.000	9.000	14.000
CPU 3	45.000	27.000	15.000	13.000
CPU 4	51.000	15.000	11.000	23.000
CPU 5	41.000	24.000	17.000	18.000
CPU 6	35.000	19.000	21.000	25.000
CPU 7	38.000	25.000	12.000	25.000
CPU 8	32.000	20.000	10.000	48.000

Figure 10. Multiple Views of a Data Object

Widget in a Cell

The EditTable widget offers a simple yet powerful mechanism to insert a widget inside a cell or to use a single widget across an entire range of cells. Any valid Motif widget, with the exception of composite widgets such as RowColumn or Form widgets, can be inserted, so long as the cell or cells are not frozen. Gadgets are not supported. To insert a widget into one or more cells (Figure 11), the application simply creates the widget with EditTable as the parent and uses constraint resource **XmNcellWidgetRange** to specify the range of cells that are to contain this widget. Figure 11 illustrates a push button widget and a toggle button widget, each propagated to an entire column in an electronic checkbook:



Date	Check Number	Transaction	Payment	Deposit	Balance	Cleared
02/23/95	423	Electric Bill	65.34		1034.34	■
02/24/95	424	Telephone	87.65		946.69	■
02/27/96	425	Groceries	43.28		903.41	■
03/01/95		Pay Check		1245.00	2148.41	■
03/03/95	426	Appartment Rent	895.00		1253.41	■
03/04/95	427	Denny's	75.34		1178.07	☐
03/04/95	428	Macy's	123.45		1054.62	☐
03/05/95	429	Gas	16.45		1038.17	☐
03/05/95	430	Cash	60.00		978.17	☐
						☐
						☐
						☐
						☐
						☐
						☐
						☐

Figure 11. Example of a Widget In a Cell Application

Cell Resources

It is important for the contents of a widget to reflect the visual characteristics that are to be displayed in the cell before the cell that contains the widget is drawn. EditTable will automatically set the cell resources listed below on the widget if the constraint resource **XmNcellWidgetSetResources** is set to True. This occurs before the

XmNcellWidgetDisplayCallback callback is called.

Resource Name	**Automatic Setting**
XmNlabelString	Current cell content, as a string.
XmNfontList	Current cell font.
XmNbackground	Current cell background.
XmNforeground	Current cell foreground.
XmNalignment	Current cell alignment.
XmNsensitive	Current cell sensitivity.

The values of these resources are available in the callback structure XintEditTableCellWidgetCallbackStruct. If **XmNcellWidgetSetResources** is False, or it is necessary to change one or more values, this can be done in the callback. The callback also permits other widget resources, which are not in the list above, to be set.

If **XmNcellWidgetSetResources** is False and **XmNcellWidgetDisplayCallback** is not called, the visual characteristics of the cells will be undefined!

In most applications it will be sufficient to set **XmNcellWidgetSetResources** to True and omit the **XmNcellWidgetDisplayCallback**. The ToggleButton widget is an example of an instance where **XmNcellWidgetDisplayCallback** is necessary in order to get information about the cell state to use in setting the XmNset resource.

Due to the power and speed of the Widget in a Cell mechanism, The XmString data type (to enable internationalization of the interface application) is easily supported. This is accomplished by specifying a Motif Label widget across a range of cells.

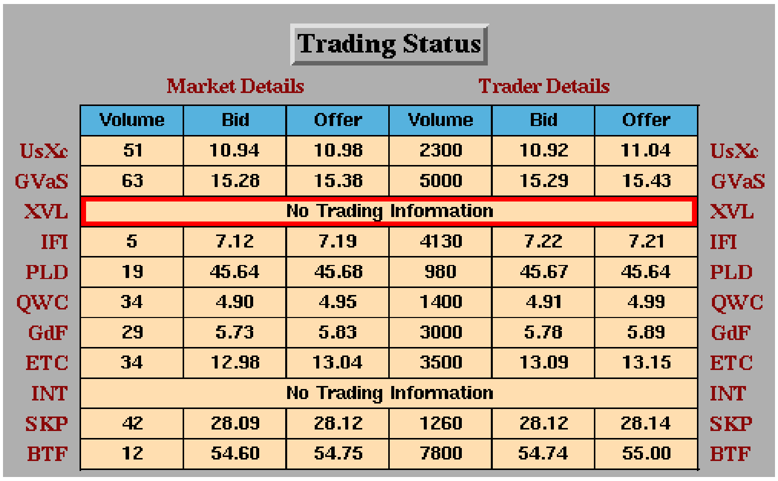
Constraint resource **XmNcellWidgetOverrideTranslations** allows the programmer to determine the focus for navigation events when the widget is mapped to the table. If set to True, arrows and tab keys take you back to the table process rather than remaining in the mapped widget. The XintEditTableGetCellWidget convenience function may be used to get the widget associated with a particular cell location.

Because the Widget in a Cell mechanism associates a range of cells with a single widget, a callback that is attached to the widget will be called as long as the current location is within that range. It is usually important to know the specific table location associated with any user event. For this purpose, the XintEditTableGetCellPointerPosition function should be used to return the table location of the cell pointer.

Example 4: Widget In A Cell on page 216 describes the coding that is required to create the checkbook example shown above. This example illustrates widget resources that are set automatically and widget resources that are set by the programmer.

Cell Spanning

The cell spanning feature of EditTable enables cells to span across several columns and/or rows (if rows/columns are not frozen). This span can be applied to all of the cells in the table, or a subset of the table locations. Cell spanning is enabled by the resource **XmNspanMode**, which also determines the manner in which the cells will be drawn. Figure 12 shows a display produced by `ETSpan.c` in the `examples/EditTable` directory:



	Market Details			Trader Details			
	Volume	Bid	Offer	Volume	Bid	Offer	
UsXc	51	10.94	10.98	2300	10.92	11.04	UsXc
GVaS	63	15.28	15.38	5000	15.29	15.43	GVaS
XVL	No Trading Information						XVL
IFI	5	7.12	7.19	4130	7.22	7.21	IFI
PLD	19	45.64	45.68	980	45.67	45.64	PLD
QWC	34	4.90	4.95	1400	4.91	4.99	QWC
GdF	29	5.73	5.83	3000	5.78	5.89	GdF
ETC	34	12.98	13.04	3500	13.09	13.15	ETC
INT	No Trading Information						INT
SKP	42	28.09	28.12	1260	28.12	28.14	SKP
BTF	12	54.60	54.75	7800	54.74	55.00	BTF

Figure 12. Cell and Annotation Spanning

The normal practice is to make sure that the adjacent cells, which are covered by the span, are empty. That is, cells for which no data is specified or where the value is one of the undefined values listed in the section *Specifying Undefined Values* on page 113. This is because any data which exists in a cell before spanning is specified can remain visible, even though that location becomes part of a spanned cell.

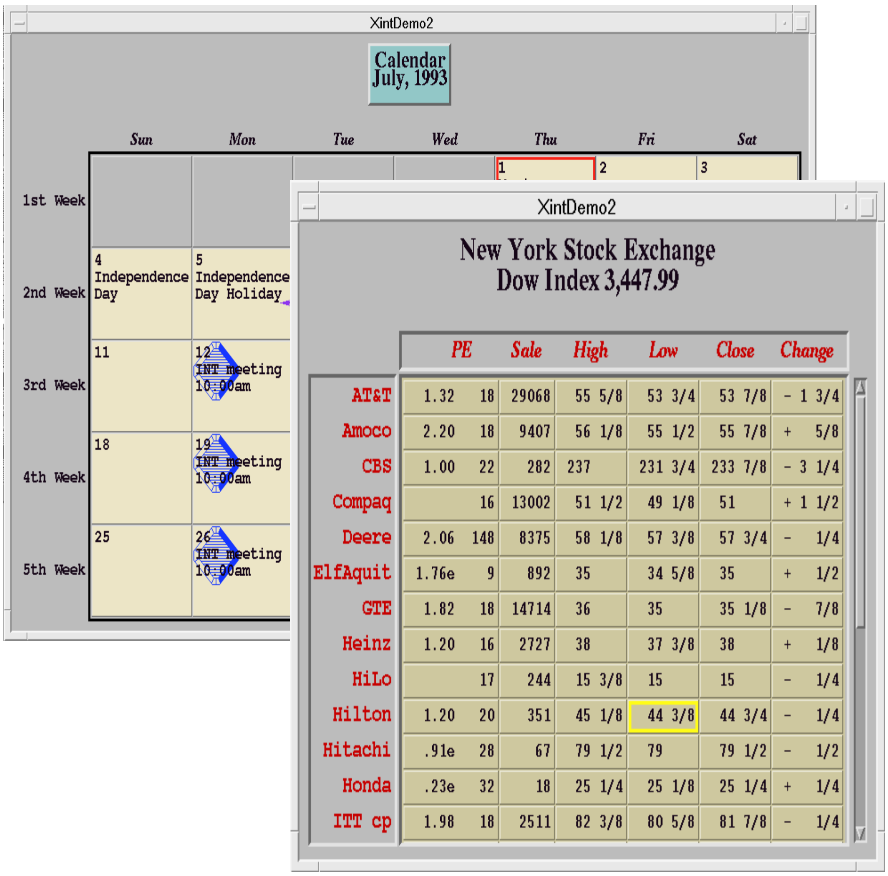
Also, if the **XmNspanCellPointer** resource is set to True, the cell pointer treats the spanned cell as a single location, and the cells that are covered by the span are inaccessible. However, if it is set to False, then the cells covered by a span can still be individually accessed and edited. Only empty row/column locations covered by a spanned cell will not interfere with the visibility of that cell.

The function XintEditTableCellSpanSetRange is used to set the cell span, as defined by the XintCellSpanFactor structure, over a given range of row/column locations. There is a corresponding function, XintEditTableCellSpanGetRange, that will retrieve the cell span factor for a particular cell.

Spanning can also occur in the horizontal and vertical annotation areas. This is done by specifying the starting row of the span range as zero (0) for the horizontal annotation and the starting column of the span range as zero (0) for the vertical annotation.

Typical Applications

A broad and flexible array of features makes EditTable particularly suitable for use in applications such as the ones discussed in the following sections Figure 13 shows an example of EditTable applications:



	Sun	Mon	Tue	Wed	Thu	Fri	Sat
1st Week					1	2	3
2nd Week	4 Independence Day	5 Independence Day Holiday					
3rd Week	11	12 INT meeting 10:00am					
4th Week	18	19 INT meeting 10:00am					
5th Week	25	26 INT meeting 10:00am					

		PE	Sale	High	Low	Close	Change
AT&T	1.32	18	29068	55 5/8	53 3/4	53 7/8	- 1 3/4
Amoco	2.20	18	9407	56 1/8	55 1/2	55 7/8	+ 5/8
CBS	1.00	22	282	237	231 3/4	233 7/8	- 3 1/4
Compaq		16	13002	51 1/2	49 1/8	51	+ 1 1/2
Deere	2.06	148	8375	58 1/8	57 3/8	57 3/4	- 1/4
ElfAquit	1.76e	9	892	35	34 5/8	35	+ 1/2
GTE	1.82	18	14714	36	35	35 1/8	- 7/8
Heinz	1.20	16	2727	38	37 3/8	38	+ 1/8
HiLo		17	244	15 3/8	15	15	- 1/4
Hilton	1.20	20	351	45 1/8	44 3/8	44 3/4	- 1/4
Hitachi	.91e	28	67	79 1/2	79	79 1/2	- 1/2
Honda	.23e	32	18	25 1/4	25 1/8	25 1/4	+ 1/4
ITT cp	1.98	18	2511	82 3/8	80 5/8	81 7/8	- 1/4

Figure 13. Example of EditTable Applications

Spreadsheets

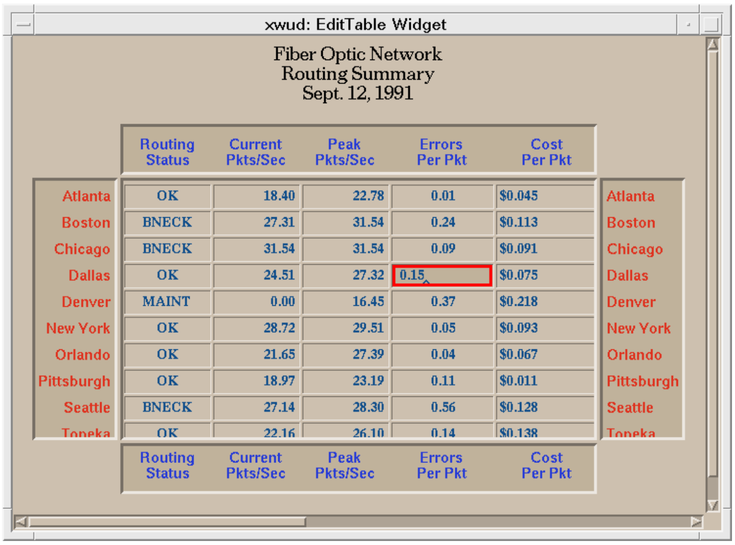
EditTable can be used to build spreadsheets that are more specialized and powerful that those provided by general-purpose spreadsheet programs. EditTable lets you use the full capabilities of C or C++, including any C-compatible data types, and have complete control of cell editing, display, and data validation. The widget's efficient data handling capabilities make it uniquely qualified to handle the display and analysis of large data sets such as those found in financial and scientific applications.

Real-time Monitoring and Analysis

You can use the EditTable widget for monitoring and analysis of real-time data, including:

- Financial trading systems
- Remote telemetry
- Telecommunication

Applications like these use EditTable's flexible color controls to indicate cells with out-of-range or alarm conditions. EditTable provides the features you need to build multiple widgets that share data — for example, a spreadsheet and plot that automatically update each other, as shown in Figure 14:



xwud: EditTable Widget

Fiber Optic Network
Routing Summary
Sept. 12, 1991

	Routing Status	Current Pkts/Sec	Peak Pkts/Sec	Errors Per Pkt	Cost Per Pkt	
Atlanta	OK	18.40	22.78	0.01	$0.045	Atlanta
Boston	BNECK	27.31	31.54	0.24	$0.113	Boston
Chicago	BNECK	31.54	31.54	0.09	$0.091	Chicago
Dallas	OK	24.51	27.32	0.15	$0.075	Dallas
Denver	MAINT	0.00	16.45	0.37	$0.218	Denver
New York	OK	28.72	29.51	0.05	$0.093	New York
Orlando	OK	21.65	27.39	0.04	$0.067	Orlando
Pittsburgh	OK	18.97	23.19	0.11	$0.011	Pittsburgh
Seattle	BNECK	27.14	28.30	0.56	$0.128	Seattle
Topeka	OK	22.16	26.10	0.14	$0.138	Topeka
	Routing Status	Current Pkts/Sec	Peak Pkts/Sec	Errors Per Pkt	Cost Per Pkt	

Figure 14. Example of Real-time Data Monitoring

Data Entry and Validation

EditTable is especially suited for applications that require intensive data entry. The application can specify which cells are editable or protected. Users can edit cells directly using backspace/retype or cut/copy/paste features. All data can be checked for accuracy or violation of certain minimum/maximum ranges immediately as the user finishes an entry or later when the entire table is saved. When used with the INT scroll widget (XintScroll) multiple tables can be scrolled simultaneously for easier viewing and editing.

Database Display and Browsing

EditTable can be used as browser for relational or flat-file databases. This capability is especially useful for viewing the actual data structure, since EditTable lets you view individual data fields in their native formats (hex, string, float, etc.) or in various converted formats controlled by the application. Selective freezing and scrolling of different rows or columns makes it easy to compare data in any part of the database to certain reference fields.

Integration with Other INT Widgets

EditTable is a subclass of other widgets, and automatically inherits the features of any other widgets that belong to the same general class. Figure 15 shows the hierarchy of widgets in the same class as EditTable:

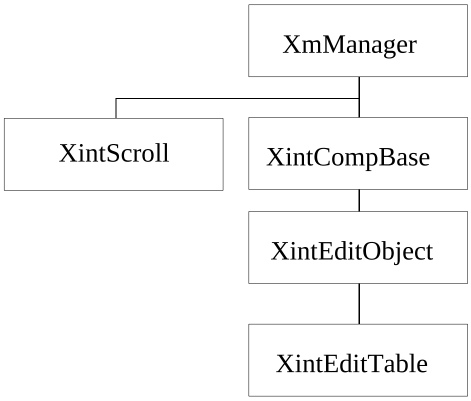


Figure 15. Overall Widget Hierarchy

EditTable offers integration with the other widgets in this metaclass, as summarized in the following sections.

Hardcopy Output and Coordinate Mapping (XintCompBase)

The CompBase widget gives all other widgets in the metaclass the ability to produce hardcopy output and perform coordinate system mapping between the widget's coordinate system and the application's coordinate system.

Enhanced Scrolling (XintScroll)

The Scroll widget is a container widget that scrolls an EditTable widget in such a way that its annotation remains visible during scrolling. A Scroll widget combines:

- One or two ScrollBar widgets.
- A viewport onto a portion of the scrolled child.
- Drawing area widgets for displaying the scrolled child widget's title, horizontal annotation and vertical annotation.

The application has control over which components of the Scroll widget are displayed and where they are located. Also, several Scroll widgets can share one or both scrollbars to allow for synchronous scrolling of multiple tables.

Graphic Objects (XintGraphic)

The XintGraphic object class is the base class that defines the basic resources and methods for displaying and editing graphic objects. Graphic objects can only be displayed in a widget that is a subclass of the XintEditObject widget class. At the lowest level, the graphic object subclasses include:

- *XintLine* for drawing straight lines, with or without arrow heads.
- *XintRectangle* for drawing rectangular objects, which can contain other objects such as text objects and circles.
- *XintTextObj* for inserting a character string that can span multiple lines within a rectangular area.
- *XintOval* object for drawing circles or ellipses, with coordinates specified in relation to XintRectangle.
- *XintPolyline* object for drawing polylines or polygons.

The XintGraphic class defines all the basic methods that apply to objects, including methods to display, select, move and resize an object. Additional methods include group, ungroup, cut, paste, and file import/export.

Graphic resources define object properties such as fill pattern, color, line thickness and line style. Figure 16 shows the hierarchy for graphic objects:

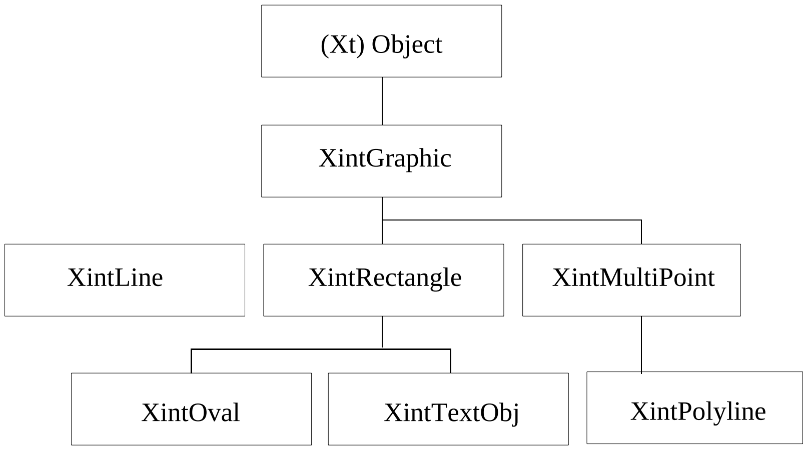


Figure 16. Hierarchy for Graphic Objects

Graphic Object Editing, Storage, and Retrieval (XintEditObject)

An optional INT product called the Object Editor Library (XintEditObject) lets users draw graphic objects such as text, lines, circles, arcs, or arrows and superimpose them directly on a table.

For example, a user may want to draw attention to a special group of data in a report or presentation, as shown in Figure 17:

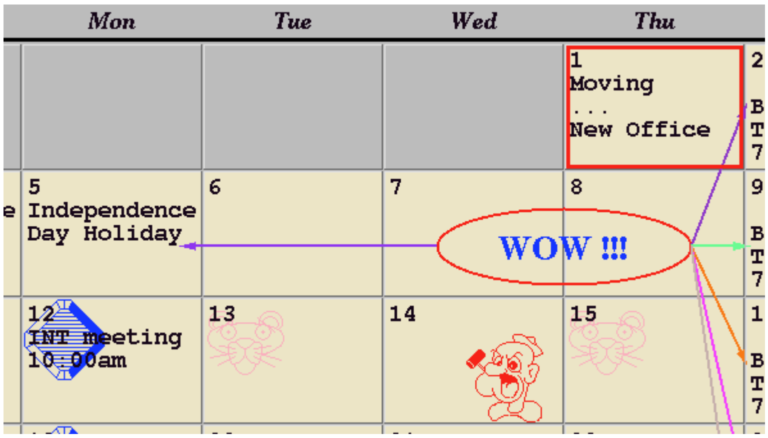


Figure 17. Example of Graphics Superimposed on Table

The EditObject widget provides support for displaying, editing, and storing/retrieving graphic objects 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 lets you save and restore objects to or from an ASCII file. An internal clipboard mechanism provides cut and paste functionality inside an application or between two different applications that use EditObject widgets (or widgets from a class that has the XintEditObject as an ancestor).

Graphics can be saved to a separate file for later retrieval and use in the table. When a component is removed from a table, any graphic object tied to that component is removed also. However, the graphic object still exists and will reappear when the removed element is restored.

ChartObject Library

ChartObject is a library of Graphic objects based on the architecture described above. This package allows the application and the end-user to produce all kinds of 2D and 3D charts. A close integration exists between the EditTable and the Chart object library, and an EditTable widget and a ChartObject can be dynamically linked. It is also possible to drag and drop cells from an EditTable widget into a ChartObject. Refer to the *ChartObject Programming Guide* for more information on ChartObject and Graphic libraries.

Instance Network

The widgets and objects in the EditTable Widget Library can only be created as children of certain specific classes of widgets. Conversely, a widget accepts only certain types of widgets and/or objects as children. The following sections describe parent/child restrictions for widgets and objects.

CompBase Class

You cannot create an instance of this widget class because it serves only as a base (metaclass) for the composite widgets.

EditObject Widget

You cannot create an instance of this widget class because it serves only as a base (metaclass) for the composite widgets.

EditTable Widget

An EditTable widget can be a child of any widget that accepts an XmManager widget as a child (such as XmForm or TopLevelShell). Often, an EditTable widget will be the child of an instance of the XintScroll widget class. Since EditTable is a subclass of XintEditObject, an EditTable widget can have children that are Graphic objects.

Scroll Widget

An XintScroll instance can be a child of any widget that accepts XmManager as a child (such as XmForm or TopLevelShell). XintScroll can control the scrolling of only one EditTable widget.

Graphic Objects

Objects are always managed automatically by their widget parent. Because of this special relationship between objects and their parents, there are stringent restrictions on who can be a parent of a specific type of object. Note also that Graphic objects cannot have children. Graphic objects must be children of an EditTable widget.

Creating and Freeing INT Widgets

Creating INT widgets

The INT widgets are created in exactly the same manner as Xm widgets are created. There are widget creation convenience functions that you can use to create (unmanaged) INT widgets. These widget creation functions are described in the INT widget class reference sections. You can also use the Xt widget creation functions:

- XtCreateManagedWidget
- XtCreateWidget
- XtVaCreateManagedWidget
- XtVaCreateWidget

As always, after you create an unmanaged widget, you must manage the widget before it is visible on the screen.

Freeing INT widgets

INT widgets are freed in exactly the same way as Xm widgets are freed. You can use the Xt function, XtDestroyWidget, to free the memory and data structures associated with a INT widget or object. When a parent widget is destroyed with this function, all of its children will also be destroyed.

Freeing data structures

Some INT widgets (for example, EditTable) use external data structures which you may need to free using the Xt function, XtFree. Also, some INT functions (for example, XintEditTableGetColumnData) return pointers to a copy of the data requested. You may also need to free the storage allocated for the copy when you have finished using it. Widget and object reference sections indicate when it’s your responsibility to free INT widget or object related data structures.

Obtaining and Setting Resource Values

Obtaining resource values

An application gets the current value of a resource of an INT widget in exactly the same manner as it gets resources of Xm widgets. You can use the Xt resource value access functions:

- XtGetValues
- XtVaGetValues.

Setting resource values

An application sets the value of a resource of an INT widget in exactly the same manner as it sets the resources of Xm widgets at widget creation time. After widget creation time you can use the following Xt resource value access functions to set resources:

- XtSetValues
- XtVaSetValues

CompBase Widget Metaclass

2

Overview

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.

This chapter includes the following sections:

- **Inherited Behavior and Resources** on page 32
- **CompBase Functions** on page 33

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>`

Resources

The following resources are defined by the CompBase class:

Name	**Default** **Type**	**Access**
XmNfontPath	NULL char *	CSG
XmNwarning	XintWARNING_POST int	CSG

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 defined integer constants listed in the following table 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, Grid or Image). 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.

Note: The INT EditTable widget class defines an additional function, XintEditTableOutputPostscript, for hardcopy output.

Function	**Description**
XintCGMDrawBox	Tells CGM output whether or not to draw a box around a CGM plot.
XintCGMGetDimensions	Gets size in inches used by a widget.
XintCGMPixelToInch	Converts size specified in pixels to a size specified in inches.
XintCGMSetVDCType	Selects either real or integer output coordinates for CGM.
XintGetWidgetSize	Returns size in pixels that a widget will occupy when output to hardcopy.
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.
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.
XintPostscriptGetDefaults	Gets the PostScript page characteristics.
XintPostscriptSetBackground	Sets the background for PostScript output.
XintPostscriptSetDefaults	Sets the PostScript page characteristics.
XintVerticalPixelToUser	Pixel to User coordinates conversion in vertical direction.
XintVerticalUserToPixel	User to Pixel coordinates conversion in horizontal direction.

XintCGMDrawBox

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

```
  void XintCGMDrawBox (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.

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 Scroll or 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 CompBase 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 CompBase 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 that are 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, 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 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 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, the function displays a warning message to the end user (as controlled by resource **XmNwarning**) and returns False; otherwise, it 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 the PostScript output page characteristics used by the 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

This function 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	The 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*.

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

XintVerticalUserToPixel

This function 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	The 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.*

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

EditObject Widget Class 3

Overview

The EditObject widget class provides support for displaying, editing, and storing/retrieving graphic objects created using 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. The resources, functions and callbacks listed in this section are only useful when EditTable is used in conjunction with the INT Chart and/or INT Graphic object library. If you are not using those two products you can skip this section.

The display and editing capabilities of the 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).

This chapter includes the following sections:

Creating an EditObject Widget

The main purpose of the EditObject widget class is to serve as a superclass for other INT widget classes such as Grid or VirtualGrid. However it is possible to instantiate an EditObject widget directly if you want an empty window that supports the drawing and editing of Graphic objects.

Coordinate System

The EditObject class does not allow the application to define its own coordinate system. An EditObject widget always uses a linear coordinate system that is between 0.0 and 100.0 both horizontally and vertically. Other classes based upon the EditObject widget class (such as Grid or PlotXY) allow the application to define a coordinate system. You can use the functions provided in the CompBase widget class to do transformations between pixel values and the EditObject coordinate system.

Creating And Deleting Objects

Graphic objects can be created using the standard Motif widget creation process. Alternatively, a convenience function, EditObjectInsert, provides for the interactive creation of an Graphic object by the end user or application. Like other widgets, graphic objects are automatically destroyed when their parent is destroyed. Objects can be also be destroyed using Motif function XtDestroyWidget or function XintEditObjectDestroyObject which is faster.

Object Selection

Once an object is created, it can be selected using BSelect. Handles are displayed around the edges of a selected object (or group of objects) to indicate it has been selected. 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 appears as an empty rectangular window. In Figure 18, a Chart object and several Graphic objects have been created in an EditObject widget:

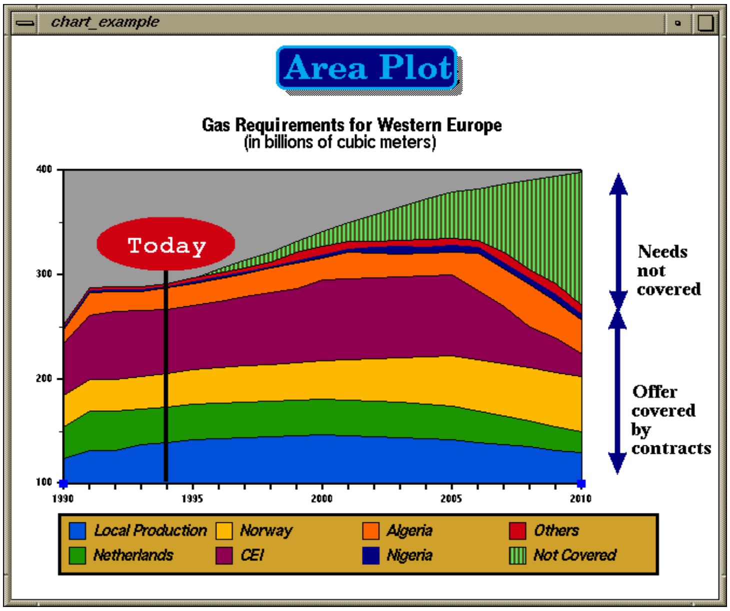


Figure 18. EditObject Containing a Chart and Various Graphic Objects

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>

Resources

The following resources are defined by the EditObject widget class.

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

Name(continued)	Default Type	Access
XmNpointSelectionTolerance	4 int	CSG
XmNresourceDialogCallback	XtCallbackList NULL	C
XmNrubberbandCallback	NULL XtCallbackList	C
XmNselectionCallback	NULL XtCallbackList	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 49 for 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 49 for 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 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 that are called when a graphic object is being 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 and 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 XintEditObjectDeselectObject 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 ObjectEdit 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 continuously 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 using (for example) the function XintGraphicRubberbandPolyline.

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.

Name	Description
ChangeCursorMask()	Changes the color of the cross hair cursor in increment.
DrawCursor()	Draws the cross hair cursor at the location of the mouse pointer.
EndDrawCursor()	Terminates the cross hair drawing operation.
InitDrawCursor()	Initiates the cross hair drawing operation.
TraverseCurrent()	Moves the focus to the current widget.
PreviousTabGroup()	Move the focus to the previous tab group.
NextTabGroup()	Moves the 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 the cursor is moved inside the widget window, this action calls the list of procedures specified by **XmNlocator-Callback**.
MotifDragStart()	Initiates a Motif drag operation and calls callback XmN-dragDrop Callback. Action is disabled if resource **XmNal-lowDrag 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()	Whenever a Button is pressed inside the widget window, this action calls the list of procedures specified by resource **XmNselectionCallback**.
InitAreaSelection(callback)	Initiates the selection of a rectangular area in the widget window. Callback XmNareaSelectionCallback will be called by action EndAreaSelection when the selection is terminated.

Name (continued)	**Description**
InitAreaSelection(single)	Initiates the selection of a rectangular area within which all graphic objects included will be selected. Previously selected objects are deselected first.
InitAreaSelection(extend)	Initiates the selection of a rectangular area within which all graphic objects included will be selected. The new selection will extend the current selection.
ExtendAreaSelection()	Draws a rectangle bounding the area being selected.
EndAreaSelection()	Terminates the area selection operation. Calls XmNarea-SelectionCallback 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**. This action is also used internally by the widget for creating an object interactively.
ObjectEditStart(move)	Initiates a Move operation on the 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 Size operation on 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 a rubberband operation where callback XmNrubberbandCallback is called continuously as the pointer moves. It is up to the application to actually draw a rubberband shape.
ObjectEdit()	Continues editing operation initiated by action ObjectEditStart.

Name (continued)	**Description**
ObjectEditEnd()	Terminates the editing operation initiated by action ObjectEditStart and calls callback XmNverifyCallback (callback is defined in Graphic class). When **XmNobject-EditMode** is set to XintEDIT_INSERT and the object being edited is not a MultiPoint object, terminates the insertion operation and calls the XmNinsertObjectCall-back.
ObjectEditEnd(m)	Terminates the editing operation initiated by action ObjectE-ditStart 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 XmNinsertOb-jectCallback.
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.
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)
None <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)

EditObject Callbacks

The following callbacks are defined by a EditObject widget.

Name	Structure	Reason
XmNareaSelectionCallback	XintEditObjectArea-SelectionCallbackStruct	XintCR_AREA_SELECTION
XmNselectionCallback	XintEditObjectCallback-Struct	XintCR_SELECTION
XmNcopyCallback	XintEditObjectEdit-CallbackStruct	XintCR_COPY
XmNcutCallback	XintEditObjectEdit-CallbackStruct	XintCR_CUT
XmNdragDropCallback	XintEditObjectDragDrop-CallbackStruct	XintCR_DRAG XintCR_DROP
XmNinsertObjectCallback	XintEditObjectInsert-CallbackStruct	XintCR_INSERT_OBJECT
XmNlocatorCallback	XintEditObjectLocatorCall-backStruct	XintCR_LOCATOR
XmNobjectDeselection-Callback	XintEditObjectSelection-CallbackStruct	XintCR_OBJECT_DESELECTION
XmNobjectSelection-Callback	XintEditObjectSelection-CallbackStruct	XintCR_OBJECT_SELECTION
XmNpasteCallback	XintEditObjectEdit-CallbackStruct	XintCR_PASTE
XmNrubberbandCallback	XintEditObjectRubberband-CallbackStruct	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 resource **XmNselectionCallback**.

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 (for 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 Name	Description
XintCreateEditObject	Creates an EditObject widget.
XintDrawCursorFromData	Causes a cross hair cursor to be drawn at a specified location in the EditObject's window
XintEditObjectBack	Moves the current object behind all other objects in the widget.
XintEditObjectCopy	Copies the selected objects into the clipboard.
XintEditObjectCurrent	Returns the last object selected.
XintEditObjectCut	Copies all selected objects into the 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.
XintEditObjectFront	Moves the current object in front of all other objects in the 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 the current object one place down in the stacking order.
XintEditObjectMove	Allows interactive movement of the selected object.

Function Name (continued)	**Description**
XintEditObjectNew	Destroys all objects belonging the EditObject widget.
XintEditObjectOpen	Manages a dialog that allows the loading of an ASCII object description file.
XintEditObjectPaste	Pastes the objects in the clipboard into the EditObject widget.
XintEditObjectRaise	Moves the current object one place up in the stacking order.
XintEditObjectReadFile	Reads an ASCII file containing a description of objects and places them into the EditObject widget.
XintEditObjectSave	Saves all the objects of and EditObject widget into an 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 the Graphic objects of an EditObject widget.
XintEditObjectSelectList	Returns the list of the selected objects.
XintEditObjectSelectObject	Adds an object to the list of selected objects.
XintEditObjectSetEditMode	Sets the value of resource XmNobjectEditMode.
XintEditObjectSize	Allows the interactive sizing of the currently selected object.
XintEditObjectUngroup	Ungroups the currently selected group object.
XintEditObjectWriteFile	Saves all the objects of an EditObject widget into an ASCII file.

XintCreateEditObject

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. This function is 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 the intersection rectangle, in pixels.
int	y	Y value of upper left corner of the intersection rectangle, in pixels.
int	width	Width of the intersection rectangle, in pixels.
int	height	Height of the intersection rectangle, in pixels.
int *	count	Returned count of objects in the 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 permeate 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, Object object)
```

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

4 EditTable Widget

Overview

An EditTable widget displays a table of values (integer, string or floating point) organized in rows and columns. A table can be manipulated by the application via convenience functions or by the end user via actions and translations.

This chapter includes the following sections:

Data Organization

The data used by the EditTable widget is organized as columns of cells. Every column in the table contains the same number of cells. Every cell in a column has the same data type and data format.

Table Size

The table's size at widget creation time is specified in terms its initial number of rows and columns, but it can be changed afterwards by adding or deleting rows or columns. The EditTable widget is designed to handle very large tables efficiently, but the practical size of a table may be limited by the available memory.

Table Orientation

Columns in a table can be oriented vertically or horizontally. A vertical orientation is the default orientation, but in some applications a horizontal orientation may be useful. When the columns are oriented horizontally, the cells in a column are drawn from left to right and the column annotation appears on the left or the right of the column. All column and row operations are used in the same way, no matter the column orientation.

Data Structures

The values in the cells of the table can exist only within the EditTable widget or they can be shared by the application and the EditTable widget. There are some restrictions on the operations that can be performed on the table data when it is shared by the application and the widget, but the memory required to hold the table data is minimized in this case.

Supported Data Types

The values in the cells of a column can be short integers, integers, long integers, floating point numbers, double precision floating point numbers, character strings or pointers.

Using EditTable with Scroll

When the EditTable widget is a child of an INT Scroll widget, the table can be scrolled so that the table title and row/column annotation remains visible. Also, some columns and/or rows can be frozen so that they remain visible as the rest of the table is scrolled.

Creating an EditTable Widget

An application creates an EditTable widget as a child of a container widget such as a Bulletin Board widget, Form widget, Scrolled Window widget, or INT Scroll widget. The number of rows and columns, along with the format of each column, are usually specified when the EditTable widget is created. If the application has data that is to be displayed in the table, then the application specifies that data to the EditTable widget after the table is created (via convenience functions). During the creation process, the application uses resources to specify callback procedures (for the table editing operations) and general table attributes such as automatic row or column annotation, grid line appearance, and fonts.

Displaying a Table

When the EditTable widget is mapped, it displays the data (if any) contained within the columns of the table. If the data for a column has not been defined, then the EditTable widget displays empty cells in that column.

Formatting Data

Formatting the data for display is based on a C format specifier that is defined for each column. A callback, XmNformatCellCallback, can be invoked to handle non standard formats or pointer data.

Updating a Table

When changes in the data for one or more columns in the table occur, the changes can originate in the application or with the end user. If the application changes the data in the shared data structures, then the application must call the function XintEditTableUpdateDataDisplay for each column that has changed, so that the EditTable widget can update the display of the table. The application can also change the value in a specific cell and cause the table display to be updated by using the function XintEditTableFillCell. If the end user changes some cell values, the EditTable widget will update the display and the shared data structures automatically.

Editing Operations

The EditTable widget supports a large number of editing operations. The edit operations can be performed by the application using provided functions or by the end user via action routines. The types of edits that can be performed are categorized as table edits, annotation edits, column edits, row edits or cell edits.

Table Edit

The size of a table and other characteristics such as the table title, highlight colors, and default column attributes can be changed via EditTable resources after the table is created. Rows and columns may be cut, pasted, inserted or deleted via action routines or convenience functions. Also, a range of columns or rows can be reordered by the application using a supplied function.

Annotation Edits

Row and column annotation can be changed following table creation by using provided functions. The application changes row and column annotation by updating the annotation for a single row or column. Alternatively, the application can change the annotation for all rows and all columns at one time. In either case, the application uses provided functions. The end user can change a single row or column annotation with an action routine. This action routine calls an application defined callback routine that actually performs the annotation editing operation.

Column Edits

A column can be modified by changing the format of the data in the column, the width of the column, or the number of rows in the column.

Row Edits

Because a table is column oriented, the rows must be edited indirectly by editing the columns in the table or by editing the cells in the row.

Cell Edits

The value of a cell can be changed if the cell is in an editable column and if the application allows the edit operation to occur. The format or data type of a cell can only be changed by changing the format or data type of the column that the cell is in. To edit the value in a cell, the end user must select the cell with the mouse or traverse to it using the keyboard. If the end user enters a value after selecting the cell, then the current value will be replaced. If the end user clicks a second time, then any new characters typed are inserted into the current value. If the end user drags the mouse cursor through a portion of the cell's current value, the portion of the string selected with the mouse will be replaced with the characters typed.

Input Validation

You can specify a validate value callback so that a value entered into a cell can be validated before the change is made permanent. The application can elect to allow the change, substitute its own value or disallow the change.

Traversing the Table

The user can traverse the table by using the keyboard (e.g. tab, return, cursor keys) or he can use the pointer to move from one location in the table to another. The application can control the traversal via a callback so that the cursor skips over areas of the table that are not relevant to the current application operation. The **XmNoverrideTextTranslations** resource allows the user to control the action of the Return, Left, Right, Up and Down keys to provide navigation between cells or within a cell.

Cutting and Pasting

When row(s) or column(s) are deleted or copied, they are placed on the EditTable's clipboard. The clipboard can simultaneously hold columns and rows. When another copy or delete operation occurs, then the data copied or deleted replaces the same type of data (row or column) currently on the clipboard. Data on the clipboard can be pasted into the table using convenience functions and action routines.

Interactive Move, Copy and Resize

A set of actions is available for performing interactive move, resize or copy operations on rows, columns or cells. The actions that operate on cells are EditTableStartDrag, EditTableExtendDrag and EditTableEndDrag. The drag action EditTableStartDrag supports an argument which specifies the type of operation, move, resize, copy or a combination of those. The actions for editing rows or columns are AnnotationStartDrag, AnnotationExtendDrag and AnnotationEndDrag. Row and column actions need to be registered in the EditTable translation table if the table is not created as a child of a Scroll widget. Otherwise, the application should use resources **XmNrowAnnotationTranslations** and **XmNcolumnAnnotationTranslations**.

Frozen Columns

When the EditTable widget is a child of a Scroll widget, columns can be frozen in the table. Frozen columns are displayed on the left or right side of the table and are not scrolled when the table is scrolled horizontally. However, cells in a frozen column can be selected and edited. The application uses a function to freeze as many columns as desired. Another function is used to release a frozen column, returning it to its former position and state.

Frozen Rows

When the EditTable widget is a child of a Scroll widget, rows can be frozen in the table. Frozen rows are displayed above or below the table and are not scrolled when the table is scrolled vertically. However, cells from a frozen row can be selected or edited. The application uses a function to freeze as many rows as desired. Another function is used to release a frozen row, returning it to its former position and state.

Cell, Column and Row Indices

The rows and columns in a table are indexed using sequential integers. The leftmost column has an index of 1 and the topmost row has an index of 1. The cells in a table are indexed with a pair of integers. The first integer is the index of the column containing the cell and the second integer is the index of the row containing the cell. When addressing a block of cells in a table, you usually specify the indices of the rows and columns that bound the block of cells.

Column Annotation

The columns in a table can be annotated with character strings. The annotation strings can be generated automatically as strings of letters in alphabetical order (A, B, C,..., AA, AB, etc.) by the EditTable widget, or the application can supply a string either at column creation time or after a column is created.

Row Annotation

The rows in a table can be annotated with any of the supported data types. The supported data types are short, integer, long integer, floating point, double precision floating point, and character string. However, you cannot mix data types for the row annotations. The annotation strings can be generated automatically as sequential integers by the EditTable widget, or the application can supply a value either at row creation time or after a row is created.

Margin Size Specification

An EditTable widget will automatically determine the optimal size for one or more of the margins around the scrolled child window according to the setting of the **XmNautoMarginAdjust** resource. You can choose which of the margins will be automatically sized and which will be sized according to values of the margin resources (**XmNtopMargin**, **XmNleftMargin**, **XmNrightMargin**, **XmNbottomMargin**). However, if an EditTable widget is a child of a Scroll widget the sizes of the (non-auto sized) margins are set using the resources of the Scroll widget (**XmNtopAnnotationHeight**, **XmNleftAnnotationWidth**, **XmNrightAnnotationHeight**, **XmNbottomAnnotationHeight**).

Sub-tables

The table displayed by an EditTable is organized as one to four sub-tables, with separate sub-tables for each of the following:

- All cells in rows and columns that are not frozen
- Cells in rows that are frozen (if any) but not including cells in frozen columns
- Cells in columns that are frozen (if any) but not including cells in frozen rows
- Cells contained in the intersection of the frozen columns and frozen rows (if any)

You can get the widget IDs of any of these subtables using the function XintEditTableGetSubtable.

Column and Row Visibility

The display of columns and rows of an EditTable widget can be selectively turned on or off using functions XintEditTableChangeColumnVisibility or XintEditTableChangeRowVisiblity. Resource **XmNallowPartialCellDisplay** prevents displaying the contents of columns that are only partially visible.

Drawing Graphics on a Table

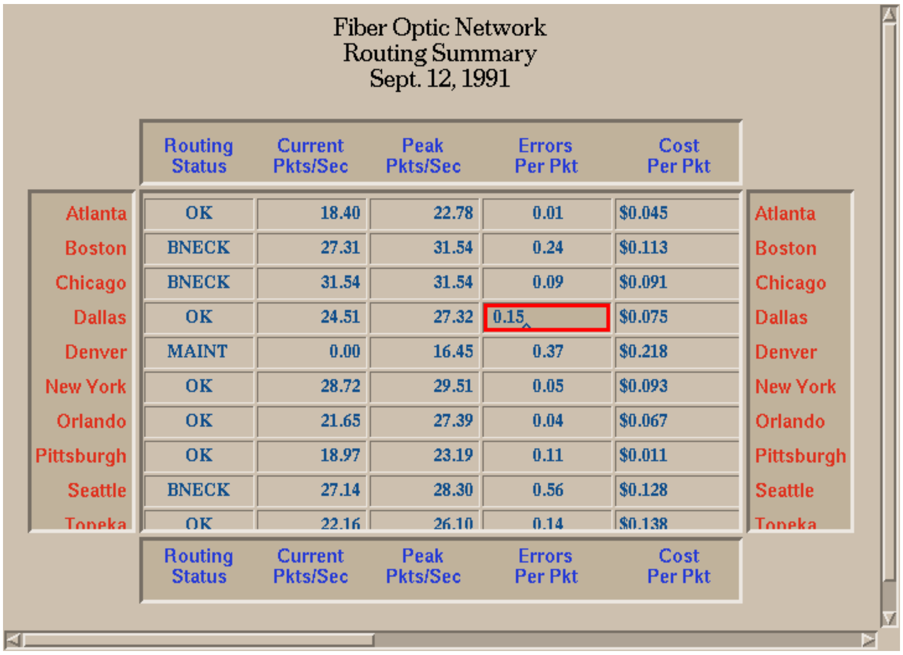
Graphic objects such as text, arrows and circles can be created and edited on a table. Those graphics can be saved to a file for retrieval for later use. Because graphic objects can have only one parent, a graphic object cannot span more than one of the subtables of an EditTable. For instance, a line cannot begin in the frozen row subtable and end in the frozen column subtable. Also note that if a graphic object spans multiple elements in a subtable (e.g. columns in the frozen column subtable) and that element is removed from the subtable (e.g. releasing a frozen column that one of the line end points was in) then the graphic object will be removed from the display. However, the graphic object does still exist and will reappear when the removed element is again placed in the subtable.

Graphic Object Coordinate System

The coordinate system used to draw INT Graphic objects inside an EditTable widget subtable uses real numbers for both the X and Y axes. The X coordinate of a point used to specify a Graphic object would be specified as the number of the column containing the point followed by the decimal fraction of the column where the point lies across the horizontal dimension of the column. For instance, to specify the X coordinate of a point located horizontally in the middle of the 3rd column, you would specify 3.5 as the X coordinate. Similarly, the Y coordinate of a point used to specify a Graphic object would be specified as the number of the row containing the point followed by the decimal fraction of the row where the point lies across the vertical dimension of the column. For instance, to specify the Y coordinate of a point located vertically one tenth of the cell height away from the top of a cell in the 14th row, you would specify 14.1 as the Y coordinate.

EditTable Widget Appearance

Figure 19 shows an example of an EditTable widget as a child of an INT Scroll widget:



Fiber Optic Network
Routing Summary
Sept. 12, 1991

	Routing Status	Current Pkts/Sec	Peak Pkts/Sec	Errors Per Pkt	Cost Per Pkt	
Atlanta	OK	18.40	22.78	0.01	$0.045	Atlanta
Boston	BNECK	27.31	31.54	0.24	$0.113	Boston
Chicago	BNECK	31.54	31.54	0.09	$0.091	Chicago
Dallas	OK	24.51	27.32	0.15	$0.075	Dallas
Denver	MAINT	0.00	16.45	0.37	$0.218	Denver
New York	OK	28.72	29.51	0.05	$0.093	New York
Orlando	OK	21.65	27.39	0.04	$0.067	Orlando
Pittsburgh	OK	18.97	23.19	0.11	$0.011	Pittsburgh
Seattle	BNECK	27.14	28.30	0.56	$0.128	Seattle
Topeka	OK	22.16	26.10	0.14	$0.138	Topeka
	Routing Status	Current Pkts/Sec	Peak Pkts/Sec	Errors Per Pkt	Cost Per Pkt	

Figure 19. Example of EditTable as a Child of INT Scroll

EditTable Resources

This section describes the EditTable resources, as follows:

- **Inherited Behavior and Resources** on page 82
- **Defined Resources** on page 82
- **Constraint Resources** on page 111

Inherited Behavior and Resources

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

- Class pointer is *xintEditTableWidgetClass*
- Class name is *XintEditTable*
- Header file is included as <Xint/EditTable.h>

Defined Resources

Defined resources

Resources defined by the widget include the following:

- Number of rows and columns
- Grid line style and color
- Font used to display data
- Title placement

Some columns can be specified as read-only while others can be edited by the application or end user. Color or monochrome PostScript output of the entire table or a subset of the table is also provided. An EditTable widget can be used in conjunction with an INT Scroll widget so that the table annotation remains visible when the table is scrolled vertically or horizontally.

Note: Because the INT EditTable widget class is a subclass of the EditObject widget class, it inherits the capability to draw Graphic objects on the table.

The following resources are defined by the XintEditTable widget class:

Name	Default Type	Access
XmNadjustTextMaxLength	False Boolean	CSG
XmNallowPartialCellDisplay	True Boolean	CSG
XmNasciiFilename	NULL char *	CSG
XmNautoColumnRowMove	True Boolean	CSG

Name (continued)	Default Type	Access
XmNautoMarginAdjust	XintADJUST_NONE int	CSG
XmNautoScrollingInterval	100 int	CSG
XmNautoTextOverflowMarker-Size	True Boolean	CSG
XmNautomaticColumnAnnota-tion	True Boolean	CSG
XmNautomaticRowAnnotation	True Boolean	CSG
XmNbottomMargin	40 int	CSG
XmNcellAttributesCallback	NULL XtCallbackList	C
XmNcellHeightData	NULL int *	CSG
XmNcellHighlightColor	dynamic Pixel	CSG
XmNcellPointerBorderThickness	3 int	CSG
XmNcellPointerColor	red Pixel	CSG
XmNcellPointerRetained	True Boolean	CSG
XmNcellSizeUnit	XintUNIT_CHARACTER int	CSG
XmNcellWidgetDisplayCallback	NULL XtCallbackList	C
XmNcellWidthData	NULL int *	CSG
XmNcheckEditModeCallback	NULL XtCallbackList	C
XmNclipAnnotation	False Boolean	CSG
XmNcolumnAlignmentData	NULL int *	CSG

Name **(continued)**	**Default** **Type**	**Access**
XmNcolumnAnnotationData	NULL String *	CSG
XmNcolumnAnnotationFont	“fixed” char *	CSG
XmNcolumnAnnotationForeground	dynamic Pixel	CSG
XmNcolumnAnnotatioTranslations	NULL XtTranslations	CSG
XmNcolumnCallback	NULL XtCallbackList	C
XmNcolumnDataFormatData	NULL String *	CSG
XmNcolumnDataTypeData	NULL int *	CSG
XmNcolumnEditModeData	NULL int *	CSG
XmNcolumnFontIndexData	NULL int *	CSG
XmNcolumnOrientation	XintCOLUMN_VERTICAL int	CSG
XmNdefaultCellHeight	1 int	CSG
XmNdefaultCellWidth	6 int	CSG
XmNdefaultColumnAlignment	XintALIGNMENT_BEGINNING_MIDDL E int	CSG
XmNdefaultColumnDataFormat	“%s” char *	CSG
XmNdefaultColumnDataType	XintTYPE_STRING int	CSG
XmNdefaultColumnEditMode	XintCOLUMN_EDITABLE int	CSG
XmNdoubleClickCallback	NULL XtCallbackList	C

Name (continued)	**Default** **Type**	**Access**
XmNdoubleClickInterval	dynamic int	CSG
XmNdragCallback	NULL XtCallbackList	C
XmNdragCursorType	dynamic int	CSG
XmNdragForeground	black Pixel	CSG
XmNdragGridLineStyle	XintGRID_LINE_SOLID int	CSG
XmNdragShowCellContent	True Boolean	CSG
XmNeditAnnotationCallback	NULL XtCallbackList	C
XmNfirstVisibleColumn	1 int	CSG
XmNfirstVisibleRow	1 int	CSG
XmNfontTable	NULL String *	CSG
XmNformatCellCallback	NULL XmCXtCallbackList	C
XmNformatColumnAnnotation-Callback	NULL XtCallbackList	C
XmNformatRowAnnotationCall-back	NULL XtCallbackList	C
XmNfreezeUpdate	False Boolean	CSG
XmNfrozenColumnPlacement	XintPLACEMENT_LEFT int	CSG
XmNfrozenRowPlacement	XintPLACEMENT_TOP int	CSG
XmNgridLineForeground	dynamic Pixel	CSG
XmNgridLineHighlightColor	dynamic Pixel	CSG

Name (continued)	**Default Type**	**Access**
XmNgridLineHighlightThickness	3 int	CSG
XmNgridLineOrientation	XintGRID_LINE_CROSS-WISE int	CSG
XmNgridLineStyle	XintGRID_LINE_SOLID int	CSG
XmNgridLineWidth	1 int	CSG
XmNhorizontalAnnotationPlace-ment	XintPLACEMENT_TOP_BOTTOM int	CSG
XmNhorizontalCellMargin	3 int	CSG
XmNhorizontalLabel	NULL char *	CSG
XmNhorizontalLabelAlignment	XintALIGNMENT_CENTER int	CSG
XmNhorizontalLabelFont	“*courier*bold-r*120*” char *	CSG
XmNhorizontalLabelOrientation	XintPARALLEL_TO_AXIS int	CSG
XmNhorizontalLabelPlacement	XintPLACEMENT_TOP_BOTTOM int	CSG
XmNhorizontalScrollIncrement	0 int	CSG
XmNlabelForeground	black Pixel	CSG
XmNleftMargin	60 int	CSG
XmNnumberOfColumns	16 int	CSG
XmNnumberOfRows	16 int	CSG
XmNnumberOfVisibleColumns	dynamic int	CSG
XmNnumberOfVisibleRows	dynamic int	CSG

Name (continued)	**Default** **Type**	**Access**
XmNoverrideTextTranslations	True Boolean	CSG
XmNreadOnlyCellColor	dynamic Pixel	CSG
XmNreferenceChar	‘W’ char	CSG
XmNreferenceFontIndex	-1 int	CSG
XmNrightMargin	60 int	CSG
XmNrowAnnotationData	NULL char **	CSG
XmNrowAnnotationDataFormat	“%s” char *	CSG
XmNrowAnnotationDataType	XintTYPE_STRING int	CSG
XmNrowAnnotationFont	“fixed” char *	CSG
XmNrowAnnotationForeground	dynamic Pixel	CSG
XmNrowAnnotationTranslations	NULL XtTranslations	CSG
XmNrowCallback	NULL XtCallbackList	C
XmNselectCellCallback	NULL XtCallbackList	C
XmNselectionScroll	True Boolean	CSG
XmNshowAnnotationGridLines	False Boolean	CSG
XmNshowTextOverflowMarker	False Boolean	CSG
XmNspanCellPointer	True Boolean	CSG
XmNspanMode	XintSPAN_NONE int	CSG

Name (continued)	Default Type	Access
XmNtableFont	“fixed” char *	CSG
XmNtableForeground	dynamic Pixel	CSG
XmNtextOverflowMarkerColor	white Pixel	CSG
XmNtextOverflowMarkerSize	5 int	CSG
XmNtextThreeD	False Boolean	CSG
XmNtitleAlignment	XintALIGNMENT_CENTER int	CSG
XmNtitleBackground	default background Pixel	CSG
XmNtitleFont	“*courier*bold-r*140*” char *	CSG
XmNtitleForeground	black Pixel	CSG
XmNtitlePlacement	XintPLACEMENT_TOP int	CSG
XmNtitleShadowThickness	3 int	CSG
XmNtitleShadowType	XintSHADOW_NONE int	CSG
XmNtitleString	NULL char *	CSG
XmNtopMargin	70 int	CSG
XmNtraverseCellCallback	NULL XtCallbackList	C
XmNuseOriginalData	False Boolean	CG
XmNvalidateValueCallback	NULL XtCallbackList	C
XmNverticalAnnotationPlacement	XintPLACEMENT_LEFT_RIGHT int	CSG

Name (continued)	**Default** **Type**	**Access**
XmNverticalCellMargin	3 int	CSG
XmNverticalLabel	NULL char *	CSG
XmNverticalLabelAlignment	XintALIGNMENT_CENTER int	CSG
XmNverticalLabelFont	"*courier*bold-r*120*" char *	CSG
XmNverticalLabelOrientation	XintPARALLEL_TO_AXIS int	CSG
XmNverticalLabelPlacement	XintPLACEMENT_LEFT_RIGHT int	CSG
XmNverticalScrollIncrement	0 int	CSG

XmNadjustTextMaxLength

Specifies whether the EditTable widget should prevent the user from entering more characters than the column width when editing a cell.

XmNallowPartialCellDisplay

Specifies whether or not the contents of partially visible columns are displayed.

XmNasciiFilename

Specifies the name of an ASCII file that provides input to the EditTable widget. EditTable will automatically convert the values in the file to the column format. If a conversion fails, the cell will be left unchanged. Also, the EditTable widget will be automatically resized to the size of the dataset contained in the file. Refer to *"XintEditTableReadAscii"* on page 175 for more information on the file format.

XmNautoColumnRowMove

Specifies whether to shift column/row automatically when inserting a column/row before the first visible column/row.

XmNautoMarginAdjust

Specifies which of the values of the margin resources (**XmNtopMargin**, **XmNbottomMargin**, **XmNleftMargin** and **XmNrightMargin**) are automatically determined by the widget. This resource is inherited from class XintCompBase. You can specify one of the following constants:

Defined Constant	Description
XintADJUST_ALL	All margins will be automatically computed by the widget.
XintADJUST_NONE (default)	None of the margins will be automatically computed by the widget.
XintADJUST_LEFT	The left margin will be automatically computed by the widget.
XintADJUST_RIGHT	The right margin will be automatically computed by the widget.
XintADJUST_TOP	The top margin will be automatically computed by the widget.
XintADJUST_BOTTOM	The bottom margin will be automatically computed by the widget.

Alternatively, you can specify a combination of the constants by using a logical OR operation or an arithmetic addition operation. For instance, to have the left and right margin sized automatically and the top and bottom margin sized as specified with the **XmNtop** and **XmNbottom** resources, you would set **XmNautoMarginAdjust** to XintADJUST_LEFT | XintADJUST_RIGHT or XintADJUST_LEFT + XintADJUST_RIGHT.

This resource has no effect unless the EditTable is the child of an INT Scroll widget.

XmNautoScrollingInterval

Specifies the scrolling interval, in millisecond, when dragging a cell/column/row outside the visible Table window.

XmNautoTextOverflowMarkerSize

Specifies whether to adjust the size of the text overflow marker automatically when resizing a cell.

XmNautomaticColumnAnnotation

Specifies whether the widget will automatically annotate each column when it is created and displayed. The default value is True, indicating that the columns will be automatically annotated when displayed using the series of labels: A, B, C,...,Z, AA, AB,...,etc.

This resource must be set to False if the **XmNcolumnAnnotationData** resource is specified.

XmNautomaticRowAnnotation

Specifies whether the widget will automatically annotate each row when it is created and displayed. The default value is True, indicating that the rows will be automatically annotated using sequential integers beginning at 1.

XmNbottomMargin

Specifies the space, in pixel units, to allocate for the bottom column annotation area. This resource is calculated automatically if resource **XmNautoMarginAdjust** is set to XintADJUST_ALL or XintADJUST_BOTTOM.

XmNcellAttributesCallback

Specifies the callback that is called just before the cell is drawn. This callback allows the programmer to change the cell attributes, such as background, foreground, alignment, and font table index of a cell dynamically. The constant values available for attribute alignment are listed below in resource **XmNcolumnAlignmentData**. This callback will be called for every visible cell in the table.

XmNcellHeightData

Specifies an integer array, as large as the number of rows (or columns if the table is transposed), which defines the height of each row in the unit system specified by resource **XmNcellSizeUnit**. If resource **XmNcellHeightData** is not specified (NULL), the height of all the rows in the table will be set to the size specified in resource **XmNdefaultCellHeight**.

XmNcellHighlightColor

Specifies the color used to draw the background of one or of a group of selected cells.

XmNcellPointerBorderThickness

Specifies the thickness of the cell pointer in pixel units. The cell pointer is a rectangle drawn around the cell that is being edited.

XmNcellPointerColor

Specifies the color of the cell pointer rectangle.

XmNcellPointerRetained

Specifies whether the cell pointer is kept or not when the table loses focus.

XmNcellSizeUnit

Specifies the unit system used to specify the size of a cell. Specify XintUNIT_CHARACTER (default) to have the cell size in character units or XintUNIT_PIXEL to have the cell size in pixels.

XmNcellWidgetDisplayCallback

Specifies the list of callbacks that is called just before a ‘widget in a cell’ is drawn or mapped to a cell. ‘Widget in a cell’ is the mechanism that permits a single widget to be used across a range of cells in the edit table. These callbacks allow the application to query data and to configure the inserted widget with the proper resources before it is drawn to the current cell location. This callback is used whenever **XmNcellWidgetSetResources** is False, or when the widget requires more or different resources.

XmNcellWidthData

Specifies an integer array, as large as the number of columns (or rows if the table is transposed), which defines the width of each column in the unit system defined by resource **XmNcellSizeUnit**. If resource **XmNcellWidthData** is not specified (NULL), the width of all the columns in the table will be set to the size specified in resource **XmNdefaultCellWidth**.

XmNcheckEditModeCallback

Specifies the list of callbacks that is called when the end user has selected a cell for editing by using the action EditTableEditCell().

XmNclipAnnotation

Specifies whether the annotation will be clipped if the annotation string is wider than the annotation area. If the column orientation is vertical, then clipping will occur only for the column annotation. If the column orientation is horizontal, then clipping will occur only for the row annotation. The default value for this resource is False, indicating that the annotation will not be clipped, but rather the width of the column will be increased to accommodate the width of the annotation. If you set this resource to True, then the annotation will be clipped and the column width will not be changed.

XmNcolumnAlignmentData

Specifies an array (of size **XmNnumberOfColumns**) containing the alignment specifications for each column. Use the defined integer constants listed below for specifying the alignment.

Resource Value	Description
XintALIGNMENT_BEGINNING_TOP	The value in each cell of the column is to be justified in the upper left hand corner of the cell.
XintALIGNMENT_CENTER_TOP	The value in each cell of the column is to be justified horizontally in the center of the cell and vertically at the top of the cell.
XintALIGNMENT_END_TOP	The value in each cell of the column is to be justified in the upper right hand corner of the cell.
XintALIGNMENT_BEGINNING_MIDDLE (default)	The value in each cell of the column is to be justified horizontally at the left side of the cell and vertically in the center of the cell.
XintALIGNMENT_CENTER_MIDDLE	The value in each cell of the column is to be justified horizontally at the center of the cell and vertically in the center of the cell.
XintALIGNMENT_END_MIDDLE	The value in each cell of the column is to be justified horizontally at the right side of the cell and vertically in the center of the cell.
XintALIGNMENT_BEGINNING_BOTTOM	The value in each cell of the column is to be justified in the lower left hand corner of the cell.
XintALIGNMENT_CENTER_BOTTOM	The value in each cell of the column is to be justfied horizontally in the center of the cell and vertically at the bottom of the cell.
XintALIGNMENT_END_BOTTOM	The value in each cell of the column is to be justified in the lower right hand corner of the cell.

Use the function XintEditTableDefineColumnFormat or the resource **XmNdefaultColumnAlignment** to set the alignment of a column. If you set the resource **XmNcolumnAlignmentData** using a resource file, specify a list of values consisting of the defined constants (in the table above) separating the values with commas.

XmNcolumnAnnotationData

Specifies the character strings to use for annotating the columns in the table. You set this resource's value by specifying a pointer to an array of pointers, each of which refers to a character string containing a column annotation. If a value is specified for this resource then the EditTable widget will make its own copy of the column annotations from the array at its creation time, regardless of the setting of the value of the resource **XmNuseOriginalData**. If this resource is set from a resource file, use commas to separate the strings. The number of strings must be equal to the number of columns defined by **XmNnumberOfColumns**.

If this resource is specified, the **XmNautomaticColumnAnnotation** resource must be set to False.

XmNcolumnAnnotationFont

Specifies the font used to draw the column annotations.

XmNcolumnAnnotationForeground

Specifies the foreground color used for column annotations.

XmNcolumnAnnotationTranslations

Specifies a (parsed) translation table defining the translations for the annotation action routines. Use this resource only when the parent of EditTable widget is an INT Scroll widget. When the parent of the EditTable widget is not an INT Scroll widget, then register these translations with the EditTable widget with an XtAugmentTranslations or an XtOverrideTranslations function.

XmNcolumnCallback

Specifies the list of callbacks that is called when the end user has executed any column operation action routine (such as column select, insert or delete).

XmNcolumnDataFormatData

Specifies an array (of size **XmNnumberOfColumns**) containing strings specifying the display format for each column. The display format is specified using C format descriptors such as "%f". The format descriptor must be compatible with the data type for that column. If you need to display a column of data using a non standard format, set the format descriptor to NULL and register callback XmNformatCellCallback.

Also, for columns whose data type is set to pointer, the format descriptor will be ignored and callback XmNformatCellCallback will be called. The display format of a column can also be specified using function XintEditTableDefineColumnFormat. If you set this resource from a resource file, separate the format strings by commas.

XmNcolumnDataTypeData

Specifies an array of size **XmNnumberOfColumns** containing integer values specifying the data type for each column of the table. Use the constants listed below for specifying the data type.

Resource Value	Description
XintTYPE_SHORT	Specifies short integer data type.
XintTYPE_INTEGER	Specifies integer data type.
XintTYPE_LONG	Specifies long integer data type.
XintTYPE_FLOAT	Specifies floating point data type.
XintTYPE_DOUBLE	Specifies double precision floating point data type.
XintTYPE_STRING (default)	Specifies character string data type.
XintTYPE_POINTER	Specifies the pointer data type.

The data type of a column can also be specified using function XintEditTableDefineColumnFormat or resource **XmNdefaultColumnDataType**. If you set resource **XmNcolumnDataTypeData** from a resource file, use the string names or integer values listed in the table above. Multiple values must be separated by commas.

XmNcolumnEditModeData

Specifies an array of size **XmNnumberOfColumns** containing integer values specifying the edit mode for each column of the table. Specify XintCOLUMN_EDITABLE if you want the cells of a column to be editable or XintCOLUMN_NON_EDITABLE if you want the cells of a column to be non editable. The edit mode of a column can also be set using resource **XmNdefaultColumnEditMode**. Refer to t*"XmNcheckEditModeCallback"* on page 92 for more information about selectively enabling or disabling the edit mode of a cell. When specifying this resource via a resource file, place commas between the elements of the list.

XmNcolumnFontIndexData

Specifies an array of size **XmNnumberOfColumns** containing integer values specifying the font index to use for each column of the table. Resource **XmNfontTable** specifies a list of X fonts that can be used for displaying the cell contents. The index value corresponds to an entry in that table. Specify -1 for columns that use the default font specified in **XmNtableFont**. Use function XintEditTableSetCellFont to specify fonts on a cell-by-cell basis.

XmNcolumnOrientation

Specifies whether the columns are oriented horizontally across the page or vertically down the page. The default is vertical orientation specified by XintCOLUMN_VERTICAL. Specify XintCOLUMN_HORIZONTAL for horizontal orientation.

XmNdefaultCellHeight

Specifies the default height of a cell in the unit system defined by resource **XmNcellSizeUnit**. Specify a positive integer as the value of this resource. Alternatively, if you specify the integer 0 as the value of this resource, the EditTable widget will compute the height of each row as the maximum number of lines of text in any character string formatted cell in that row (can be quite slow for big tables).

XmNdefaultCellWidth

Specifies the default width of a cell in the unit system defined by resource **XmNcellSizeUnit**. If you specify the integer 0 as the value of this resource, the EditTable widget will compute the width of each column as the maximum number of characters in any character string formatted cell in that column.You can change the width of a column using the function XintEditTableDefineColumnFormat or use resource **XmNcellWidthData** to set the width of all the columns in the table.

XmNdefaultColumnAlignment

Specifies the alignment of the cell contents for newly created columns. Refer to *"XmNcolumnAlignmentData"* on page 93 for a list of defined constants used to specify the value of this resource.

XmNdefaultColumnDataFormat

Specifies the default display format for the cells in the table. Specify the format value as any C format descriptor (such as %s, %d, %f) that corresponds to the data type specified by the value of the resource **XmNdefaultColumnDataType**. Specify NULL if you want to handle the formatting of the data yourself using callback XmNformatCellCallback.

XmNdefaultColumnDataType

Specifies the data type of the values of the cells in a column that will be used when a new column is created. You can specify one of the following defined constants:

Resource Value	Description
XintTYPE_SHORT	Specifies short integer data type.
XintTYPE_INTEGER	Specifies integer data type.
XintTYPE_LONG	Specifies long integer data type.

Resource Value	Description
XintTYPE_FLOAT	Specifies floating point data type.
XintTYPE_DOUBLE	Specifies double precision data type.
XintTYPE_STRING (default)	Specifies character string data type.
XintTYPE_POINTER	Specifies pointer data type.
XintTYPE_NONE	Indicates that no data is to be managed by EditTable.

XmNdefaultColumnEditMode

Specifies the edit mode for newly created columns. The default value for this resource is XintCOLUMN_EDITABLE indicating that the values in a newly created column can be edited by the end user. Specify the constant XintCOLUMN_NON_EDITABLE if you want to prohibit end-user editing in newly created columns.

XmNdoubleClickCallback

Specifies a list of callbacks that is called when the end user performs a double-click operation on a cell. The callback structure is XintEditTableDoubleClickCallbackStruct and the reason is XintCR_DOUBLE_CLICK.

XmNdoubleClickInterval

Specifies the maximum interval, in milliseconds, between which two button clicks are considered to be a double-click action rather than two single-click actions. The default is the value returned by function XtGetMultiClickTime.

XmNdragCallback

Specifies the list of callbacks that is called when the end user performs a move, resize or copy operation on a cell, row or column.

XmNdragCursorType

Specifies the cursor type to use when performing an interactive copy, move or resize operation on a row, column or cell.

XmNdragForeground

Specifies the color, as a pixel value, used to draw the outline of a row, column or cell while performing an interactive copy, move or resize operation.

XmNdragGridLineStyle

Specifies the style used to draw the outline of a row, column or cell while performing an interactive copy, move or resize operation. Use one of the following defined constants when specifying a value for this resource:

Resource Value	Description
XintGRID_LINE_SOLID (default)	The outline is drawn using a solid line.
XintGRID_LINE_DASHED	The outline is drawn using a dashed line.
XintGRID_LINE_DOUBLE_DASHED	The outline is drawn using a double dashed line.
XintGRID_LINE_NONE	No outline is drawn.

XmNdragShowCellContent

Specifies whether the cell values should be displayed, along with the outline, while performing an interactive copy, move or resize operation on a row, column or cell.

XmNeditAnnotationCallback

Specifies the list of callbacks that is called when the end user has selected a row or column annotation for editing via the action AnnotationEdit().

XmNfirstVisibleColumn

Specifies the first column to display when the table is created.

XmNfirstVisibleRow

Specifies the first row to display when the table is created.

XmNfontTable

Specifies a NULL terminated list of X font names that can be used for setting fonts for individual cells, rows or columns. Once this table is defined, fonts can be assigned with resource **XmNcolumnFontIndexData** or functions XintEditTableSetCellFont, XintEditTableSetColumnFont and XintEditTableSetRowFont. Specify NULL if you want the whole table to use the font specified in **XmNtableFont**.

XmNformatCellCallback

Specifies the list of callbacks that can be used to format the data before display. This callback is invoked when the format specifier for a column is set to NULL or the data type for a column is XintTYPE_POINTER.

XmNformatColumnAnnotationCallback

Specifies the list of callbacks that can be used to input the column annotation before display. This resource will have no effect if **XmNautomaticColumnAnnotation** is True.

XmNformatRowAnnotationCallback

Specifies the list of callbacks that can be used to input the row annotation before display. This resource will have no effect if **XmNautomaticRowAnnotation** is True.

XmNfreezeUpdate

This resource can be used to temporarily disable (True) any geometry update and redrawing before performing a series of changes on an EditTable widget. The resource value should be set back to False after the changes are made so that the table can automatically calculate its new geometry and redisplay itself.

XmNfrozenColumnPlacement

Specifies the placement of a frozen column. Specify XintPLACEMENT_LEFT or XintPLACEMENT_RIGHT if the columns are oriented vertically. Specify XintPLACEMENT_TOP or XintPLACEMENT_BOTTOM if the columns are orientated horizontally.

XmNfrozenRowPlacement

Specifies the placement of a frozen row. Specify XintPLACEMENT_TOP or XintPLACEMENT_BOTTOM if the rows are oriented horizontally. Specify XintPLACEMENT_LEFT or XintPLACEMENT_RIGHT if the rows are orientated vertically.

XmNgridLineForeground

Specifies foreground color used to draw grid lines when grid line style is set to solid or dashed.

XmNgridLineHighlightColor

Specifies color used for the grid lines that outline a block of selected cells.

XmNgridLineHighlightThickness

Specifies width (in pixels) of the highlight border drawn around one cell or a group of cells when they are selected. The thickness defined using this resource will be restricted to be at most half of the grid line width specified in resource **XmNgridLineWidth**. Use **XmNgridLinelHighlightColor** to define the color for the highlight border.

XmNgridLineOrientation

Specifies the orientation of grid lines.Use one of the following defined constants when specifying a value for this resource:

Resource Value	Description
XintGRID_LINE_CROSSWISE (default)	Grid lines are drawn to separate rows and columns.
XintGRID_LINE_COLUMNWISE	Grid lines are drawn between columns only.
XintGRID_LINE_ROWWISE	Grid lines are drawn between rows only.

XmNgridLineStyle

Specifies line style used to display grid lines separating cells in the table. Use one of the following constants when specifying a value for this resource:

Resource Value	Description
XintGRID_LINE_SOLID (default)	Cells in table will be separated by solid line.
XintGRID_LINE_DASHED	Cells in table will be separated by dashed line.
XintGRID_LINE_DOUBLE_DASHED	Cells in table will be separated by double dashed line.
XintGRID_LINE_SHADOW_IN	Cells in table will have a shadow border that goes into the screen.
XintGRID_LINE_SHADOW_OUT	Cells in table will have a shadow border that goes out of the screen.

XmNgridLineWidth

Specifies width (in pixels) of the grid lines drawn between the cells in a table.

XmNhorizontalAnnotationPlacement

Specifies placement of horizontal annotation. You can use the following constants:

Defined Constant	Description
XintPLACEMENT_NONE	No horizontal annotation displayed.
XintPLACEMENT_TOP	Horizontal annotation is displayed above the table.
XintPLACEMENT_BOTTOM	Horizontal annotation is displayed below the table.
XintPLACEMENT_TOP_BOTTOM	Horizontal annotation is displayed both above and below the table.

XmNhorizontalCellMargin

Specifies the width (in pixels) of the margin between the left side of a cell's value and the left side of the cell. The same value is used for the margin between the right side of a cell's value and the right side of the cell.

XmNhorizontalLabel

Specifies a single or multiple line string that provides labelling for the horizontal annotation. The label placement is controlled by the resource **XmNhorizontalLabelPlacement**.

XmNhorizontalLabelAlignment

Specifies the alignment of each line of the horizontal label. Multiple lines can be specified by inserting the special character '\n' between each line in the label string. Label lines are adjusted in respect to the horizontal label area, which is the same width as the widget's display area. You can specify one of the following constants:

Defined Constant	Description
XintALIGNMENT_BEGINNING	Lines of the label are left aligned.
XintALIGNMENT_CENTER	Lines of the label are centered.
XintALIGNMENT_END	Lines of the label are right aligned.

XmNhorizontalLabelFont

Specifies the name of a font used to draw the horizontal label specified by resource **XmNhorizontalLabel**.

XmNhorizontalLabelOrientation

Specifies the orientation of the label string. Specify constant XintPARALLEL_TO_AXIS (default) to have the label placed along the horizontal axis, or specify constant XintSTACKED to have the letters of the label stacked vertically.

XmNhorizontalLabelPlacement

Specifies the placement of the horizontal label. You can specify one of the following constants:

Defined Constant	Description
XintPLACEMENT_TOP	The horizontal label is drawn above the widget's display area
XintPLACEMENT_BOTTOM	The horizontal label is drawn below the widget's display area.
XintPLACEMENT_TOP_BOTTOM	The horizontal label is drawn both above and below the widget's display area.

XmNhorizontalScrollIncrement

Specifies the number of pixels the EditTable widget will scroll horizontally when the left or right scrollbar arrow is pressed. Specify 0 (default) to have the EditTable scroll an entire cell width.

XmNlabelForeground

Specifies the pixel used to draw the horizontal and vertical labels.

XmNleftMargin

Specifies the space, in pixels units, to allocate for the left row annotation area. This resource is calculated automatically if resource **XmNautoMarginAdjust** is set to XintADJUST_ALL or XintADJUST_LEFT.

XmNnumberOfColumns

Specifies the number of columns in the table.

XmNnumberOfRows

Specifies the number of rows in the table.

XmNnumberOfVisibleColumns

Specifies how many columns should be visible (size of the viewport), when the EditTable widget is created as a child of a INT Scroll widget. Specify 0 (default) to have all the columns in the table visible. This resource takes effect only if parent scroll resource **XmNhorizontalAutoSized** is set to True.

XmNnumberOfVisibleRows

Specifies how many rows should be visible (size of the viewport), when the EditTable widget is created as a child of a INT Scroll widget. Specify 0 (default) to have all the rows in the table visible. This resource will take effect only if the parent scroll resource **XmNverticalAutoSized** is set to True.

XmNoverrideTextTranslations

Specifies if the table overrides the text translations for the Return, Left, Right, Up and Down keys, so that they can be used to navigate between cells inside the table. This resource is usually set to True. Set it to False to be able to edit multiline text in a cell.

XmNreadOnlyCellColor

Specifies the default background color of the cells in a non editable column.

XmNreferenceChar

Specifies the character whose width is used to compute the width of a column that is displaying string data. This resource is only used when fonts containing characters of different widths are used (proportional fonts).

XmNreferenceFontIndex

Specifies an index corresponding to an entry in the font table (specified in resource **XmNfontTable**) that is used to calculate the dimension of each column in the table. This resource only applies if resource **XmNfontTable** is not NULL and should be usually set to the index corresponding to the largest font. The default value for this resource is -1, which means that the table will calculate the size of each column based on the font used by each cell. This process can be slow for large tables.

XmNrightMargin

Specifies the space, in pixels units, allocated for the right row annotation area. This resource is calculated automatically if resource **XmNautoMarginAdjust** is set to XintADJUST_ALL or XintADJUST_RIGHT.

XmNrowAnnotationData

Specifies the values to use for annotating the rows in the table. You set this resource's value by specifying a pointer to an array containing the row annotation values. The row annotation can be any of the supported data types for cell data. If a value is specified for this resource then the EditTable widget will make its own copy of the row annotations in the array at widget creation time regardless of the setting of the value of the resource **XmNuseOriginalData**. The number of values in the array must be equal to the number of rows defined by the **XmNnumberOfRows** resource.

XmNrowAnnotationDataFormat

Specifies the format of the values used for the row annotation. You specify the value as any C format descriptor (such as %s, %d, %f) that corresponds to the data type specified by the value of the resource **XmNrowAnnotationDataType**.

XmNrowAnnotationDataType

Specifies the data type of the values used for row annotation. You can specify one of the following defined constants:

Resource Value	Description
XintTYPE_SHORT	Specifies short integer data type.
XintTYPE_INTEGER (default)	Specifies integer data type.
XintTYPE_LONG	Specifies long integer data type.
XintTYPE_FLOAT	Specifies floating point data type.
XintTYPE_DOUBLE	Specifies double precision floating point data type.
XintTYPE_STRING	Specifies character string data type.

XmNrowAnnotationFont

Specifies the font used to draw the row annotations.

XmNrowAnnotationForeground

Specifies the foreground color used for row annotations.

XmNrowAnnotationTranslations

Specifies the parsed translation table defining the translations for the row annotation action routines. Use this resource only when the parent of EditTable widget is an INT Scroll widget. When the parent of the EditTable widget is not an INT Scroll widget, then do not define the row annotation translations using this resource, but include those translations in a translation table registered with the EditTable widget by using an XtAddTranslations call.

XmNrowCallback

Specifies the list of callbacks that is called when the end user has executed any row operation action routine (e.g. row select, insert or delete).

XmNselectCellCallback

Specifies the list of callbacks that is called when the end user has selected a cell using any cell selection action routine (such as EditTableEndSelect).

XmNselectionScroll

Specifies whether the EditTable will automatically scroll to show the area that has been selected via the selection convenience function XintEditTableSetSelection.

XmNshowAnnotationGridLines

Specifies whether grid lines should be drawn in the annotation area.

XmNshowTextOverflowMarker

Specifies whether to show the text overflow marker when the width /height of a cell is less than the actual dimensions of the displayed text.

XmNspanCellPointer

Specifies whether or not the cell pointer will treat spanned cell row/column locations as a single location and skip the individual locations that have been overlapped by the span. If True (the default), the cell pointer will skip overlapped locations and treat the cell as a single location. If False, the cell pointer will visit every row/column location, regardless of whether or not that location is spanned.

XmNspanMode

Specifies whether or not cell spanning is enabled. **XmNspanMode** must be specified as one of the following constants.

Defined Constant	Description
XintSPAN_NONE	Spanning is not enabled (the default).
XintSPAN_ALWAYS	Spanning is enforced.

Defined Constant	Description
XintSPAN_DATA_ONLY	Cells with data are drawn according to their span factor. Empty cells, regardless of their span, are drawn at 1x1.
XintSPAN_DATA_AND_EMPTY	Cells are drawn according to their span factor, regardless of whether or not they have data.

XmNtableFont

Specifies the font used to draw the values of all cells in the table.

XmNtableForeground

Specifies the color (as a pixel value) used when displaying the values of the cells in the table.

XmNtextOverflowMarkerColor

Specifies the color (pixel), of the text overflow marker. The color white is the default color.

XmNtextOverflowMarkerSize

Specifies the size of the text overflow marker, in pixel units. The default size is 5 pixels wide.

XmNtextThreeD

Specifies whether the widget's labels and title are drawn with a shadow, so that they have a 3-dimensional appearance. The default is False, indicating that they will not appear 3-dimensional.

XmNtitleAlignment

Specifies the alignment of the lines of the title if it has more than one line. Multiple lines can be specified by inserting special character '\n' between lines in the title string. You can specify one of the following constants:

Defined Constant	Description
XintALIGNMENT_BEGINNING	Lines of the title are left adjusted.
XintALIGNMENT_CENTER	Lines of the title are centered.
XintALIGNMENT_END	Lines of the title are right adjusted.

XmNtitleBackground

Specifies the pixel value used to draw the background of the title.

XmNtitleFont

Specifies the name of the font used to draw the title.

XmNtitleForeground

Specifies the pixel value used to draw the foreground of the title.

XmNtitlePlacement

Specifies the placement of the title string. Specify one of the following constants:

Defined Constant	Description
XintPLACEMENT_NONE	The title is not drawn.
XintPLACEMENT_TOP (default)	The title is drawn above the widget's window.
XintPLACEMENT_BOTTOM	The title is drawn at the bottom of the widget's window.

XmNtitleShadowThickness

Specifies the thickness of the title shadow in pixels if resource **XmNtitleShadowType** is not set to XintSHADOW_NONE.

XmNtitleShadowType

Specifies the type of shadow to draw around the title. You can specify one of the following constants:

Defined Constant	Description
XintSHADOW_NONE	No shadow is drawn.

Defined Constant	Description
XintSHADOW_IN	Shadow drawn so that title appears inset.
XintSHADOW_OUT	Shadow drawn so that tile appears outset.
XintSHADOW_ETCHED_IN	Shadow drawn using a double line inset.
XintSHADOW_ETCHED_OUT	Shadow drawn using a double line outset.

XmNtitleString

Specifies the string used to draw the title. The color, font and location of the title string can be specified using resources **XmNtitleFont**, **XmNtitleForeground** and **XmNtitlePlacement**.

XmNtopMargin

Specifies the space, in pixels units, reserved for the top column annotation area. This resource is automatically calculated by the widget when the value of the resource **XmNautoMarginAdjust** is XintADJUST_ALL or XintADJUST_TOP.

XmNtraverseCellCallback

Specifies the list of callbacks that is called when the end-user has executed the EditTableEnterCell() action routine.

XmNuseOriginalData

Indicates whether the EditTable widget should use the application's table data or whether the EditTable widget should create a copy of the application's table data. The default is False, indicating that the EditTable widget should make its own copies of the application's table data. Table data includes the values of all cells in the table.

If you set this resource to True, insertion and deletion of rows (or columns if the table is transposed) will be disabled because it is not possible for the table to reallocate your data. When resource **XmNuseOriginalData** is True, you should use function XintEditTableFillColumnData to fill the table.

Note: This resource can only be set at creation time.

XmNvalidateValueCallback

Specifies the list of callbacks to be called so that the application can validate the input when the end user has changed the value of a cell.

XmNverticalAnnotationPlacement

Specifies placement of vertical annotation, using the following constants:

Defined Constant	Description
XintPLACEMENT_NONE	No vertical annotation displayed.
XintPLACEMENT_RIGHT	Vertical annotation is displayed to the right of the table.
XintPLACEMENT_LEFT	Vertical annotation is displayed to the left of the table.
XintPLACEMENT_LEFT_RIGHT (default)	Vertical annotation is displayed both left and right of the table.

XmNverticalCellMargin

Specifies height (in pixels) of margin between top edge of cell's value and top edge of the cell. The same value is used for the margin between the bottom edge of a cell's value and the bottom edge of the cell.

XmNverticalLabel

Specifies single or multiple line string that provides labelling for the vertical annotation. Label placement controlled by **XmNverticalLabelPlacement**.

XmNverticalLabelAlignment

Specifies alignment of each line of the vertical label. Specify multiple lines by inserting special character '\n' between lines in the label string. Label lines are adjusted with respect to vertical label area, which is the same height as the widget's display area. For a vertical label, beginning alignment indicates lines justified to top of the widget's display area and continue downward. End alignment indicates lines justified to bottom of the widget's display area and continue upward. Specify with one of the following constants:

Defined Constant	Description
XintALIGNMENT_BEGINNING	Lines of the label are aligned with the top of the widget's display area.
XintALIGNMENT_CENTER	Lines of the label are centered.
XintALIGNMENT_END	Lines of the label are aligned with the bottom of the widget's display area.

XmNverticalLabelFont

Specifies the name of a font used to draw the vertical label specified by the resource **XmNverticalLabel**.

XmNverticalLabelOrientation

Specifies orientation of label string. Specify constant XintPARALLEL_TO_AXIS To rotate the label 90 degrees from horizontal and place it along the vertical axis. Specify constant XintSTACKED to stack the letters of the label vertically.

XmNverticalLabelPlacement

Specifies placement of vertical label, according to the following constants:

Defined Constant	Description
XintPLACEMENT_LEFT	Vertical label drawn left of widget's display area.
XintPLACEMENT_RIGHT	Vertical label drawn right of widget's display area.
XintPLACEMENT_LEFT_RIGHT	Vertical label drawn both left and right of widget's display area.

XmNverticalScrollIncrement

Specifies the number of pixels the EditTable widget will scroll vertically when the up or down scrollbar arrow is pressed. Specify 0 (default) to have the EditTable scroll an entire cell height.

Constraint Resources

The following constraint resources can be specified on a widget inserted in an EditTable widget. They specify the range of cells into which the widget will be inserted and whether or not some resources will be automatically applied when the widget is mapped to a cell.

Name	Default Type	Access
XmNcellWidgetRange	NULL XintCellWidgetRange *	CSG
XmNcellWidgetSetResources	True Boolean	CSG
XmNcellWidgetOverrideTranslations	True Boolean	CSG

XmNcellWidgetRange

Specifies the range of EditTable cells that will contain the designated widget. This resource is a pointer to a structure as shown below.

```
typedef struct {
int row, rows, column, columns;
} XintCellWidgetRange;
```

where the structure variables are:

Member	Description
row	Starting row in the range of cells that contain the widget.
rows	Number of rows in the range of cells that contain the widget. 0 = all rows in the table, beginning with the starting row.
column	Starting column in the range of cells that contain the widget.
columns	Number of columns in the range of cells that contain the widget. 0 = all columns in the table, beginning with the starting column.

XmNcellWidgetSetResources

Specifies whether or not the EditTable will automatically set a number of cell resources for the most common Motif widgets to the values shown below.

Resource Name	Automatic Setting
XmNlabelString	Current cell content, as a string.
XmNfontList	Current cell font.
XmNbackground	Current cell background.
XmNforeground	Current cell foreground.
XmNalignment	Current cell alignment.
XmNsensitive	Current cell sensitivity.

Before calling XmNcellWidgetDisplayCallback, set **XmNcellWidgetSetResources**. If set to False, and XmNcellWidgetDisplayCallback is omitted, the visual characteristics of the cells will be undefined.

XmNcellWidgetOverrideTranslations

Specifies whether or not to change or replace the translations of the widget in a cell so that arrows and tab keys take you back to the table process rather than the focus remaining in the mapped widget.

Widget in a Cell Example

The following code fragment illustrates the ease of installing Widget In A Cell and setting its resources. Since **XmNcellWidgetSetResources** is True by default, the cell values will be set automatically.

Code

```
/*
 * Create the edit table
 */
edit_table = XtCreateManagedWidget(“edit_table”,
             xintEditTableWidgetClass, scroll, arg, n);

/*
 * Put a widget into cells that span 400 rows and 2 columns
 */
range.row     = 1;
range.rows    = 400;
range.column  = 1;
range.columns = 2;
XtVaCreateWidget(“pushb”, xmPushButtonWidgetClass,
             edit_table, XmNcellWidgetRange, &range, NULL);
```

For a more complete example which illustrates the use of a callback to set resources, see *“Example 4: Widget In A Cell”* on page 216.

EditTable Data

Data structures

The values of every cell in a table can be created and maintained entirely by the EditTable widget. Alternatively, the application can define and manage the data structures holding the values of every cell in the table. The **XmNuseOriginalData** resource needs to be set to True at EditTable widget creation time if the application wants to share the table data with the widget. In either case, the EditTable creates and manages the data structures for the row annotation and for the column annotation.

Application-defined Data Structures

If the application defines and maintains the data structures for the cell values, then the data structures required are a collection of arrays with the same number of elements; one array for every column in the table. The application will need to pass the address of each array to the EditTable by using the XintEditTableFillColumnData function. If the end user makes a change in the value of a cell, then the EditTable widget will update the application's data with the new value. When the application directly changes the values in one or more cells in a column, then the application must use the function XintEditTableUpdateDataDisplay to signal the EditTable widget that the display of a column needs to be updated. When the application shares the table data with the widget, there are restrictions on the operations that can be performed on the table. For instance, rows can not be deleted from the table. When the end user deletes a column, the application will need to use a delete column callback so that it is aware that the column has been deleted from the table.

Specifying Undefined Values

If the value of a cell is undefined, EditTable will display an empty cell. To specify an undefined value for a cell, use the following defined constants:

Defined Constant	Description
XintUNDEFINED_SHORT	Specifies undefined short integer value.
XintUNDEFINED_INTEGER	Specifies undefined integer value.
XintUNDEFINED_LONG	Specifies undefined long integer value.
XintUNDEFINED_FLOAT	Specifies undefined floating point value.
XintUNDEFINED_DOUBLE	Specifies undefined double precision floating point value.
XintUNDEFINED_STRING	Specifies undefined character string.
XintUNDEFINED_POINTER	Specifies NULL pointer.

EditTable Actions

The following action procedures are defined by the EditTable widget for manipulating a table. These action procedures can be tied to user actions via a translation table.

Name	Description
AnnotationEdit()	Calls the callback list specified by the resource **XmNeditAnnotationCallback**.
AnnotationStartSelect() or (*single*)	Initiates a row or column selection operation and clears all previous selections.
AnnotationsStartSelect(*multiple*)	Initiates a row or column selection operation and adds to the previous selections.
AnnotationExtendSelect()	Extends the existing row or column selections to a list of rows or columns according to the position of the mouse pointer.
AnnotationEndSelect()	Terminates the pointer selection in the annotation area and marks the selection block. It calls the callback list specified by the resource **XmNcolumnCallback** or **XmNrowCallback**.
AnnotationResizeHandlers (*tolerance*)	Shows the row or column resizing cursor when the mousepointer is over the border between adjacent rows or columns in the annotation area. Tolerance is the maximum distance in pixels within which the resizing cursor will be shown. If it is null or not a number, the tolerance will be set to 3.
AnnotationStartDrag(*move*)	Initiates a row or column move operation.
AnnotationStartDrag(*copy*)	Initiates a row or column copy operation.
AnnotationStartDrag(*resize*)	Initiates a row or column resize operation.
AnnotationStartDrag(*move, resize*) or (*copy, resize*)	Initiates one of the specified operations on a row or column. Resize will be initiated if pointer is pressed close to the row or column border, otherwise a move or copy will be initiated.
AnnotationExtendDrag()	Extends the drag operation.
AnnotationEndDrag()	Terminates the drag operation and calls callback XmNdragCallback.
EditTableEditCell()	If a cell has been selected, this routine calls the callback list specified by the resource **XmNcheckEditModeCallback**.

Name (continued)	**Description**
EditTableConfirmEdit()	Calls the callbacks for XmNvalidateValueCallback. If the new value is approved, it is stored into the cell being edited, otherwise the old value is restored.
EditTableStartSelect() or (*single*)	Initiates selection of a block of cells and clears all previous selections.
EditTableStartSelect(*row*) or (*column*)	Initiates a row or a column selection from inside the table and clears all previous selections.
EditTableStartSelect (*multiple*)	Initiates the selection of a block of cells and adds to the previous selections.
EditTableStartSelect(*multiple, row*) or (*multiple, column*)	Initiates a row or a column selection from inside the table and adds to the previous selections.
EditTableExtendSelect()	Extends the existing cell selection to a block of cells according to the position of the mouse pointer.
EditTableEndSelect()	Terminates the pointer selection and marks the selection area. It calls the callback list specified by XmNselectCellCallback.
EditTableClearAllSelections()	Clears all existing selections.
EditTableClearSelection()	Clears the selection where the pointer resides.
EditTableEnterCell(*Down*)	Moves the cursor (highlight) to the cell below the currently selected one and calls the callback list specified by the resource **XmNtraverseCellCallback**.
EditTableEnterCell(*Left*)	Moves the cursor (highlight) to the cell to the left of the currently selected one and calls the callback list specified by the resource **XmNtraverseCellCallback**.
EditTableEnterCell(*Right*)	Moves the cursor (highlight) to the cell to the right of the currently selected one and calls the callback list specified by the resource **XmNtraverseCellCallback**.
EditTableEnterCell(*Pointer*)	Moves the cursor (highlight) to the cell that the user has clicked on and calls the callback list specified by the resource **XmNtraverseCellCallback**.
EditTableEnterCell(*Up*)	Moves the cursor (highlight) to the cell above the currently selected one and calls the callback list specified by the resource **XmNtraverseCellCallback**.
EditTableInsertColumns()	If a column has been selected, this routine inserts the column(s) on the clipboard before the selected column.
EditTableInsertRows()	If a row has been selected, this routine inserts the row(s) on the clipboard before the selected row.

Name (continued)	**Description**
EditTablePasteColumn()	If a column has been selected, this routine replaces the selected column with the column on the clipboard.
EditTablePasteRows()	If a row has been selected, this routine replaces the selected row (and possibly the rows below the selected row) with the row(s) on the clipboard.
EditTableCopyColumn()	Copies the data of the currently selected (highlighted) column to the clipboard.
EditTableCopyRows()	Copies the data of the currently selected (highlighted) rows to the clipboard.
EditTableTraverseCurrent()	Causes EditTable widget to receive input focus.
EditTableDeleteColumns()	Deletes the currently selected (highlighted) columns and saves the column data to the clipboard.
EditTableDeleteRows()	Deletes the currently selected (highlighted) rows and saves the row data to the clipboard.
EditTableUndeleteColumns()	Restores the columns that were last deleted.
EditTableUndeleteRows()	Restores the rows that were last deleted.
EditTableEnterTable()	Sets the keyboard focus to the table.
EditTableStartDrag(*copy*)	Initiates a cell copy operation.
EditTableStartDrag(*move*)	Initiates a cell move operation.
EditTableStartDrag(*resize*)	Initiates a cell resize operation.
EditTableStartDrag(*move*, *resize*) or (*copy*, *resize*)	Initiates one of the specified operations on a cell. Resize will be initiated if pointer is pressed close to the cell border, otherwise a move or copy will be initiated.
EditTableExtendDrag()	Extends a cell drag operation.
EditTableEndDrag()	Terminates the cell drag operation and calls callback XmNdragCallback.
EditTableResizeHandler()	Shows the cell resizing cursor when mouse pointer is over the border/corner between adjacent cells.
NextTabGroup()	Traverses to the next tab group.
PreviousTabGroup()	Traverses to the previous tab group.
TextConfirmEdit()	Calls callback XmNvalidateValueCallback to confirm the value entered.
TextAbandonEdit()	Cancels current editing operation on a cell and restores original value.

EditTable Translations

The default translation table defined for use with an EditTable widget is the following:

Event Sequence	**Action Name**

<FocusIn>	ManagerFocusIn()
	EditTableEnterCell(FocusIn)

<FocusOut>	ManagerFocusOut()

Shift <Key>Tab	EditTableEnterCell(Left)
	EditTableClearAllSelections()

Shift <Key>Return	EditTableEnterCell(Up)
	EditTableClearAllSelections()

None <Key>osfLeft	EditTableEnterCell(Left)
	EditTableClearAllSelections()

None <Key>osfRight	EditTableEnterCell(Right)
	EditTableClearAllSelections()

None <Key>Tab	EditTableEnterCell(Right)
	EditTableClearAllSelections()

None <Key>osfUp	EditTableEnterCell(Up)
	EditTableClearAllSelections()

None <Key>osfDown	EditTableEnterCell(Down)
	EditTableClearAllSelections()

None <Key>Return	EditTableEnterCell(Down)
	EditTableClearAllSelections()

~Ctrl ~Shift ~Meta ~Alt <Btn1Down>	EditTableEnterCell(Pointer)
	EditTableStartSelect(single)

<Btn1Up>	EditTableEndSelect()
<Btn2Down>	MotifDragStart()
<Key>osfActivate	EditTableConfirmEdit()
~Ctrl ~Meta ~Alt <Key>	EditTableEditCell()
Ctrl ~Meta ~Alt <Key> Tab	NextTabGroup()
Shift Ctrl ~Meta ~Alt <Key> Tab	PreviousTabGroup()

Actions With No Default Translations

Some EditTable actions have no default translations specified. In particular, the default translation table contains no definitions for any cut, copy and paste operations on rows and columns nor does it contain definitions for any of the annotation actions.

Changing the Default Translation Table

The EditTable default translations table can be modified using the functions XtAugmentTranslations or XtOverrideTranslations or it can be replaced by specifying your own translation table using resource **XmNtranslations**. For example, you may wish to add support for multiple selections (add a translation for action EditTableStartSelect(multiple)) or add translations for cut and paste operations.

Specifying Translations for Annotation Actions

The annotation actions (AnnotationEdit, AnnotationStartSelect, AnnotationExtendSelect, AnnotationEndSelect, AnnotationStartDrag, AnnotationExtendDrag and AnnotationEndDrag) have no default translations. When the EditTable widget is not a child of an INT Scroll widget, you must merge (using XtAugmentTranslations or XtOverrideTranslations) the application defined translations for the annotation actions with the existing default translation table. When an EditTable widget is a child of an INT Scroll widget, then the translations for column annotation actions must be registered by specifying a translation table for those actions as the value of the EditTable resource **XmNcolumnAnnotationTranslations**. Similarly, the translations for row annotation actions must be registered by specifying a translation table for those actions as the value of the EditTable resource **XmNrowAnnotationTranslations**. Note that it is possible to use the same translation table for the row and column annotation translations.

Text Actions

Actions TextConfirmEdit and TextAbandonEdit should not be registered directly to the EditTable widget but rather to the internal text widget(s) created by the EditTable widget. The ID of the text widgets created by an EditTable can be obtained using convenience function XintEditTableGetTextChild.

EditTable Callbacks

The following callbacks are defined by an EditTable widget.

Name	Structure	Reason
XmNcellAttributesCall-back	XintEditTableCellAt-tributesCallbackStruct	XintCR_CELL_RESIZE XintCR_COLUMN_RESIZE XintCR_DISPLAY_CELL XintCR_ROW_RESIZE
XmNcellWidgetDis-playCallback	XintEditTableCellWid-getCallbackStruct	XintCR_QUERY_CELL_ WIDGET XintCR_UPDATE_CELL_ WIDGET
XmNcheckEditMode-Callback	XintEditTableCheck-EditModeCallbackStruct	XintCR_CHECK_EDIT_MODE
XmNcolumnCallback	XintEditTableOperation-CallbackStruct	XintCR_DELETE_COLUMN XintCR_INSERT_COLUMN XintCR_SELECT_COLUMN
XmNdoubleClickCall-back	XintEditTableDouble-ClickCallbackStruct	XintCR_DOUBLE_CLICK
XmNdragCallback	XintEditTableDrag-CallbackStruct	XintCR_CELL_MOVE XintCR_CELL_RESIZE XintCR_CELL_COPY XintCR_COLUMN_MOVE XintCR_COLUMN_RESIZE XintCR_COLUMN_COPY XintCR_ROW_MOVE XintCR_ROW_RESIZE XintCR_ROW_COPY
XmNdragDropCallback	XintEditTableDragDrop-CallbackStruct	XintCR_DRAG XintCR_DROP
XmNeditAnnotation-Callback	XintEditTableEditAnno-tationCallbackStruct	XintCR_EDIT_COLUMN_ ANNOTATION XintCR_EDIT_ROW_ ANNOTATION
XmNformatCellCall-back	XintEditTableFormat-CellCallbackStruct	XintCR_DISPLAY_CELL XintCR_CALCULATE_CELL_ WIDTH XintCR_CALCLULATE_CELL_ HEIGHT

Name (continued)	**Structure**	**Reason**
XmNformatColumnAnnotationCallback	XintEditTableFormatAnnotationCallbackStruct	XintCR_GET_COLUMN_ANNOTATION
XmNformatRowAnnotationCallback	XintEditTableFormatAnnotationCallbackStruct	XintCR_GET_ROW_ANNOTATION
XmNrowCallback	XintEditTableOperationCallbackStruct	XintCR_DELETE_ROW XintCR_INSERT_ROW XintCR_SELECT_ROW
XmNselectCellCallback	XintEditTablOperationCallbackStruct	XintCR_SELECT_CELL
XmNtraverseCellCallback	XintEditTableTraverseCellCallbackStruct	XintCR_TRAVERSE_CELL_DOWN XintCR_TRAVERSE_CELL_LEFT XintCR_TRAVERSE_CELL_RIGHT XintCR_TRAVERSE_CELL_UP XintCR_TRAVERSE_CELL_POINTER XintCR_TRAVERSE_CELL_FOCUS_IN
XmNvalidateValueCallback	XintEditTableValidateValueCallbackStruct	XintCR_VALIDATE_VALUE

XintEditTableCellAttributesCallbackStruct

The following table lists the members of the callback structure XintEditTableCellAttributesCallbackStruct used by the EditTable widget for callback XmNcellAttributesCallback.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the event that triggered the callback.
int	row	Indicates the row number of the selected cell.
int	column	Indicates the column number of the selected cell.
int	alignment	Text alignment constant in the cell.
Pixel	background	Cell background color
Pixel	foreground	Cell foreground color.
int	font_table_index	Index to the font table. If there is no font table added to EditTable, the index will have no effect. To add font table, refer to“*XmNfontTable*” on page 99.

The *reason* will be set to one of the following defined constants

Defined Constant	Description
XintCR_CELL_RESIZE	Indicates that the end user has performed a cell resize operation.
XintCR_COLUMN_RESIZE	Indicates that the end user has performed a column resize operation.
XintCR_ROW_RESIZE	Indicates that the end user has performed a row resize operation.
XintCR_DISPLAY_CELL	The cell is about to be displayed on screen. Allows the programmer to set the cell attributes before the cell is displayed.

XintEditTableCellWidgetCallbackStruct

The following ordered table lists the members of the callback structure XintEditTableCellWidgetCallbackStruct used by the EditTable widget for callback XmNcellWidgetDisplayCallback.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the event that triggered the callback.
int	row	Indicates the row number of the selected cell.
int	column	Indicates the column number of the selected cell.
Widget	widget	ID of the Widget In a Cell.
XintCellWidget-Resources *	resources	Points to the resources that may be set automatically on the widget.
Boolean	doit	Indicates whether or not the resources defined above will be set on the widget before it is drawn. This value is initialized to the setting of resource **XmNcellWid-getSetResources**.
Boolean	displayit	Indicates whether or not the widget in this cell should be displayed.

The *reason* member will be set to one of the following defined constants.

Defined Constant	Description
XintCR_QUERY_CELL_WIDGET	The column width or row height was set to 0 (for auto calculate). Allows the programmer to set the resources on the widget so that proper cell size can be calculated.
XintCR_UPDATE_CELL_WIDGET	The widget is about to be drawn on the virtual screen. Allows the programmer to set the resources on this widget before it is drawn.

Structure XintCellWidgetResources is defined as follows.

```
    typedef struct {

        String     string;
        String     font;
        Pixel      background;
        Pixel      foreground;
        int        alignment;
        int        sensitive;

    } XintCellWidgetResources;
```

string	Format of the display string for the cell value.
font	X window font name for the display string.
background	Cell background color.
foreground	Cell foreground color.
alignment	Motif alignment constant for the cell.
sensitive	Cell read-only.

If the **XmNcellWidgetSetResources** constraint resource is set to True, primitive widgets, such as labels and buttons, do not require a callback. Their resources will be set correctly.

In other cases, such as inserting toggle buttons or option menus, a callback is required. Inside this callback the application should extract the current cell contents, then set the widget state and graphic resources accordingly. When the state of a widget in a cell is changed (for example, an option menu is changed to a new selection), the application should update the cell contents. A cell is identified by its location. If a widget is used for more than one cell, the cell location can be obtained by using function XintEditTableGetCellPointerPosition.

The *doit* member will be set to the value of the constraint resource **XmNcellWidgetSetResources** before the callback is called.

XintEditTableCheckEditModeCallbackStruct

The following ordered table lists the members of the callback structure XintEditTableCheckEditModeCallbackStruct used by the EditTable widget for the check edit mode callback.

Data Type	Member	Description
int	reason	Set to XintCR_CHECK_EDIT_MODE.
XEvent *	event	Points to the event that triggered the callback.
int	column	Indicates the column containing the cell to be edited.
int	row	Indicates the row containing the cell to be edited.
int	column_edit_mode	Indicates the edit mode of column containing the cell to be edited.
Boolean	doit	Indicates whether the cell is editable (used only when the column containing the cell is editable).

The *column_edit_mode* member will be set to one of the following:

Defined Constant	Description
XintCOLUMN_EDITABLE	Indicates that the column containing the cell is editable.
XintCOLUMN_NON_EDITABLE	Indicates that the column containing the cell is not editable.

The *doit* flag is used when *column_edit_mode* has the value XintCOLUMN_EDITABLE. The associated action routine sets this flag to True before the callback list is invoked. A callback procedure should set this flag to False if the cell (in an editable column) is not enabled for editing. When *column_edit_mode* has the value XintCOLUMN_NON_EDITABLE the cell selected for editing always is non-editable.

XintEditTableDoubleClickCallbackStruct

The following ordered table lists the members of the callback structure XintEditTableDoubleClickCallbackStruct used by the EditTable widget for callback XmNdoubleClickCallback.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the event that triggered the callback.
int	column	Indicates the column number where the button was pressed.
int	row	Indicates the row number where the button was pressed.

XintEditTableDragCallbackStruct

The following ordered table lists the members of the callback structure XintEditTableDragCallbackStruct used by the EditTable widget for callback XmNdragCallback.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the event that triggered the callback.
int	src_column	Source column index (set if a column or cell operation).
int	src_row	Source row index (set if a row or cell operation).
int	current_column	Destination column index (set if a column or cell operation).
int	current_row	Destination row index (set if a row or cell operation).
Boolean	doit	Set to False to cancel move, resize or copy operation.

The *reason* member will be set to one of the following defined constants:

Defined Constant	**Description**
XintCR_CELL_MOVE	Indicates that the end user has performed a cell move operation (moving the content of a cell into another).
XintCR_CELL_RESIZE	Indicates that the end user has performed a cell resize operation.
XintCR_CELL_COPY	Indicates that the end user has performed a cell copy operation.
XintCR_COLUMN_FREEZE	Indicates that the end user has performed a column move operation into a frozen column.
XintCR_COLUMN_MOVE	Indicates that the end user has performed a column move operation.
XintCR_COLMUN_RELEASE	Indicates that the end user has performed a column move operation from a frozen column.
XintCR_COLUMN_RESIZE	Indicates that the end user has performed a column resize operation.
XintCR_COLUMN_COPY	Indicates that the end user has performed a column copy operation.
XintCR_ROW_FREEZE	Indicates that the end user has performed a row move operation into a frozen row.
XintCR_ROW_MOVE	Indicates that the end user has performed a row move operation.
XintCR_ROW_RELEASE	Indicates that the end user has performed a row move operation from a frozen row.
XintCR_ROW_RESIZE	Indicates that the end user has performed a row resize operation.
XintCR_ROW_COPY	Indicates that the end user has performed a row copy operation.

XintEditTableDragDropCallbackStruct

Callback XmNdragDropCallback is defined by the EditObject class to support Motif drag and drop operations. The EditTable class redefines a new callback structure, XintEditTableDragDropCallbackStruct, that is passed when this callback is invoked for an EditTable widget.

Data Type	**Member**	**Description**
int	reason	Indicates why the callback was invoked.

Data Type	Member	Description
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 where drag started or drop occurred.
int	row_start	On drag, starting index of the row that is dragged. On drop, row index where the cursor is located and where the cells will be dropped. You can modify this field on a drag or drop operation.
int	row_count	On drag, number of rows being dragged. On drop, this field is not used. You can modify this field on a drag operation.
int	column_start	On drag, starting index of the column that is dragged. On drop, column index where the cursor is located and where the cells will be dropped. You can modify this field on a drag or drop operation.
int	column_count	On drag, number of columns being dragged. On drop, this field is not used. You can modify this field on a drag operation.
Boolean	doit	Set to False to cancel the drag or drop operation.

XintEditTableEditAnnotationCallbackStruct

The following ordered table lists the members of the callback structure XintEditTableEditAnnotationCallbackStruct used by the EditTable widget for the edit annotation callback.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the event that triggered the callback.
int	row	Indicates the row containing the annotation to be edited.

Data Type	Member	Description
int	column	Indicates the column containing the annotation to be edited.
Boolean	if_auto	Indicates whether the annotation selected for editing is editable.

The *reason* member will be set to one of the following defined constants:

Defined Constant	Description
XintCR_EDIT_COLUMN_ ANNOTATION	Indicates that the end user has selected a column annotation for editing.
XintCR_EDIT_ROW_ ANNOTATION	Indicates that the end user has selected a row annotation for editing.

When the *if_auto* flag is True, it indicates that the annotation selected for editing was generated automatically by the EditTable widget. If the annotation was generated by the EditTable widget, then it cannot be edited by the application or the end user.

XintEditTableFormatAnnotationCallbackStruct

The following table lists the members of the callback structure XintEditTableFormatAnnotationCallbackStruct used by the EditTable widget for the format column annotation callback and format row annotation callback.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Pointer to the XEvent that triggered the call-back.
int	row	The row index. If the callback is to format column annotation, the row index will always be 1.
int	column	The column index. If the callback is to format row annotation, the column index will always be 1.
char *	annotation_string	The annotation string to be displayed in the column/row. EditTable will make a copy of the string.

The *reason* member will be set to one of the following defined constants:

Defined Constant	Description
XintCR_GET_COLUMN_ANNOTATION	Indicates that EditTable is about to display the column annotation
XintCR_GET_ROW_ANNOTATION	Indicates that EditTable is about to display the row annotation.

XintEditTableFormatCellCallbackStruct

The following table lists callback XintEditTableFormatCellCallbackStruct members used by the EditTable widget for allowing the application to define its own mechanism to format a cell before display:

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Pointer to the XEvent that triggered the call-back.
int	row	The index of the row of the cell to be formatted.
int	column	The index of the column of the cell to be for-matted.
int	column_data_type	The column data type for that cell.
XintCellValue	cell_value	A Union containing the cell value.
char *	display_string	Returns the string to display in that cell.
Boolean	to_be_freed	If Set to True, string *display_string* will be freed by the widget after it has been used. This member is initialized to False, which corresponds to the case where *display_string* is allocated statically by the application.

The *reason* member will be set to one of the following defined constants:

Defined Constant	Description
XintCR_DISPLAY_CELL	Formatting required to display cell contents.
XintCR_CALCULATE_CELL_WIDTH	Formatting required to calculate cell width.
XintCR_CALCULATE_CELL_HEIGHT	Formatting required to calculate cell height.

Union XintCellValue is defined as follows

```
typedef union {
    short short_value;
    int integer_value;
    long long_value;
    float float_value;
    double double_value;
    char *string_value;
    XtPointer pointer_value;
} XintCellValue;
```

XintEditTableOperationCallbackStruct

The following ordered table lists the members of the callback structure XintEditTableOperationCallbackStruct used by the EditTable widget for the column, row, and select cell callbacks.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Pointer to the XEvent that triggered the callback.
int	column_start	Indicates the first column involved in the operation.
int	row_start	Indicates the first row involved in the operation.
int	column_count	Indicates the number of columns involved in the operation.
int	row_count	Indicates the number of rows involved in the operation.
Boolean	doit	Indicates whether the associated action should be executed (used only for delete row and for delete column operations).

The *reason* member will be set to one of the following defined constants:

Defined Constant	Description
XintCR_DELETE_COLUMN	Indicates that a column will be deleted if the *doit* flag is set to True.
XintCR_INSERT_COLUMN	Indicates that one or more columns will be inserted.
XintCR_SELECT_COLUMN	Indicates that one or more columns have been selected.
XintCR_DELETE_ROW	Indicates that a row will be deleted if the *doit* flag is set to True.
XintCR_INSERT_ROW	Indicates that one or more rows will be inserted.
XintCR_SELECT_ROW	Indicates that one or more rows have been selected.
XintCR_SELECT_CELL	Indicates that one or more cells have been selected.

The *doit* flag is used for the row delete and column delete operations. The associated action routine sets this flag to True before the callback list is invoked. A callback procedure should set this flag to False if the row or column should not be deleted by the action routine.

XintEditTableTraverseCellCallbackStruct

The following ordered table lists the members of the callback structure XintEditTableTraverseCellCallbackStruct used by the EditTable widget for the traverse cell callback.

Data Type	Member	Description
int	reason	Indicates why the callback was invoked.
XEvent *	event	Points to the event that triggered the callback.
int	row	Indicates the row number of the cell where the cell pointer is currently located.
int	column	Indicates the column number of the cell where the cell pointer is currently located.
int	next_row	Indicates the row number of the cell that the cell pointer will enter.
int	next_column	Indicates the column number of the cell that the cell pointer will enter.
int	number_of_columns	Indicates the number of columns in the table.
int	number_of_rows	Indicates the number of rows in the table.

The *reason* member will be set to one of the following defined constants:

Defined Constant	Description
XintCR_TRAVERSE_CELL_DOWN	Indicates cell entered from cell above.
XintCR_TRAVERSE_CELL_LEFT	Indicates that cell entered from cell on left.
XintCR_TRAVERSE_CELL_RIGHT	Indicates cell entered from cell on the right.
XintCR_TRAVERSE_CELL_UP	Indicates cell entered from cell below.
XintCR_TRAVERSE_CELL_POINTER	Indicates cell entered with pointer selection.
XintCR_TRAVERSE_CELL_FOCUS_IN	Indicates cell entered because focus was brought back to the table.

XintEditTableValidateValueCallbackStruct

The following ordered table lists the members of the callback structure XintEditTable-ValidateValueCallbackStruct used by the EditTable widget for the validate value callback.

Data Type	**Member**	**Description**
int	reason	Set to XintCR_VALIDATE_VALUE.
XEvent *	event	Points to the event that triggered the callback.
int	row	Indicates the row number of the cell whose value needs to be validated.
int	column	Indicates the column number of the cell whose value needs to be validated.
int	column_data_type	Indicates the data type of the cell whose value needs to be validated.
char *	format	A character string containing the (C language) format descriptor for the cell whose value needs to be validated.
char *	old_value_string	A character string containing the value of the cell before it was changed.
char *	new_value_string	A character string containing the value of the cell after it was changed.
Boolean	to_be_freed	Set to True if you want EditTable to free the string specified in *new_value_string* after it has been used.
XintCellValue	cell_value	Pointer to the new cell value.
Boolean	doit	Indicates whether the cell's value should be changed to the new value.

The associated action routine sets the *doit* flag to True before the callback list is invoked. To cancel the editing and restore the old cell value, simply set the *doit* flag to False. To overwrite the user entry and specify a new cell value you have two options. The first solution is to replace member *new_value_string* with your own string. In this case, if you have allocated the new value string dynamically, you may want to set *to_be_freed* to True so that it is deallocated by the table after it is no longer needed. The second solution is to replace directly member *cell_value* with the new cell value.

Cell Attributes Callback

This callback is called by the cell painting routine in EditTable. The callback structure XintEditTableCellAttributesCallbackStruct contains the display attributes such as text alignment, background, foreground, and the font table index. The font table index will have affect if and only if a valid font table is added to the EditTable (refer to *"XmNfontTable"* on page 99 for more information). Using this callback, the programmer can set these cell attributes dynamically. In addition, because the callback is dynamic, memory for these attributes does not need to be allocated by EditTable, and thus improves the overall performance.

Check Edit Mode Callback

This type of callback is called by the EditTableEditCell action routine when a cell has been selected for editing. The callback structure XintEditTableCheckEditModeCallbackStruct will contain the edit mode of the column containing the cell. This callback should determine whether the cell can be edited by the user and set the *doit* flag to False if the cell should not be edited. If the cell is in a non-editable column, the setting of the *doit* flag is ignored by the action routine and the cell will not be editable. This callback can be used by the application to display a message so that the user will know that the selected cell cannot be edited. Another application for this callback is implementing non-editable cells in an editable column.

Column Callback

This type of callback is called by the EditTableEndSelect, AnnotationEndSelect, EditTableDeleteColumns, and EditTableInsertColumns action routines. The callback is called after the action occurs, except for EditTableDeleteColumns where it is called before. The callback structure XintEditTableOperationCallbackStruct will contain the indices of the column(s) being operated on. The callback structure also contains the *doit* flag which is initialized to True by the EditTableDeleteColumns action procedure. If you do not want to delete the selected column(s), then set the *doit* flag to False.

Double-click Callback

This type of callback is called when the user double clicks on a cell. No action is connected to this callback. The callback structure is XintEditTableDoubleClickCallback. The code of the Button used for the double click can be found in the XEvent structure. The callback structure XintEditTableDragCallbackStruct will contain the indices of the source and destination cells, rows or columns. The callback structure also contains the *doit* flag which is initialized to True. If you do not want to delete the selected column(s), then set the *doit* flag to False. A number of resources, **XmNdragForeground**, **XmNdragCursorType**, XmNdragGridLineStyle and **XmNdrawShowCellContent** is provided to customize the appearance of the cell, row or column being dragged.

Drag Callback

This type of callback is called by the AnnotationEndDrag and EditTableEndDrag action routines before a Copy, Move or Resize operation action occurs on a cell, row or column. This type of callback only allows the manipulation of cells, rows or columns locally. EditObject callback XmNdragDropCallback provides full Motif Drag and Drop functionality. Refer to *Chapter 1—Introduction* and *Chapter 3—EditObject Widget Class* for more information on Motif Drag and Drop support.

Edit Annotation Callback

This type of callback is called by the AnnotationEdit action routine so that the callback list can implement row annotation editing operations and column annotation editing operations. The callback structure XintEditTableEditAnnotationCallbackStruct will indicate whether the user has selected a row annotation or a column annotation and it will specify the location of the annotation selected. The callback structure will also indicate whether or not the annotation selected was generated automatically by the EditTable widget. If the annotation was generated by the widget, then it cannot be edited, and the application should display a message to that effect.

Format Cell Callback

Called each time the EditTable needs to display a cell when the column format specifier for that cell is set to NULL or the data type is XintTYPE_POINTER. Allows the application to provide custom formats or to format complex data types that are stored in the pointer format.

Format Column Annotation Callback

This type of callback, when added to Xt, is called when automatic annotation is set to False. It allows the programmer to change the column annotation dynamically. Since the callback is dynamic, i.e. the programmer can provide column annotation strings at runtime, EditTable is thus freed from allocating memory to store these annotation strings. Consequently, the performance of EditTable is improved. Note that EditTable will make a copy of the annotation string passed in, and thus, it is the programmer's responsibility to free the string passed to EditTable.

Format Row Annotation Callback

This type of callback, when added to Xt, is called when automatic annotation is set to False. It allows the programmer to change the row annotation dynamically. Since the callback is dynamic, i.e. the programmer can provide row annotation strings at run-time, EditTable is thus freed from allocating memory to store these annotation strings. Consequently, the performance of EditTable is improved -- imagine an EditTable that has tens of thousands of rows. Note that EditTable will make a copy of the annotation string passed in, and thus, it is the programmer's responsibility to free the string passed to EditTable

Row Callback

This type of callback is called by the EditTableEndSelect, AnnotationEndSelect, EditTableCopyRow, EditTableDeleteRows, and EditTableInsertRows action routines. The callback is called after the action occurs, except for EditTableDeleteRows where it is called after. The callback structure XintEditTableOperationCallbackStruct will contain the indices of the row(s) being operated on. The callback structure also contains the *doit* flag which is initialized to True by the EditTableDeleteRows action procedure. If you do not want to delete the selected row(s), then set the *doit* flag to False.

Select Cell Callback

Called by the EditTableEndSelect action routine after the user has selected one or more cells in the table. XintEditTableOperationCallbackStruct will contain the indices of the cell(s) selected.

Traverse Cell Callback

This type of callback is called by the EditTable widget whenever the user moves the cell pointer to a new cell using the keyboard. The action routine EditTableEnterCell calls this type of callback so that the callback procedures can determine whether the cell should be entered. The callback structure XintEditTableTraverseCellCallbackStruct will contain the indices of the cell to which the user wants to traverse. The callback procedure can change the indices of the cell to be traversed to so that the action procedure moves the cursor to a different cell. Using this mechanism, the application can control how the user traverses through a table, skipping over irrelevant cells.

Validate Value Callback

This type of callback is called by the EditTableConfirmEdit action procedure or when the user terminates editing by leaving a cell. It gives the application programmer the ability to check the new cell content before it is stored into the table. The callback structure XintEditTableValidateValueCallbackStruct will contain the location of the cell being edited as well as the old and new values of the cell. The callback structure also contains the *doit* flag which is initialized to True by the EditTableConfirmEdit action procedure. If you do not want the EditTableConfirmEdit action procedure to change the value, then set the *doit* flag to False. Alternatively, you can leave the *doit* flag set to True and substitute a different value for the new value so that the action procedure will change the cell's value to the application specified value.

EditTable Functions

The following convenience functions are defined for creating and manipulating an EditTable widget:

Function Name	Description
XintCreateEditTable	Creates an unmanaged EditTable widget.
XintEditTableAbandonEdit	To abandon editing of the current cell and restore the original value.
XintEditTableAddLocalCallback	Adds a callback procedure that is local to a cell, row or column.
XintEditTableAddToSelection	Adds a selection to the table.
XintEditTableAssociateData	Associates a Data Object with an EditTable.
XintEditTableCellFlash	Makes a cell flash at a specified time interval over a specified period of time.
XintEditTableCellSpanGetRange	Returns the span for the specified cell.
XintEditTableCellSpanSetRange	Sets the span on a range of cells.

Function Name (continued)	**Description**
XintEditTableChangeColumnVisibility	Sets a range of columns to be visible or non visible.
XintEditTableChangeRowVisibility	Sets a range of rows to be visible or non visible.
XintEditTableClearAllSelections	Clears all the selections of a table.
XintEditTableClearCells	Clears the content of the specified cells.
XintEditTableClearSelectionByNumber	Removes a specific selection.
XintEditTableColumnScroll	Controls scrolling of columns.
XintEditTableConfirmEdit	To confirm the editing of the current cell and invoke callback XmNvaliidateValueCallback.
XintEditTableCopyColumn	Copies a single column to the EditTable widget’s clipboard.
XintEditTableCopyRows	Copies a range of rows to the EditTable widget’s clipboard.
XintEditTableDefineColumnFormat	Defines the format of the cells in a specified column.
XintEditTableDeleteColumns	Copies a range of columns to the EditTable widget’s clipboard and removes the columns from the table.
XintEditTableDeleteRows	Copies a range of rows to the EditTable widget’s clipboard and removes the columns from the table.
XintEditTableFillCell	Sets the value of a specified cell.
XintEditTableFillCellNoUpdate	Sets the value of a specified cell without updating the display.
XintEditTableFillColumnAnnotation	Sets the column annotation for a specified column.
XintEditTableFillColumnData	Sets the values of every cell in a specified column using an array of data.
XintEditTableFreezeColumn	Moves a specified column to the left side of the table display so that it always stays visible when scrolling horizontally.
XintEditTableFreezeRow	Moves a specified row to the top of the table display so that it always stays visible when scrolling vertically.

Function Name (continued)	**Description**
XintEditTableFreezeUpdate	To enable or disable geometry updates and redrawing on the table.
XintEditTableGetCellBackground	Returns the background pixel color of a specified cell.
XintEditTableGetCellData	Returns the address of a copy of the value in a specified cell.
XintEditTableGetCellFont	Returns the font index used by a cell.
XintEditTableGetCellForeground	Returns the foreground pixel color of a specified cell.
XintEditTableGetCellGeometry	Returns the position and size of a cell inside the EditTable widget.
XintEditTableGetCellHeight	Returns the height of the specified cell.
XintEditTableGetCellPixmap	Returns the pixmap ID of a cell.
XintEditTableGetCellPointerPosition	Returns the location of the cell pointer.
XintEditTableGetCellWidget	Returns the cell widget for the specified location.
XintEditTableGetCellWidth	Returns the width of the specified cell.
XintEditTableGetColumnAnnotionAlignment	Returns a pointer to the column's annotation alignment.
XintEditTableGetColumnAttributes	Returns the attributes of a specified column.
XintEditTableGetColumnData	Returns the address of a copy of an array containing the values in the cells of a specified column.
XintEditTableGetColumnUserData	Returns a pointer to a column’s user data.
XintEditTableGetFrozenColumns	Returns the address of an integer array containing the column numbers of the frozen columns.
XintEditTableGetFrozenRows	Returns the address of an integer array containing the row numbers of the frozen rows.
XintEditTableGetHiddenColumns	Returns the columns that are not visible.
XintEditTableGetHiddenRows	Returns the rows that are not visible.
XintEditTableGetSelectedCells	Returns information regarding the selected cells.
XintEditTableGetSelectedColumns	Returns information regarding the selected columns.

Function Name (continued)	Description
XintEditTableGetSelectedRows	Returns information regarding the selected rows.
XintEditTableGetSelectionByNumber	Extracts a specific selection from the table.
XintEditTableGetSelectionCount	Returns the number of selections in the table.
XintEditTableGetSubtable	Returns the ID of a specific subtable.
XintEditTableGetTextChild	Returns the ID of the text widget used to edit a cell content.
XintEditTableGetVisibleArea	Passes back the visible area of the table.
XintEditTableInsertColumns	Inserts a specified number of annotated columns before a specified column.
XintEditTableInsertRows	Inserts a specified number of nonannotated rows before a specified row.
XintEditTableIsCellDefined	Tells if a cell is defined or not.
XintEditTableIsColumnFrozen	Tells if a column is frozen or not.
XintEditTableIsRowFrozen	Tells if a row is frozen or not.
XintEditTableIsColumnHidden	Tells if a column is hidden or not.
XintEditTableIsRowHidden	Tells if a row is hidden or not.
XintEditTableOutputAscii	Creates an ASCII formatted file based on a specified range of cells in a table.
XintEditTableOutputPostscript	Creates a PostScript file based on a specified range of cells in the table.
XintEditTableOutputSimplePS	Output of PostScript file carries only the courier-10 font and ignores all but left and middle alignment.
XintEditTableOutputSimplePS2	Output of PostScript file uses fonts and alignments set in EditTable.
XintEditTableOutputSylkFile	Creates a SYLK formatted file based on a specified range of cells in the table.
XintEditTablePSReportStyle	Creates a Postscript formatted report based on a specified range of cells in the table.
XintEditTablePasteColumns	Replaces the values in a specified range of columns by a range of columns contained on the EditTable widget's clipboard.
XintEditTablePasteRows	Replaces the values in a specified range of rows by a range of rows contained on the EditTable widget's clipboard.

Function Name (continued)	Description
XintEditTableReadAscii	Reads an ASCII file and loads the data.
XintEditTableReleaseColumn	Changes a frozen column’s state so that it is no longer frozen.
XintEditTableReleaseRow	Changes a frozen row’s state so that it is no longer frozen.
XintEditTableReorderColumns	Changes the ordering of a range of columns.
XintEditTableReorderRows	Changes the ordering of a range of rows.
XintEditTableRemoveAllLocalCallbacks	Removes all local callback procedures attached to a specified callback.
XintEditTableRemoveLocalCallback	Removes a local callback procedure attached to a cell, row or column.
XintEditTableRowScroll	Allows the application to control scrolling or rows.
XintEditTableSetCellBackground	Sets the background color of a specified set of cells.
XintEditTableSetCellDisplayAttributes	Sets the background, foreground and data of a specified cell.
XintEditTableSetCellFont	Sets the font of a range of cells.
XintEditTableSetCellForeground	Sets the foreground color of a specified set of cells.
XintEditTableSetCellHeight	Sets the height of a range of cells.
XintEditTableSetCellPixmap	Sets the background pixmap of a specified set of cells.
XintEditTableSetCellPixmapList	Sets the background pixmap of a set of cells from a pixmap list.
XintEditTableSetCellPointerPosition	Specifies the cell location at which to position the cell pointer.
XintEditTableSetCellWidth	Sets the width of a range of cells.
XintEditTableSetColumnAnnotationAlignment	Specifies the annotation alignment starting at a specific column and using that alignment for a specified number of columns.
XintEditTableSetColumnFont	Sets the font of a specified column.
XintEditTableSetColumnUserData	Assigns user data to a specific column.
XintEditTableSetListBehavior	Simulates the XmList behavior.
XintEditTableSetRowFont	Sets the font of a specified row.

Function Name (continued)	**Description**
XintEditTableSetSelection	Causes a specified range of cells, rows or columns to be selected.
XintEditTableSortByColumn	This functions allows the user to sort rows in the specified column.
XintEditTableUndeleteColumns	Causes the last column delete operation to be reversed so that the columns are not deleted.
XintEditTableUndeleteRows	Causes the last row delete operation to be reversed so that the rows are not deleted.
XintEditTableUnfreeze	Unfreezes the table.
XintEditTableUpdateDataDisplay	Causes the EditTable widget to update the display of a cell, row or column.

XintCreateEditTable

XintCreateEditTable creates an unmanaged EditTable widget.

```
    Widget   XintCreateEditTable (...)
```

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

XintEditTableAbandonEdit

This function cancels the editing of the current cell and restores the original cell value. This function is called by action TextAbandonEdit.

```
    void XintEditTableAbandonEdit (Widget widget)
```

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

XintEditTableAddLocalCallback

This function adds a callback procedure that is local to a cell, row or column.

```
    Boolean XintEditTableAddLocalCallback (...)
```

Widget	widget	EditTable widget ID.
int	column	Column number (specify 0 to register the call-back for all columns).
int	row	Row number (specify 0 to register the call-back for all rows).
char *	callback_name	Callback name (must be a cell, row or column callback defined by EditTable).
XtCallbackProc	callback	Callback procedure.
XtPointer	client_data	Client data for callback procedure.
int	calling_sequence	Specifies if local callback is called instead of, before or after normal callbacks.

The *calling_sequence* argument must be one of the following defined constants:

Defined Constant	Description
XintLOCAL_CALLBACK_EXCLUSIVE	Local callback is called instead of normal callback.
XintLOCAL_CALLBACK_BEFORE	Local callback is called before.
XintLOCAL_CALLBACK_AFTER	Local callback is called after.

XintEditTableAddToSelection

This function adds a selection to the current selection. The selection can be specified as a set of cells, columns or rows.

```
    Boolean XintEditTableAddToSelection (...)
```

Widget	widget	EditTable widget ID.
int	col_start	Starting column number.
int	num_cols	Number of columns.
int	row_start	Starting row number.
int	num_rows	Number of rows.
int	selection_mode	Specifies if selection is for a set of cells, rows or columns.

The *selection_mode* argument must be one of the following:

Defined Constant	Description
XintSELECT_CELL	Selection is a block of cells.
XintSELECT_ROW	Selection is a set of rows.
XintSELECT_COLUMN	Selection is a set of columns.

XintEditTableAssociateData

This function is only available when the EditTable widget is used in conjunction with ChartObject. If you are not using ChartObject, you may want to ignore this function. XintEditTableAssociateData provides a mechanism to associate a data object (usually a DataGroup object) with an EditTable widget. This function handles a DataGroup object as follows:

1. If a DataLabel object oriented along the X direction is found, it is used to provide column annotation.
2. If a DataLabel object oriented along the Y direction is found, it is used to provided row annotation.
3. Each DataSampled found is used to fill a column.

If one of the objects in the DataGroup has a range set, it is ignored. Only arguments *col_start* and *row_start* are used to position the data inside the table.

```
Boolean XintEditTableAssociateData (...)
```

Widget	widget	EditTable widget ID.
Object	data	Data object to associate with the table.
int	col_start	First column where to position the data.
int	row_start	First row where to position the data.
Boolean	linked	If True, the table is linked with the data object (i.e.: changes in both the table and the data object are propagated to the other). If False, no connection is established (data object is just used to fill the table).

XintEditTableCellFlash

This function will make the specified cell flash at a specified rate (in unit milliseconds), over a specified period of time (in unit seconds).

```
Boolean XintEditTableCellFlash (...)
```

Widget	widget	EditTable widget ID.
int	column	Column location of the cell.
int	row	Row location of the cell.
long	interval	Flashing rate in unit milliseconds.
long	duration	Flashing time period in unit seconds
Pixel	color	Specifies the flashing color

The function returns False if the widget is not an EditTable widget or if the specified cell is not in the visible table area.

XintEditTableCellSpanGetRange

This function returns the cell span factor for the specified cell.

```
void XintEditTableCellSpanGetRange (...)
```

Widget	table	EditTable widget ID.
int	row	Row location of the cell.
int	column	Column location of the cell.
XintCellSpanFactor *	factor	Pointer to the structure containing the span data.

XintEditTableCellSpanSetRange

This function will set the cell span factor for all of the cells within the designated row/column range.

```
void XintEditTableCellSpanSetRange (...)
```

Widget	table	EditTable widget ID.
int	column	First column in the range of columns to be selected.
int	row	First row in the range of rows to be selected.
int	columns	Number of columns in the range. 0 = all columns following the starting column.
int	rows	Number of rows in the range. 0 = all rows following the starting row.
XintCellSpanFactor *	factor	Pointer to the structure containing the span data.

XintEditTableChangeColumnVisibility

This function turns on or off the visibility of a range of columns.

```
Boolean XintEditTableChangeColumnVisibility (...)
```

Widget	widget	EditTable widget ID.
int	col_start	First column to update.
int	col_end	Last column to update.
Boolean	visible	True if column is to be made visible; False if column is to be made invisible (i.e., hidden).

XintEditTableChangeRowVisibility

This function turns on or off the visibility of a range of rows.

```
    Boolean XintEditTableChangeRowVisibility (...)
```

Widget	widget	EditTable widget ID.
int	row_start	First row to update.
int	row_end	Last row to update.
Boolean	visible	True if row is to be made visible; False if row is to be made invisible (i.e., hidden).

XintEditTableClearAllSelections

This functions clears all existing selections.

```
    Boolean XintEditTableClearAllSelections (Widget widget)
```

where *widget* is the ID of an EditTable widget. This functions returns True if there was a selection prior to the call.

XintEditTableClearCells

This function clears the content of the specified cells.

```
    Boolean XintEditTableClearCells (...)
```

Widget	widget	EditTable widget ID.
int	col_start	Starting column number.
int	col_end	Ending column number.
int	row_start	Starting row number.
int	row_end	Ending row number.

XintEditTableClearSelectionByNumber

This function clears a specific selection. False is returned if *selection_number* is invalid.

```
    Boolean XintEditTableClearSelectionByNumber (...)
```

Widget	widget	EditTable widget ID.
int	selection_number	Number of the selection to clear.

XintEditTableColumnScroll

This function causes an EditTable widget to scroll the displayed table.

```
void XintEditTableColumnScroll (...)
```

Widget	widget	EditTable widget ID.
int	operation	Specifies the direction and amount of scrolling.

The *operation* argument must be one of the following:

Defined Constant	**Description**
XintDECREMENT	Scroll one column left.
XintINCREMENT	Scroll one column right.
XintPAGE_DECREMENT	Scroll one page left.
XintPAGE_INCREMENT	Scroll one page right.
XintTO_FIRST	Scroll back to the first column.
XintTO_LAST	Scroll to the last column.

XintEditTableConfirmEdit

This function allows the confirmation of the editing of the current cell and the invocation of the callback XmNvalidateValueCallback. This function is invoked by action TextConfirmEdit.

```
void XintEditTableConfirmEdit (Widget widget)
```

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

XintEditTableCopyColumn

This function copies a specified column to the EditTable widget's clipboard.

```
Boolean XintEditTableCopyColumn (...)
```

Widget	widget	EditTable widget ID.
int	column	Column index.

This function returns False if the column index is out of range. Otherwise, it returns True.

XintEditTableCopyRows

This function copies a specified range of rows to the EditTable widget's clipboard.

```
Boolean XintEditTableCopyRows (...)
```

Widget	widget	EditTable widget ID.
int	row	Row index of the first row to copy.
int	num_rows	Number of rows to copy including the first row.

XintEditTableDefineColumnFormat

This function defines the attributes of a specified column.

```
Boolean XintEditTableDefineColumnFormat (...)
```

Widget	widget	EditTable widget ID.
int	column	Column index.
int	edit_mode	Specify whether the cells in the column are editable by using one of the following defined constants: XintCOLUMN_EDITABLE or XintCOLUMN_NON_EDITABLE.
int	alignment	Specify the alignment of the values in the cells of the column using one of the defined constants below.
int	max_char	The width of the column in terms of the number of characters displayed in a cell.
int	data_type	Specify the data type of the values in the cells of the column using one of the defined constants below.
char *	data_format	Specify the format of the values in the cells of the column using a character string containing a C language format descriptor (such as "%d") NOTE: the data format must be consistent with the data type. Specify NULL if you want to format the cells in the column yourself using callback XmNformatCellCallback. This format descriptor is ignored for pointer data in which case callback XmNformatCellCallback is always called.

Specify constant XintCOLUMN_DEFAULT if you want the EditTable to use the default value for one of the integer arguments above. The *alignment* argument can be one of the following:

Resource Value	Description
XintALIGNMENT_BEGINNING_TOP	Value to be justified in upper left hand corner of the cell.
XintALIGNMENT_CENTER_TOP	Value to be justified horizontally in center of cell and vertically at top of cell.
XintALIGNMENT_END_TOP	Value to be justified in upper right hand corner of cell.
XintALIGNMENT_BEGINNING_MIDDLE	Value to be justified horizontally at left side of cell and vertically in center of cell.
XintALIGNMENT_CENTER_MIDDLE	Value to be justified horizontally at center of cell and vertically in center of cell.
XintALIGNMENT_END_MIDDLE	Value to be justified horizontally at right side of cell and vertically in center of cell.
XintALIGNMENT_BEGINNING_BOTTOM	Value to be justified in lower left hand corner of cell.
XintALIGNMENT_CENTER_BOTTOM	Value to be justified horizontally in center of cell and vertically at bottom of the cell.
XintALIGNMENT_END_BOTTOM	Value is to be justified in lower right hand corner of cell.

The *data_type* can be one of the following:

Defined Constant	Description
XintTYPE_DOUBLE	Values are double precision floating point numbers.
XintTYPE_FLOAT	Values are single precision floating point numbers.
XintTYPE_INTEGER	Values are integer numbers.
XintTYPE_LONG_INTEGER	Values are long integer numbers.
XintTYPE_SHORT	Values are short integer numbers.
XintTYPE_STRING	Values are character strings.
XintTYPE_POINTER	Values are pointers.

Function XintEditTableGetColumnAttrinutes retrieves column attributes.

XintEditTableDeleteColumns

This function deletes a specified range of columns from a table. The deleted columns are copied to the EditTable Widget's clipboard so that they can be undeleted if necessary.

```
Boolean XintEditTableDeleteColumns (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the first column to delete.
int	num_columns	Number of columns to delete.

This function returns False if the starting column index or number of columns is out of range. Otherwise, it returns True.

XintEditTableDeleteRows

This function deletes a specified range of rows from a table. The deleted rows are copied to the EditTable Widget's clipboard so that they can be undeleted if necessary.

```
Boolean XintEditTableDeleteRows (...)
```

Widget	widget	EditTable widget ID.
int	row	Index of the first row to delete.
int	num_rows	Number of rows to delete.

This function returns False if the starting row index or number of rows is out of range. Otherwise, it returns True.

XintEditTableFillCell

This function stores a value into a cell and updates the display.

```
Boolean XintEditTableFillCell (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the column containing the cell.
int	row	Index of the row containing the cell.
XtPointer	data_address	Address of the value for the cell (if data type is XintTYPE_POINTER, pass pointer value directly).

This function returns False if the column index or row index is out of range or if the *data_address* is NULL. Otherwise, it returns True. This function will not work for the cell that is being edited. Use the resource **XmNvalidateValueCallback** to define a callback procedure that changes the value of the cell being edited.

You can use this function to set the annotation for a row in the table by specifying the defined constant XintROW_ANNOTATION for the value of *column* and setting *data_address* to the address of the memory space containing the new row annotation.

Warning: This function is slow and should not be used to update a whole table or a large portion a table. Use XintEditTableFillCellNoUpdate instead.

XintEditTableFillCellNoUpdate

This function is similar to XintEditTableFillCell except that it does not update the display. This function is designed for cases where the application needs to update a large number of cells as fast as possible. After all the cells have been filled, the application needs to call function XintEditTableUpdateDataDisplay, with both row and column arguments set to XintUPDATE_ALL, to update the EditTable display.

```
    Boolean XintEditTableFillCellNoUpdate (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the column containing the cell.
int	row	Index of the row containing the cell.
XtPointer	data_address	Address of the value for the cell (if data type is XintTYPE_POINTER, pass pointer value directly).

This function returns False if the column index or row index is out of range or if the *data_address* is NULL. Otherwise, it returns True. This function will not work for the cell that is being edited. Use the resource **XmNvalidateValueCallback** to define a callback procedure that changes the value of the cell being edited.

XintEditTableFillColumnAnnotation

This function sets the annotation character string for a specified column.

```
Boolean XintEditTableFillColumnAnnotation (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the column whose annotation is to be changed.
char *	annotation_string	Character string containing the annotation for the column.

This function returns False if *column* is out of range or if the column specified is frozen. Otherwise, it returns True.

XintEditTableFillColumnData

This function sets the values of every cell in a specified column. The values are specified by an array that must be the same size as the specified column and be of the same data type as the column.

```
Boolean XintEditTableFillColumnData (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the column whose values are to be changed.
XtPointer	data_array	Array containing the new values for the cells in the specified column.

This function returns False if the column index is out of range or if *data_array* is NULL. Otherwise, it returns True.

You can use this function to set the annotation for every row in the table by specifying the defined constant XintROW_ANNOTATION for the value of column and putting the row annotation strings in *data_array*.

XintEditTableFreezeColumn

This function freezes a specified column. This function is only effective when the parent of the EditTable widget is an INT Scroll widget.

```
Boolean XintEditTableFreezeColumn (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the column to be frozen.

This function returns False if the column number is out of range, if the column is already frozen or if the parent of widget is not an INT Scroll widget. Otherwise, it returns True.

XintEditTableFreezeRow

This function freezes a specified row. This function is only effective when the parent of the EditTable widget is an INT Scroll widget.

```
Boolean XintEditTableFreezeRow (...)
```

Widget	widget	EditTable widget ID.
int	row	Index of the row to be frozen.

This function returns False if the row number is out of range, if the row is already frozen or if the parent of widget is not an INT Scroll widget. Otherwise, it returns True.

XintEditTableFreezeUpdate

This function is a convenience function that sets the value of resource **XmNfreezeUpdate**. Resource **XmNfreezeUpdate** when set to True will disable all geometry updates and redisplay operations while an application is performing a series of changes on the table. After the changes are completed, resource **XmNfreezeUpdate** should be set back to False so that the table can automatically recalculate its new geometry and redisplay itself.

```
Boolean XintEditTableFreezeUpdate (...)
```

Widget	widget	EditTable widget ID.
Boolean	state	True to freeze updates, False to calculate new geometry and redisplay.

XintEditTableGetCellBackground

This function returns the background color of a specified cell (as a pixel value).

```
Pixel XintEditTableGetCellBackground (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the column.
int	row	Index of the row.

This function returns the color used for the background of the specified cell as a pixel value.

XintEditTableGetCellData

Returns a pointer to a copy of the value in a specified cell. You should free the memory used by the copy of the value after you have finished with it.

```
    XtPointer XintEditTableGetCellData (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the column containing the cell.
int	row	Index of the row containing the cell.

Returns NULL if the column index or row index is out of range or if the value of the cell is undefined. Otherwise, it returns a pointer to a copy of the cell's value, unless the value is a pointer type. In the case of a pointer type, e.g., String, the pointer itself is returned (and it points to the original cell value).

XintEditTableGetCellFont

Returns the index of the font used by the specified cell.

```
    int XintEditTableGetCellFont (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the column.
int	row	Index of the row.

Returns XintUNDEFINED_INTEGER if the row or column specification is out of range or if the font index assigned to the cell was not valid.

XintEditTableGetCellForeground

Returns the foreground color of a specified cell (as a pixel value).

```
    Pixel XintEditTableGetCellForeground (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the column.
int	row	Index of the row.

Returns the pixel value used to paint the foreground of the specified cell.

XintEditTableGetCellGeometry

Returns the position and size in pixels coordinates of a cell relative to the upper left corner of the EditTable widget window.

```
    Boolean XintEditTableGetCellGeometry (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the column.
int	row	Index of the row.
int *	x	Returns the X-coordinate of the cell.
int *	y	Returns the Y-coordinate of the cell.
int *	width	Returns the cell's width in pixels.
int *	height	Returns the cell's height in pixels.

Returns False if the specified cell is out of range.

XintEditTableGetCellHeight

Returns the height of the specified cell in the unit system specified by resource **XmNcellSizeUnit**. Note that all the cells in a defined row (column if table is transposed) have the same height.

```
    Boolean XintEditTableGetCellHeight (...)
```

Widget	widget	EditTable widget ID.
int	index	Index of the row (column if table is transposed) containing the cell.
int *	height	Returns the height of the cell.

Returns False if the index specified is out of range.

XintEditTableGetCellPixmap

Returns the pixmap ID used for the background of the specified cell.

```
    Pixmap XintEditTableGetCellPixmap (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the column.
int	row	Index of the row.

Returns XintUNDEFINED_PIXMAP if no pixmap is assigned to that cell or if the cell specification is out of range.

XintEditTableGetCellPointerPosition

This function can be used to query the location of the cell pointer. It returns False it the cell pointer is not active.

```
    Boolean XintEditTableGetCellPointerPosition (...)
```

Widget	widget	EditTable widget ID.
int *	column	Returns the index of the column where the cell pointer is located.
int *	row	Returns the index of the row where the cell pointer is located.

XintEditTableGetCellWidget

Returns the cell widget associated with a particular cell location. If no widget is associated with the cell, the function returns NULL.

```
    Widget XintEditTableGetCellWidget (...)
```

Widget	table	EditTable widget ID.
int	column	The index of the column location of the cell widget.
int	row	The index of the row location of the cell widget.

XintEditTableGetCellWidth

Returns the width of the specified cell in the unit system specified by resource **XmNcellSizeUnit**. Note that all the cells in a defined column (row if table is transposed) have the same width.

```
    Boolean XintEditTableGetCellWidth (...)
```

Widget	widget	EditTable widget ID.
int	index	Index of the column (row if table is transposed) containing the cell.
int *	width	Returns the width of the cell.

Returns False if the index specified is out of range.

XintEditTableGetColumnAnnotationAlignment

Returns the annotation alignment for a specific column.

```
Boolean XintEditTableGetColumnAnnotationAlignment(...)
```

Widget	widget	Widget ID of the EditTable.
int	col	Column number to retrieve the annotation alignment.
int *	annot_align	Pointer to annotation alignment for *col*.

Returns False if widget is not a valid EditTable widget or if column is out of range. Returns True otherwise.

XintEditTableGetColumnAttributes

Obtains the attributes of a a specified column. If one or more of the attributes is of no interest to you, then set the corresponding argument to NULL.

```
Boolean XintEditTableGetColumnAttributes (...)
```

Widget	widget	EditTable widget ID.
int	column	Column index.
int *	size	Returns number of cells in the column.
int *	edit_mode	Returns column edit mode as one of defined integer constants below.
int *	alignment	Returns column alignment as one of defined integer constants below.
int *	width	Returns width of the column in unit system specified by resource XmNcellSizeUnit.
int *	data_type	Returns data type of the values in the cells of column as one of defined constants below.
char **	data_format	Returns format of the values in cells of column using character string containing a C language format descriptor (such as “%d”).
char **	annotation_string	Returns character string that annotates column.

The *edit_mode* is returned as one of the following:

Defined Constant	**Description**
XintCOLUMN_EDITABLE	Indicates values in cells of column are editable.
XintCOLUMN_NON_EDITABLE	Indicates values in cells of column are not editable.

The *data_type* is returned as one of the following:

Defined Constant	**Description**
XintTYPE_DOUBLE	Values are double precision floating point numbers.
XintTYPE_FLOAT	Values are single precision floating point numbers.
XintTYPE_INTEGER	Values are integer numbers.
XintTYPE_LONG_INTEGER	Values are long integer numbers.
XintTYPE_SHORT	Values are short integer numbers.
XintTYPE_STRING	Values are character strings.
XintTYPE_POINTER	Values are pointers.

The *alignment* is returned as one of the following:

Resource Value	**Description**
XintALIGNMENT-_BEGINNING_TOP	Value to be justified in upper left hand corner of cell.
XintALIGNMENT_CENTER_TOP	Value in each cell to be justified horizontally in center of cell and vertically at top of cell.
XintALIGNMENT_END_TOP	Value in each cell to be justified in upper right hand corner of cell.
XintALIGNMENT_BEGINNING_ MIDDLE	Value in each cell to be justified horizontally at left side of the cell and vertically in center of cell.
XintALIGNMENT_CENTER_MIDDLE	Value in each cell to be justified horizontally at center of cell and vertically in center of cell.
XintALIGNMENT_END_MIDDLE	Value in each cell to be justified horizontally at right side of cell and vertically in center of cell.
XintALIGNMENT_BEGINNING_ BOTTOM	Value in each cell to be justified in lower left hand corner of cell.
XintALIGNMENT_CENTER_BOTTOM	Value in each cell to be justified horizontally in center of cell and vertically at bottom of cell.
XintALIGNMENT_END_BOTTOM	Value in each cell to be justified in lower right hand corner of cell.

To set the column attributes, use XintEditTableDefineColumnFormat.

XintEditTableGetColumnData

This function gets a copy of the values of every cell in a specified column. The values are returned in an array that you should free when you have finished with it.

```
XtPointer XintEditTableGetColumnData (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the column whose values are to be returned.
int *	size	Returns the number of elements in the array.

Returns NULL if the column index is out of range or if the column contains no data. Otherwise, it returns the address of the array.

XintEditTableGetColumnUserData

Returns a pointer to the column's user data.

```
XtPointer XintEditTableGetColumnUserData (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the column whose user data is to be retrieved.

XintEditTableGetFrozenColumns

Returns an integer array containing the column indices of the columns currently frozen in a table. The column indices are returned in an array that you should free when you are finished with it.

```
int * XintEditTableGetFrozenColumns (...)
```

Widget	widget	EditTable widget ID.
int *	num_cols	Returns the number of columns in the frozen list.

Returns NULL if there are no frozen columns. Otherwise, it returns a pointer to the array of frozen column indices.

XintEditTableGetFrozenRows

Returns an integer array containing the row indices of the rows currently frozen in a table. The row indices are returned in an array that you should free when you are finished with it.

```
    int * XintEditTableGetFrozenRows (...)
```

Widget	widget	EditTable widget ID.
int *	num_rows	Returns the number of rows in the frozen list.

Returns NULL if there are no frozen rows. Otherwise, it returns a pointer to the array of frozen row indices.

XintEditTableGetHiddenColumns

Returns the list of the columns which are currently not visible.

```
    int * XintEditTableGetHiddenColumns (...)
```

Widget	widget	EditTable widget ID.
int *	num_cols	Returns the number of hidden columns.

Argument *num_cols* is a pointer to an integer value, which contains on return the number of hidden columns. The function returns an array that contains the indices of the hidden columns or NULL if all the columns are visible. The array must be deallocated by the application after it is no longer needed.

XintEditTableGetHiddenRows

Returns the list of the rows which are currently not visible.

```
    int * XintEditTableGetHiddenRows (...)
```

Widget	widget	EditTable widget ID.
int *	num_rows	Returns the number of hidden rows.

Argument *num_rows* is a pointer to an integer value, which contains on return the number of hidden rows. The function returns an array that contains the indices of the hidden rows or NULL if all the rows are visible. The array must be deallocated by the application after it is no longer needed.

XintEditTableGetSelectedCells

Returns information regarding the selected cells.

```
    Boolean XintEditTableGetSelectedCells(...)
```

Widget	widget	EditTable widget ID.
int*	col_start	The column start value for the selection.
int*	row_start	The row start value for the selection.
int*	cols	The number of columns in the selection.
int *	rows	The number of rows in the selection.

The function returns False if no selection exists. Otherwise it returns True.

XintEditTableGetSelectedColumns

Returns information regarding the selected columns.

```
    Boolean XintEditTableGetSelectedColumns(...)
```

Widget	widget	EditTable widget ID.
int*	col_start	The column start value for the selection.
int *	cols	The number of columns in the selection.

The function returns False if no selection exists. Otherwise it returns True.

XintEditTableGetSelectedRows

Returns information regarding the selected rows.

```
    Boolean XintEditTableGetSelectedRows(...)
```

Widget	widget	EditTable widget ID.
int*	row_start	The row start value for the selection.
int *	rows	The number of rows in the selection.

The function returns False if no selection exists. Otherwise it returns True.

XintEditTableGetSelectionByNumber

Returns information regarding the specified selection. Selection numbers are positive integers which are assigned automatically by the EditTable widget. In case of a single selection, the selection_number should be set to 1. Otherwise, for a multiple selection, valid values are between 1 and the number returned by function XintEditTabelGetSelectionCount.

```
Boolean XintEditTableGetSelectionByNumber (...)
```

Widget	widget	EditTable widget ID.
int	selection_number	Number of the selection for which we want information.
int *	col_start	The column start value for selection.
int *	num_cols	The number of columns in selection.
int *	row_start	The row start value for the selection.
int *	num_rows	The number of rows in the selection.
int *	selection_mode	The type of the selection.

The function returns False if *selection_number* is invalid, or if no selection exists. Otherwise it returns True. The *selection_mode* is returned as one of the following constants:

Defined Constant	Description
XintSELECT_CELL	Selection is a block of cells.
XintSELECT_ROW	Selection is a set of rows.
XintSELECT_COLUMN	Selection is a set of columns.
XintSELECT_NONE	No selection exists.

XintEditTableGetSelectionCount

Returns the number of current selections.

```
int XintEditTableGetSelectionCount (Widget widget)
```

where *widget* is the widget ID of an EditTable widget. The functions returns the number of selections currently defined. Returns 0 if there are no current selections.

XintEditTableGetTextChild

Returns the ID of the text widget used to edit a cell. Editing is done using a floating text widget that is mapped when the user starts entering text. There are up to four text widgets, one for the main table, one for the frozen column area, one for the frozen row area and one for the area intersecting the frozen rows and columns.

```
    Widget XintEditTableGetTextChild (...)
```

Widget	widget	EditTable widget ID.
int	child	Code referring to one of the text widgets in the EditTable widget.

where *widget* is the widget ID of an EditTable widget and *child* is a constant, from the table below, which refers to one of the EditTable text widgets. The function returns the widget ID of the specified text widget.

Defined Constant	Description
XintEDIT_TABLE_MAIN_TEXT	Refers to the text widget used in the main table.
XintEDIT_TABLE_FROZEN_COLUMN_TEXT	Refers to the text widget used to edit frozen columns.
XintEDIT_TABLE_FROZEN_ROW_TEXT	Refers to the text widget used to edit frozen rows.
XintEDIT_TABLE_FROZEN_CELLS_TEXT	Refers to the text widget used to edit frozen cells (intersection of frozen rows and frozen columns).

XintEditTableGetSubtable

Returns the ID of one of the subtable widgets. A subtable may or may not exist so you should check the returned widget ID to ensure that it is not NULL.

```
    Widget XintEditTableGetSubtable (...)
```

Widget	widget	EditTable widget ID.
int	subtable_code	Code referring to one of the subtables in the EditTa-ble widget.

The function returns the widget ID of the specified subtable if it exists. Argument *subtable_code* is specified as one of the following constants:

Defined Constant	Description
XintEDIT_TABLE_FROZEN_ COLUMN_SUBTABLE	Refers to the subtable of frozen columns.
XintEDIT_TABLE_FROZEN_ROW_ SUBTABLE	Refers to the subtable of frozen rows.
XintEDIT_TABLE_FROZEN_INTER SECTION_ SUBTABLE	Refers to the subtable in the intersection of the frozen rows and columns.
XintEDIT_TABLE_MAIN_SUBTABLE	Refers to the subtable of non-frozen cells.

XintEditTableGetVisibleArea

This function passes back the visible area of the table through its argument list. The visible area of the table starts from the *first_column* (the leftmost column) to the *last_column* (the rightmost column) and from the *first_row* (the top row) to the *last_row* (the bottom row).

```
    Boolean XintEditTableGetVisibleArea (...)
```

Widget	widget	EditTable Widget ID.
int *	first_column	Returns the index of the first (leftmost) column.
int *	last_column	Returns the index of the last (rightmost) column.
int *	first_row	Returns the index of the first (top) row.
int *	last_row	Returns the index of the last (bottom) row.

Returns False if the widget argument is not an EditTable widget. Otherwise, it returns True.

XintEditTableInsertColumns

This function inserts one or more empty columns before a specified column. You can specify an array of character strings for the new column annotations or specify NULL for new columns with no annotations. Specify 0 for the column index to have the new columns inserted at the end of the table.

```
Boolean XintEditTableInsertColumns (...)
```

Widget	widget	EditTable widget ID.
int	column	The column index of the column before which the columns will be inserted.
int	num_cols	The number of empty columns to be inserted.
char **	annotation	Pointer to an array of character strings each of which contains the annotation for one of new columns.

Returns False if the specified column index is out of range. Otherwise, it returns True.

XintEditTableInsertRows

This function inserts one or more empty rows before a specified row. Specify 0 for the row index to cause the new rows to be inserted at the end of the table.

```
Boolean XintEditTableInsertRows (...)
```

Widget	widget	EditTable widget ID.
int	row	The row index of the row before which the new rows will be inserted.
int	num_rows	The number of rows to be inserted.

Returns False if the specified row index is out of range. Otherwise, it returns True.

XintEditTableIsCellDefined

Returns True if the specified cell is within the table and if its content has been initialized.

```
Boolean XintEditTableIsCellDefined (...)
```

Widget	widget	EditTable widget ID.
int	column	The column index of the cell of interest.
int	row	The row index of the cell of interest.

XintEditTableIsColumnFrozen

Returns True if the specified column is frozen.

```
Boolean XintEditTableIsColumnFrozen (...)
```

Widget	widget	EditTable widget ID.
int	column	The index of the column of interest.

XintEditTableIsRowFrozen

Returns True if the specified row is frozen.

```
Boolean XintEditTableIsRowFrozen (...)
```

Widget	widget	EditTable widget ID.
int	row	The index of the row of interest.

XintEditTableIsColumnHidden

Returns True if the specified column is hidden. A column is hidden by calling function XintEditTableChangeColumnVisibility with the *visible* argument set to False.

```
Boolean XintEditTableIsColumnHidden (...)
```

Widget	widget	EditTable widget ID.
int	column	The index of the column of interest.

XintEditTableIsRowHidden

Returns True if the specified row is hidden.A row is hidden by calling function XintEditTableChangeRowVisibility with the *visible* argument set to False.

```
Boolean XintEditTableIsRowHidden (...)
```

Widget	widget	EditTable widget ID.
int	row	The index of the row of interest.

XintEditTableOutputAscii

This function creates an ASCII output file that contains the values in a range of cells. You must specify a delimiter to be inserted between the values in a row.

```
    Boolean XintEditTableOutputAscii (...)
```

Widget	widget	EditTable widget ID.
char *	filename	Character string specifying the output file name.
int	col_start	Column index of the first column containing cell values to be output.
int	col_end	Column index of the last column containing cell values to be output.
int	row_start	Row index of the first row containing cell values to be output.
int	row_end	Row index of the last row containing cell values to be output.
char	delimiter	Character to be inserted between values in a row (e.g. ‘,‘).

Returns False if the file is not opened successfully or if any of the row or column indices are out of range. Otherwise, it returns True. Refer to *“XintEditTableReadAscii”* on page 175 for a description of the ASCII file format.

XintEditTableOutputPostscript

This function creates a PostScript output file that contains the values in a range of specified cells.

```
    Boolean XintEditTableOutputPostscript (...)
```

Widget	widget	EditTable widget ID.
char *	filename	Character string specifying the output file name.
int	color_mode	Specify XintMONOCHROME for a black and white printer and XintCOLOR for a color printer.
float	scale	Specifies the scale factor for the output image
int	col_start	Column index of the first column containing cell values to be output.
int	col_end	Column index of the last column containing cell values to be output.
int	row_start	Row index of the first row containing cell values to be out-put.
int	row_end	Row index of the last row containing cell values to be output.

The *scale* argument in the function call specifies the scaling that will be applied to the table to produce the output hardcopy display. A value of zero (0) causes the total set of selected cells to be scaled to fit the page. This means that all of the cells in the range of *col_start* to *col_end* and *row_start* to *row_end* will be displayed on a single sheet of paper. A value of one (1) causes the output cells to be displayed at their screen size. If a cell measures 1-inch by 3-inches on the screen, it will have those same dimensions on the output display. In general (and depending on the number of cells defined by the column and row ranges), a scale value of 1 will generate more than one page of output. Larger scale values create proportionately greater than screen size displays; while values between 0 and 1 cause the display to be scaled to less than screen size. Remember that the range of cells to be displayed may be significantly greater than the amount of the table that is visible in the viewport on the screen.

Returns False if the file is not opened successfully or if any of the row or column indices are out of range. Otherwise, it returns True.

The XintOutputPostscript function provides a more general hardcopy capability for any widget instance derived from the CompBase widget class. It is described in that section.

XintEditTableOutputSimplePS

This function generates Postscript output, ignoring all font specifiers, and using only Courier-10.

```
Boolean XintEditTableOutputSimplePS (...)
```

Widget	widget	EditTable widget ID.
char *	filename	Character string specifying the output PostScript file name.
int	orientation	Orientation of output. Use the constant of either XintORIENTATION_LANDSCAPE or XintORIENTATION_PORTRAIT.
int	start_column	Starting column for output to PostScript.
int	end_column	Ending column for output to PostScript.
int	start_row	Starting row for output to PostScript.
int	end_row	Ending row for output to PostScript.

Returns False if the file is not opened successfully or if any of the row or column indices are out of range. It also returns False if *widget* is not a valid EditTable widget, otherwise, it returns True.

XintEditTableOutputSimplePS2

This function generates Postscript output, using all fonts and alignments set up in the EditTable. If the scale is set to 0, the output will be fitted to one page.

```
Boolean XintEditTableOutputSimplePS2 (...)
```

Widget	widget	EditTable widget ID.
char *	filename	Character string specifying the output PostScript file name.
int	orientation	Orientation of output. Use the constant of either XintORIENTATION_LANDSCAPE or XintORIENTATION_PORTRAIT.
double	scale	Scale to use for output to PostScript. Using a scale of 0 will fit the output EditTable to one page.
int	start_column	Starting column for output to PostScript.
int	end_column	Ending column for output to PostScript.
int	start_row	Starting row for output to PostScript.
int	end_row	Ending row for output to PostScript.

Returns False if the file is not opened successfully or if any of the row or column indices are out of range. It also returns False if *widget* is not a valid EditTable widget, otherwise, it returns True.

XintEditTableOutputSylkFile

Creates a SYLK formatted output file that contains the values in a range of specified cells. You can specify a font on the machine the SYLK file will be used on. Alternatively, specify NULL to use the default font.

```
Boolean XintEditTableOutputSylkFile (...)
```

Widget	widget	EditTable widget ID.
char *	filename	Character string specifying output file name.
int	col_start	Column index of first column containing cell values to output.
int	col_end	Column index of last column containing cell values to output.
int	row_start	Row index of first row containing cell values to output.
int	row_end	Row index of last row containing cell values to output.
char *	font_name	Character string specifying name of font to use in output file.

Returns False if the file is not opened successfully or if any of the row or column indices are out of range. Otherwise, it returns True.

XintEditTablePasteColumns

This function copies the columns currently on the EditTable widget's clipboard into the columns of the table starting at a specified column. You must specify whether a data type conversion will or will not be performed when the data is pasted from the clipboard to the specified column. If no conversion is specified and the column types are not compatible (e.g. trying to paste strings into a floating point column), then no operation is performed and the function returns False.

```
    Boolean XintEditTablePasteColumns (...)
```

Widget	widget	EditTable widget ID.
int	column	The column index of the column whose values will be replaced by the values of the column on the clipboard.
int	conversion	Specify the type of conversion performed on the data by using one of the defined integer constants below.

where *conversion* is specified as one of the following:

Defined Constant	Description
XintCONVERSION	Indicates that the values should be converted (e.g. "123" becomes the integer value 123).
XintNO_CONVERSION	Indicates that no conversion is to be performed. Function will return False on data type mismatch.
XintTYPE_CAST	Indicates that the values should be type cast (e.g. "123" becomes (int) "123").

Returns False if specified column index is out of range or a data type mismatch occurs when conversion set to XintNO_CONVERSION. Otherwise, returns True.

XintEditTablePSReportStyle

This function generates a postscript formatted report.

```
    Boolean XintEditTablePSReportStyle (...)
```

Widget	table	EditTable widget ID.
EditTableReportRange *	range	Pointer to EditTableReportRange structure that specifies the range to be generated.
EditTableReportLayout *	layout	Pointer to EditTableReportLayout structure that specifies the layout of the report.
EditTableReportAttributes *	attributes	Pointer to EditTableReportAttributes that specifies the page attributes of the report.
char *	filename	The output file name.

Returns False if the widget is not an EditTable widget or if there exist s a column/row whose width/height is greater than the page width or page height. It is the programmer's responsibility to free the parameters (range, layout, attributes) after completion.

The structures used by this function are as follows:

```
typedef struct {

  int column_start, column_end, row_start, row_end;

} EditTableReportRange;
```

where the structure variables are:

column_start	Starting column in the range of cells.
column_end	Ending column in the range of cells.
row_start	Starting row in the range of cells.
row_end	Ending row in the range of cells.

```
typedef struct {

  int orientation, processing_direction;

} EditTableReportLayout;
```

where the structure variables are:

orientation	Specifies the orientation of the page. The user can specify any one of the constants shown in the first Constants table below.
processing_direction	Specifies the page stacking order. The user can specify any one of the constants shown in the second Constants table below.

Defined Constant	**Description**
XintORIENTATION_PORTRAIT	Specifies the portrait orientation.
XintORIENTATION_LANDSCAPE	Specifies the landscape orientation.
XintFROM_L_TO_R	Specifies to process the table from left to right (process the table in terms of columns).
XintFROM_T_TO_B	Specifies to process the table from top to bottom (process the table in terms of rows).

```
typedef struct {

  int       show_on_page;
  String    string;
  String    font;
  int       placement;
  int       alignment;

} FieldAttr;

typedef struct {

  FieldAttr   title;
  FieldAttr   horz_annotation;
  FieldAttr   vert_annotation;
  FieldAttr   page_number;
  String      table_text_font;
  int         table_text_alignment;

} EditTableReportAttributes;
```

where the variables of structure FieldAttr are:

show_on_page	Indicates whether to show on every page or just the front page. The user can specify one of the constants in the Constants table below.
string	The actual string in the report.
font	The actual font string to be used in the report.
placement	Specifies the placement of the EditTableReportAttributes members. The user can specify one of the constants in the Constants table below
alignment	Specifies the alignment of the EditTableReportAttributes members. The user can specify one of the constants in the Constants table below.

The constants available for members of the FieldAttr structure are:

Defined Constant	Description
XintALL_PAGES	Constant value for show_on_page variable in structure FieldAttr. Indicates the user wishes to show attribute on all pages in the report.
XintFRONT_PAGE_ONLY	Constant value for show_on_page variable in structure FieldAttr. Indicates the user wishes to show attribute on front page only.
XintPLACEMENT_NONE	Constant value for placement variable. Indicates not to place the attribute on page.
XintPLACEMENT_TOP	Constant value for placement variable. Indicates to place the attributes above the report content.
XintPLACEMENT_BOTTOM	Constant value for placement variable. Indicates to place the attributes below the report content.
XintPLACEMENT_TOP_BOTTOM	Constant value for placement variable. Indicates to place the attributes both above and below the report content.
XintPLACEMENT_LEFT	Constant value for placement variable. Indicates to place the attributes to the left of the report content.
XintPLACEMENT_RIGHT	Constant value for placement variable. Indicates to place the attributes to the right of the report content.
XintPLACEMENT_LEFT_RIGHT	Constant value for placement variable. Indicates to place the attributes both left and right of the report content.
XintALIGNMENT_BEGINNING	Constant value for alignment variable. Indicates to align the string starting from the left border of a cell.
XintALIGNMENT_CENTER	Constant value for alignment variable. Indicates to align the string at the center of a cell
XintALIGNMENT_END	Constant value for alignment variable. Indicates to align the string ending at the right border of a cell

The following table lists the variables of structure EditTableReportAttribute:

Variable	Description
title	Title specification.
horz_annotation	Specifies the horizontal annotations.
vert_annotation	Specifies the vertical annotations,
page_number	Page number specification.
table_text_font	Specifies the text font for the report, if other than the default table font.
table_text_alignment	Specifies the text alignment for the report, if other than the default table text alignment.

XintEditTablePasteRows

This function copies the rows currently on the EditTable widget's clipboard into the rows of the table starting at a specified row.

```
    Boolean XintEditTablePasteRows (...)
```

Widget	widget	EditTable widget ID.
int	row	The row index of the row where the paste will begin.

Returns False if the specified row index is out of range.

XintEditTableReadAscii

This function reads data from an ASCII file into the EditTable widget. The EditTable automatically converts the string data into the data type value of each column. If the conversion fails for a cell, the cell will be left unchanged. Set the column data type to XintTYPE_STRING to have no conversion performed (refer to*"XmNdefaultColumnDataType"* on page 96 and *"XmNcolumnDataTypeData"* on page 95 for more information).

```
    Boolean XintEditTableReadAscii (...)
```

Widget	widget	EditTable widget ID.
char *	filename	The name of the ASCII file.
Boolean	resize	EditTable widget resizes itself to match size of input dataset if resize is True. Otherwise, table size remains unchanged. In that case, if the dataset is larger than the table, the extra values are ignored, and if the table is larger than the dataset, the remaining table cells will be left unchanged.

Returns False if it cannot open the specified file.

Each line in the ASCII file corresponds to a row in the table. The default delimiter is a tab. If no tab is found in the first line, the widget will try a comma as the delimiter. If no comma is found, it will try a space. You can also explicitly specify the character to use as the delimiter by inserting the following command at the beginning of the file:

```
#DELIMITER = ‘delimiter’
```

where delimiter is the character to use as the delimiter. The file format supports missing values. A missing value is simply omitted and its following delimiter is supplied in its place. To specify the column annotation from the ASCII file, insert the following command at the beginning of the file:

```
#COLUMN ANNOTATION
```

This command specifies that the first row of data should be used as the column annotation. To specify the row annotation from the ASCII file, insert the following command at the beginning of the file.

```
#ROW ANNOTATION
```

This command specifies that the first value of each row is to be used as the row annotation for that line. An example of a valid ASCII file follows (Note - the entry for row 3, column 3 is missing):

```
#DELIMITER=’,’
#COLUMN ANNOTATION
#ROW ANNOTATION
Houston,Dallas,San Antonio,Austin
QTR 1,12.5,21.4,34.6,12.6
QTR 2,11.5,22.6,41.4,14.8
QTR 3,14.1,27.6,19.5
QTR 4,19.5,28.5,43.5,25.1
```

XintEditTableReleaseColumn

This function releases a specified frozen column.

```
Boolean XintEditTableReleaseColumn (...)
```

Widget	widget	EditTable widget ID.
int	column	The index of the frozen column to be released.

Returns False if the specified column index is out of range or if the column is not frozen. Otherwise, it returns True.

XintEditTableReleaseRow

This function releases a specified frozen row.

```
    Boolean XintEditTableReleaseRow (...)
```

Widget	widget	EditTable widget ID.
int	row	The index of the frozen row to be released.

Returns False if the specified row index is out of range or if the row is not frozen. Otherwise, it returns True.

XintEditTableReorderColumns

This function changes the order of a list of columns. The columns to be reordered must be a contiguous sequence of columns. The new order is specified as an array of original column indices indicating the new ordering of the columns. For instance, if columns 5 through 10 are to be reordered, then the new order can be specified as an integer array containing the new order 8, 5, 9, 7, 6, 10; where the original column 8 is now column 5.

```
    Boolean XintEditTableReorderColumns (...)
```

Widget	widget	EditTable widget ID.
int	col_start	Column index of the first column to be reordered.
int	col_end	Column index of the last column to be reordered.
int *	order	Pointer to integer array containing column indices in different order.

Returns False if the specified columns could not be reordered.

XintEditTableReorderRows

This function changes the order of a list of rows. The rows to be reordered must be a contiguous sequence of rows. The new order is specified as an array of original row indices indicating the new ordering of the rows. For instance, if rows 5 through 10 are to be reordered, then the new order can be specified as an integer array containing the new order 8, 5, 9, 7, 6, 10; where the original row 8 is now row 5.

```
    Boolean XintEditTableReorderRows (...)
```

Widget	widget	EditTable widget ID.
int	row_start	The row index of the first column to be reordered.
int	row_end	The column index of the last row to be reordered.
int *	order	Pointer to integer array containing row indices in a different order.

Returns False if the specified rows could not be reordered.

XintEditTableRemoveAllLocalCallbacks

This function removes all local callback procedures associated with a specified callback resource.

```
    Boolean XintEditTableRemoveAllLocalCallbacks (...)
```

Widget	widget	EditTable widget ID.
char *	callback_name	Callback resource name defined by EditTable widget.

Returns False if *callback_name* is not a valid callback resource name.

XintEditTableRemoveLocalCallbacks

This function removes a local callback attached to a cell, row or column.

```
    Boolean   XintEditTableRemoveLocalCallbacks (...)
```

Widget	widget	EditTable widget ID.
int	col	Index of column where local callback is registered.
int	row	Index of the row where local callback is registered.
char *	callback_name	Name of the callback to remove.
XtCallbackProc	callback_proc	Callback procedure to remove.
XtPointer	client_data	Client data attached to callback_proc.

Returns False if specified column or row indices are out of range or if the name of the callback is invalid.

XintEditTableRowScroll

This function allows the program to control scrolling of rows.

```
    void   XintEditTableRowScroll (...)
```

Widget	widget	EditTable widget ID.
int	scroll_operation	Specifies the direction and amount of scrolling.

The operation argument must be one of the following:

Defined Constant	Description
XintDECREMENT	To scroll one row up.
XintINCREMENT	To scroll one row down.
XintPAGE_DECREMENT	To scroll one page up.
XintPAGE_INCREMENT	To scroll one page down.
XintTO_FIRST	To scroll back to the first row.
XintTO_LAST	To scroll to the last row.

XintEditTableSetCellBackground

This function sets the background color of a specified block of cells.

```
    Boolean   XintEditTableSetCellBackground (...)
```

Widget	widget	EditTable widget ID.
int	col_start	The index of the starting column.
int	num_cols	Number of columns.
int	row_start	The index of the starting row.
int	num_rows	The number of rows.
Pixel	pixel	The pixel value used to draw the cell background.

Returns False if the cell specification is out of range. Otherwise, returns True.

XintEditTableSetCellDisplayAttributes

Sets the cell display attributes of the specified range of cells.

```
    Boolean   XintEditTableSetCellDisplayAttributes (...)
```

Widget	widget	EditTable widget ID.
int	col_start	Index of the starting column.
int	num_cols	Number of columns.
int	row_start	Index of the starting row.
int	num_rows	Number of rows.
Pixel	background	Background color of the cells.
Pixel	foreground	Foreground color of the cells.
XtPointer	data_addr	Address of value for cell (if data type is XintTYPE_POINTER, pass pointer value directly).

Returns False if widget is not an EditTable widget.

XintEditTableSetCellFont

This function sets the index of the font used for displaying the specified set of cells. This function will have no effect if resource **XmNfontTable** is set to NULL.

```
    Boolean   XintEditTableSetCellFont (...)
```

Widget	widget	EditTable widget ID.
int	col_start	The index of the starting column.
int	num_cols	Number of columns.
int	row_start	The index of the starting row.
int	num_rows	The number of rows.
int	index	The index of the font into the font table (starts at 0). Specify -1 to use the default table font.

The function returns False if the cell specification is out of range. Otherwise, it returns True.

XintEditTableSetCellForeground

This function sets the foreground color of a specified block of cells.

```
    Boolean   XintEditTableSetCellForeground (...)
```

Widget	widget	EditTable widget ID.
int	col_start	The index of the starting column.
int	num_cols	Number of columns.
int	row_start	The index of the starting row.
int	num_rows	The number of rows.
Pixel	pixel	The pixel value used to draw the cell foreground.

The function returns False if the cell specification is out of range. Otherwise, it returns True.

XintEditTableSetCellHeight

This function sets the height, in the unit system specified by resource **XmNcellSizeUnit**, of a specified block of cells. Note that all the cells in a row (column if table is transposed) have the same height.

```
    Boolean   XintEditTableSetCellHeight (...)
```

Widget	widget	EditTable widget ID.
int	index_start	The index of the starting row (column if table is transposed).
int	num_units	Number of rows (columns if table is transposed) to change.
int	height	The height specification.

The function returns False if the index specification is out of range. Otherwise, it returns True.

XintEditTableSetCellPixmap

This function sets the background pixmap of a specified block of cells. A pixmap of depth 1 or a depth equal to that of the EditTable widget window is supported. For a bitmap (pixmap of depth 1), the cell foreground color is used for the set bit and the background color is used for the unset bit.

```
    Boolean XintEditTableSetCellPixmap (...)
```

Widget	widget	EditTable widget ID.
int	col_start	The index of the starting column.
int	num_cols	Number of columns.
int	row_start	The index of the starting row.
int	num_rows	The number of rows.
Pixmap	pixmap	The pixmap ID. Specify XintUNDEFINED_PIXMAP to have no pixmap drawn in the background.

The function return False if the cell specification is out of range. Otherwise, it returns True.

XintEditTableSetCellPixmapList

This function sets the background pixmap of a specified block of cells from a list of pixmaps.

```
    Boolean XintEditTableSetCellPixmapList (...)
```

Widget	widget	EditTable widget ID.
int	col_start	The index of the starting column.
int	num_cols	Number of columns.
int	row_start	The index of the starting row.
int	num_rows	The number of rows.
Pixmap *	pixmap_list	The list of pixmap IDs stored column wise. There must exactly the same number of pixmaps in the list as cells specified. Specify XintUNDEFINED_PIXMAP to have no pixmap drawn in the background of a cell.

The function return False if the cell specification is out of range. Otherwise, it returns True.

XintEditTableSetCellPointerPosition

This function sets the cell pointer to the specified cell. The cell pointer indicates the cell that is being edited.

```
    Boolean XintEditTableSetCellPointerPosition (...)
```

Widget	widget	EditTable widget ID.
int	col	The column index of the cell in which to position the cell pointer.
int	row	The row index of the cell in which to position the cell pointer. Specify -1 to have the cell pointer disappear.

The function returns False if the cell location is out of range. Otherwise, it returns True.

XintEditTableSetCellWidth

This function sets the width, in the unit system specified by resource **XmNcellSizeUnit**, of a specified block of cells. Note that all the cells in a column (row if table is transposed) have the same width.

```
    Boolean XintEditTableSetCellWidth (...)
```

Widget	widget	Widget ID of the EditTable.
int	col	Column number to set the annotation alignment.
int	num_cols	Number of columns to repeat the annotation alignment.
int	annot_align	Annotation alignment.

The function return False if the index specification is out of range. Otherwise, it returns True.

XintEditTableSetColumnAnnotationAlignment

This function sets the annotation alignment, starting at a user specified column and uses that alignment for a user specified number of columns.

```
    Boolean XintEditTableSetColumnAnnotationAlignment(...)
```

Widget	widget	EditTable widget ID.
int	index_start	The index of the starting column (row if table is transposed).
int	num_units	Number of columns (rows if table is transposed) to change.
int	width	The width specification.

Returns False if widget is not a valid EditTable widget or if column is out of range. Returns True otherwise.

XintEditTableSetColumnFont

This function specifies the font index to be used by a range of columns. This function has no effect if resource XmNfontTable is NULL.

```
    Boolean XintEditTableSetColumnFont (...)
```

Widget	widget	EditTable widget ID.
int	col_start	Starting column number.
int	num_cols	Number of columns.
int	font_index	Specifies an index into the font table (starts at 0). Specify -1 to use the default font.

The function returns NULL if the column specification is out of range.

XintEditTableSetColumnUserData

This function specifies the column and the column's user data.

```
Boolean XintEditTableSetColumnUserData (...)
```

Widget	widget	EditTable widget ID.
int	column	Column number to store the user data.
XtPointer	user_data	Pointer to user data.

The function returns False if *widget* is not a valid EditTable or if the column does not exist. Returns True, otherwise.

XintEditTableSetListBehavior

This function simulates the XmList widget. The number of the selection will be equal to the number of rows selected. (When this function is "off" the selection will be equal to one no matter the number of rows selected, since they are treated as one "group".)

```
Boolean XintEditTableSetListBehavior (...)
```

Widget	widget	EditTable widget ID.
int	policy	Specifies the selection policy.
Boolean	editable	Specifies the edit mode. Editable is False by default. Setting this resource to True or False in turn sets the ColumnEditMode of the table.

The *policy* argument must include one of the following:

Defined Constant	Description
XintDEFAULT_TABLE_SELECT	Restores the default table selection mode.
XmSINGLE_SELECT	Select one row at a time.
XmMULTIPLE_SELECT	Allows multiple rows selection.
XmEXTENDED_SELECT	Allows extended selection.
XmBROWSE_SELECT	Allows single selection with "drag and browse" functionality.

Specifying one of the above arguments will make the table read only. To restore the table back to its default behavior, call the function again with *policy* set to XintDEFAULT_TABLE_SELECT. The function returns False if the widget is not an EditTable widget.

XintEditTableSetRowFont

This function specifies the font index to be used by a range of rows. This function has no effect if resource **XmNfontTable** is NULL.

```
    Boolean XintEditTableSetRowFont (...)
```

Widget	widget	EditTable widget ID.
int	row_start	Starting row number.
int	num_rows	Number of rows.
int	font_index	Specifies an index into the font table (starts at 0). Specify -1 to use the default font.

The function returns NULL if the row specification is out of range.

XintEditTableSetSelection

This function clears all previous selections and sets a new selection.The selection can be specified as a set of cells, columns or rows. The function will trigger the added selection callback function, if any, unless XintNO_CALLBACK is also included as part of the *selection_mode* argument.

```
    Boolean XintEditTableSetSelection (...)
```

Widget	widget	EditTable widget ID.
int	col_start	Starting column number.
int	num_cols	Number of columns.
int	row_start	Starting row number.
int	num_rows	Number of rows.
int	selection_mode	Specifies if selection for set of cells, rows or columns.

The *selection_mode* argument must include one of the following:

Defined Constant	Description
XintSELECT_CELL	Selection is a block of cells.
XintSELECT_ROW	Selection is a set of rows.
XintSELECT_COLUMN	Selection is a set of columns.

In addition, if the callback is NOT to be called, include the XintNO_CALLBACK constant by using bit-wise OR, as illustrated below.

```
XintEditTableSetSelection(table, 1, 2, 1, 10,
                          XintSELECT_CELL | XintNO_CALLBACK()
```

XintEditTableSortByColumn

This function sorts the rows in the specified column using the user defined comparator function.

```
int * XintEditTableSortByColumn (...)
```

Widget	widget	EditTable widget ID.
int	column	The column to be sorted.
int (*)()	comparator	The comparator function pointer.

Returns NULL if the widget is not an EditTable widget. Otherwise, it returns the sorted row index array of the specified column.

XintEditTableUndeleteColumns

This function restores to the table columns that have been deleted with the function XintEditTableDeleteColumns.

```
Boolean XintEditTableUndeleteColumns (Widget widget)
```

where *widget* is the widget ID of a EditTable widget. Returns True if the deleted columns were restored successfully to the table. Otherwise, it returns False.

XintEditTableUndeleteRows

This function restores rows to the table that have been deleted with the function XintEditTableDeleteRows.

```
Boolean XintEditTableUndeleteRows (Widget widget)
```

where *widget* is the widget ID of a EditTable widget. Returns True if the deleted rows were restored successfully to the table. Otherwise, it returns False.

XintEditTableUnfreeze

This function unfreezes a table that has been frozen in order to update one, or a small number, of cells without re-drawing the table after every change. Since XintEditTableUnfreeze does not redisplay the table, the XintEditTableUpdateDataDisplay function must be called to redisplay the cell, or cells, which have been changed.

The table should be frozen (using function XintEditTableFreezeUpdate) when several cell attributes are to be changed, such as changing both the foreground and background colors. Freezing the table prevents the cell from being re-drawn after each individual change. Of course, if only a single attribute is changed, freezing is unnecessary.

Using this function together with XintEditTableUpdateDataDisplay is more efficient than calling the XintEditTableFreezeUpdate function with the *state* argument set to False, because the application can control more precisely which part of the table to re-draw. However, this function should only be used when the changes made do not affect the geometry.

```
    Boolean XintEditTableUnfreeze (...)
```

Widget	table	EditTable widget ID.

Returns False if the *table* argument is not an EditTable widget. Otherwise, it returns True.

XintEditTableUpdateDataDisplay

This function causes the EditTable widget to redisplay the values of the specified cell, row or column. To specify a whole row or a whole column, use integer constant XintUPDATE_ALL for the other index value. To specify the whole table, set both row and column indexes to XintUPDATE_ALL. You need to call this function only if 1) you have directly changed the value of one or more array elements in the specified cell, row or column and the application is sharing column data with the EditTable widget, or 2) after you have made a series of calls to function XintEditTableFillCellNoUpdate.

```
    Boolean XintEditTableUpdateDataDisplay (...)
```

Widget	widget	EditTable widget ID.
int	column	Index of the cell or column to be updated.
int	row	Index of the cell or row to be updated.

Returns False if the specified column or row indices are out of range. Otherwise, it returns True.

Scroll Widget Class 5

Overview

The Scroll widget is a container widget that scrolls INT widgets so that the scrolled child widget's annotation remains visible during scrolling. The Scroll widget combines one or two ScrollBar widgets, a viewing area that implements a visible window onto a portion of the scrolled child, and drawing area widgets for displaying the scrolled child widget's title, horizontal annotation and vertical annotation. The application controls which of the Scroll widget's components are displayed and where they are displayed. An application that displays more than one Scroll widget can have the multiple Scroll widgets share one or two ScrollBar widgets so that they are scrolled simultaneously.

This chapter includes the following sections:

Scroll Children

A Scroll widget can manage the scrolling of only one child. In particular, the Scroll widget accepts as a scrolled child an EditTable widget. The Scroll widget creates several child widgets for displaying the annotation and for the scrolling operation. There is a convenience function provided for accessing these widgets so that the application can customize the layout, appearance or behavior of the children.

Creating a Scroll Widget

An application creates a Scroll widget as a child of any container widget using either an Xt widget creation function or the supplied convenience function XintCreateScroll.

Specifying Annotation and Annotation Placement

You specify the placement and how the title, horizontal and vertical annotation will be displayed with resources defined by the scrolled child. However, you specify the margins between the title, the annotation, the scrolled child and the Scroll widget using resources defined by the Scroll widget. Note that you must explicitly set the height of the horizontal annotation window(s) and the width of the vertical annotation window(s) using Scroll resources.

Specifying Scrollbar Display Policy and Placement

Using resources, you specify whether the scroll bars will always be displayed or if they will be displayed only when needed. You also specify where the scroll bars will be placed using Scroll resources. You can further customize the appearance and behavior of the scroll bars by accessing them directly using their widget IDs which are obtained via a convenience function. If the application has created a ScrollBar widget used by a Scroll widget, then the application is responsible for its placement.

Specifying a 3-D look for the Viewing Areas

The viewing areas for the scrolled widget, its title, its horizontal annotation and its vertical annotation are enclosed within Frame widgets that have a 3-D appearance.

Synchronized Scrolling

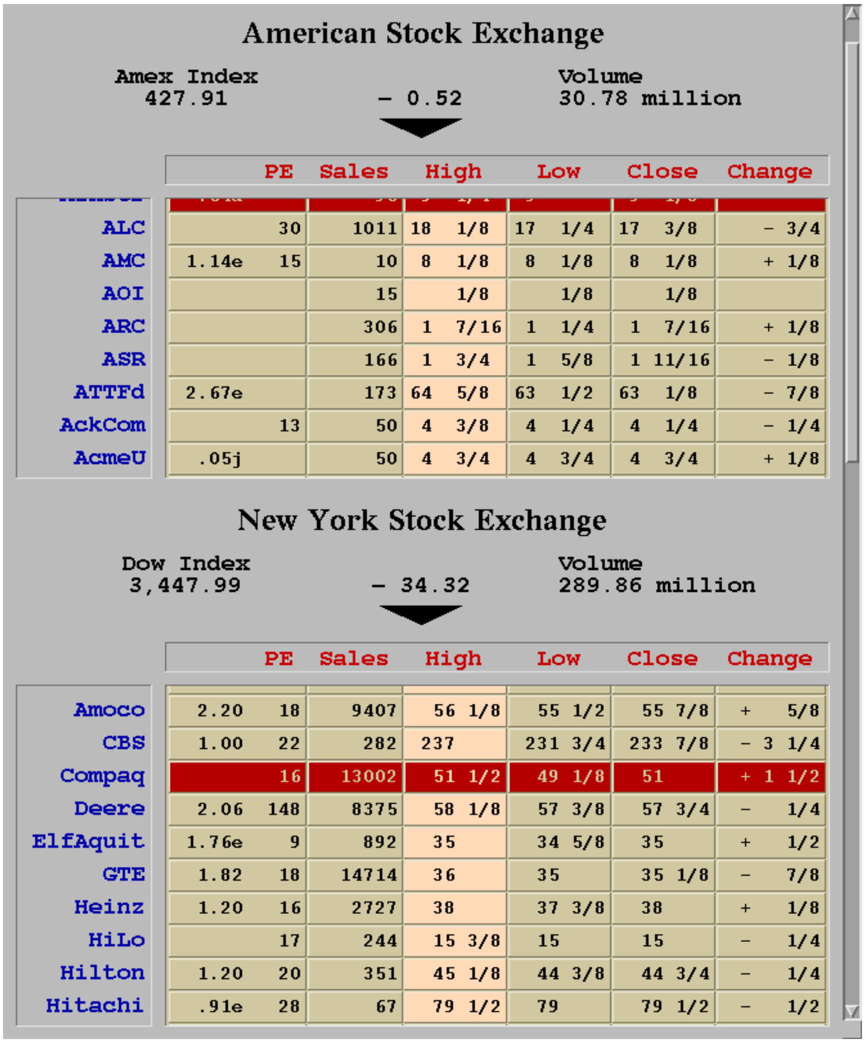
Multiple Scroll widgets can share scrollbar(s), so that it is possible to synchronize the scrolling of the scrolled child widgets in the horizontal and/or vertical direction. If two scrolled child widgets that are connected to the same scrollbar have a different height and/or width, the scrolling rate will be proportional to the height and/or width of each child.

Connecting Scrollbars to Scroll Widgets

You accomplish synchronized scrolling by using the Scroll resources, **XmNhorizontalScrollBar** and **XmNverticalScrollBar**, to specify (at widget creation time) that a Scroll widget is to use a specific scrollbar to control the scrolling of its scrolled child. T he shared scrollbar(s) can be XmScrollBar widget(s) created by the application or XmScrollBar widget(s) created automatically when a Scroll widget was created. To obtain the widget ID for a scrollbar created by a Scroll widget use the XintScrollGetChild function. To obtain the widget ID of the Scroll widget controlling a specific ScrollBar widget, get the value of the ScrollBar widget's **XmNuserData** resource.

Appearance

Figure 20 shows a vertical scrollbar which is shared by the two scrolled child widgets and thus scrolls both of them simultaneously.



American Stock Exchange

Amex Index 427.91 — 0.52 Volume 30.78 million

		PE	Sales	High	Low	Close	Change
[illegible]	[illegible]		[illegible]	[illegible]	[illegible]	[illegible]	
ALC		30	1011	18 1/8	17 1/4	17 3/8	– 3/4
AMC	1.14e	15	10	8 1/8	8 1/8	8 1/8	+ 1/8
AOI			15	1/8	1/8	1/8	
ARC			306	1 7/16	1 1/4	1 7/16	+ 1/8
ASR			166	1 3/4	1 5/8	1 11/16	– 1/8
ATTFd	2.67e		173	64 5/8	63 1/2	63 1/8	– 7/8
AckCom		13	50	4 3/8	4 1/4	4 1/4	– 1/4
AcmeU	.05j		50	4 3/4	4 3/4	4 3/4	+ 1/8

New York Stock Exchange

Dow Index 3,447.99 — 34.32 Volume 289.86 million

		PE	Sales	High	Low	Close	Change
Amoco	2.20	18	9407	56 1/8	55 1/2	55 7/8	+ 5/8
CBS	1.00	22	282	237	231 3/4	233 7/8	– 3 1/4
Compaq		16	13002	51 1/2	49 1/8	51	+ 1 1/2
Deere	2.06	148	8375	58 1/8	57 3/8	57 3/4	– 1/4
ElfAquit	1.76e	9	892	35	34 5/8	35	+ 1/2
GTE	1.82	18	14714	36	35	35 1/8	– 7/8
Heinz	1.20	16	2727	38	37 3/8	38	+ 1/8
HiLo		17	244	15 3/8	15	15	– 1/4
Hilton	1.20	20	351	45 1/8	44 3/8	44 3/4	– 1/4
Hitachi	.91e	28	67	79 1/2	79	79 1/2	– 1/2

Figure 20. Example of a Shared Scrollbar

Scroll Layout

Figure 21 shows the layout of a Scroll widget with the title at the top and the ScrollBars at the right and at the bottom:

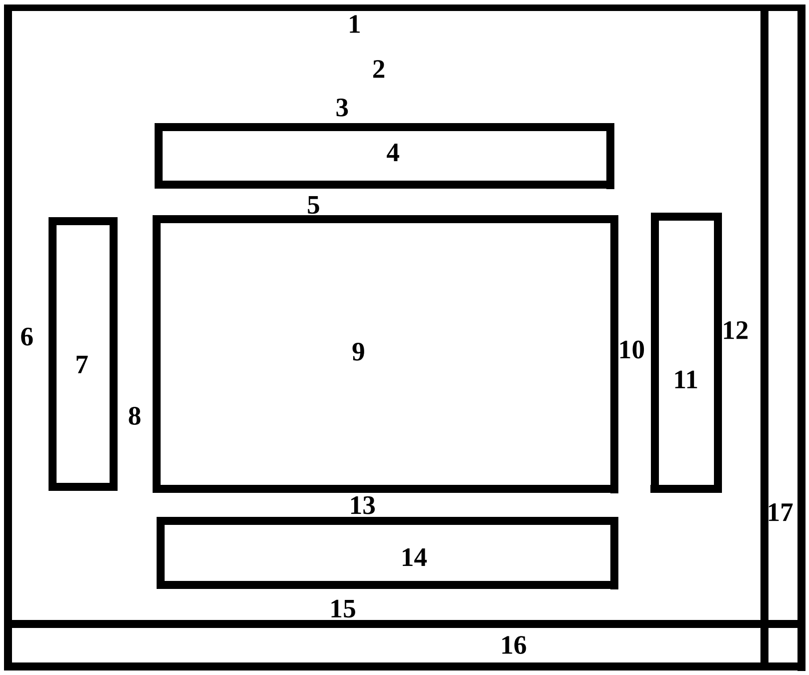


Figure 21. Scroll Layout

Figure key

The following table describes the key for Figure 21:

1	Horizontal outside margin	**10**	Vertical inside margin
2	Title label location	**11**	Right vertical annotation window
3	Title margin	**12**	Vertical outside margin
4	Top horizontal annotation window	**13**	Horizontal inside margin
5	Horizontal inside margin	**14**	Bottom horizontal annotation window
6	Vertical outside margin	**15**	Horizontal Outside Margin
7	Left vertical annotation window	**16**	Horizontal Scrollbar
8	Vertical inside margin	**17**	Vertical ScrollBar
9	Scrolled child viewing window		

Inherited Behavior and Resources

The Scroll widget inherits behavior and resources from *Core*, *Composite*, *Constraint*, and *Manager* classes.

- Class pointer is *xintScrollWidgetClass*
- Class name is *XintScroll*
- Header file is included as <Xint/Scroll.h>

Resources The following resources are defined by the XintScroll widget class.

Name	Default Type	Access
XmNaddChildrenToTabGroup	True Boolean	CG
XmNbottomAnnotationHeight	30 int	CSG
XmNhorizontalAutoSized	False Boolean	CSG
XmNhorizontalInsideMargin	10 int	CSG
XmNhorizontalOutsideMargin	10 int	CSG
XmNhorizontalScrollBar	NULL Widget	CSG
XmNleftAnnotationWidth	30 int	CSG
XmNrightAnnotationWidth	30 int	CSG
XmNscrollBarDisplayPolicy	XintAS_NEEDED int	CSG
XmNscrollBarExtend	True Boolean	CSG
XmNscrollBarPlacement	XintBOTTOM_RIGHT int	CG
XmNshadowThickness	3 Dimension	CSG
XmNshadowType	XintSHADOW_IN int	CSG

Name (continued)	Default Type	Access
XmNtitleMargin	20 int	CSG
XmNtopAnnotationHeight	30 int	CSG
XmNverticalAutoSized	False Boolean	CSG
XmNverticalInsideMargin	10 int	CSG
XmNverticalOutsideMargin	10 int	CSG
XmNverticalScrollBar	NULL Widget	CSG

XmNaddChildrenToTabGroup

Specifies whether the Scroll widget includes it children into tab groups using function XmAddTabGroup.

XmNbottomAnnotatationHeight

Specifies the height (in pixels) of the window containing the horizontal annotation displayed below the scrolled child widget. If the child of the Scroll widget has resource **XmNautoMarginAdjust** set to XintADJUST_BOTTOM, this resource will be ignored and the viewing area height will be calculated automatically.

XmNhorizontalAutoSized

When this resource is set to True, the Scroll widget will try to adjust automatically its width, so that its viewport width exactly matches the scrolled child width.

XmNhorizontalInsideMargin

Specifies the margin width (in pixels) between the scrolled child widget's viewing area and the horizontal annotation viewing areas.

XmNhorizontalOutsideMargin

Specifies the margin width (in pixels) between the Scroll widget's window border and the horizontal annotation viewing areas.

XmNhorizontalScrollBar

Specifies the widget ID of a horizontal ScrollBar to be used by the Scroll widget. If this resource is set to NULL at widget creation time, then the Scroll widget will create a new horizontal ScrollBar widget to control the horizontal scrolling of its scrolled child. This resource provides a mechanism to share a horizontal scrollbar among several Scroll widgets. Note: The value of this resource can be set after widget creation time only if the application had specified an application created scrollbar at widget creation time.

XmNleftAnnotationWidth

Specifies the width (in pixels) of the viewing area containing the vertical annotation to the left of the scrolled widget. If the child of the Scroll widget has resource **XmNautoMarginAdjust** set to XintADJUST_LEFT, this resource will be ignored and the viewing area width will be calculated automatically.

XmNrightAnnotationWidth

Specifies the width (in pixels) of the viewing area containing the vertical annotation to the right of the scrolled widget. If the child of the Scroll widget has resource **XmNautoMarginAdjust** set to XintADJUST_RIGHT, this resource will be ignored and the viewing area width will be calculated automatically.

XmNscrollBarDisplayPolicy

Controls when the horizontal and vertical ScrollBars will be visible. Specify the defined integer constant XintAS_NEEDED if you want the scroll bars to be displayed when the scrolled widget exceeds the visible viewing area in the horizontal or vertical directions. Specify XintSTATIC if scroll bars are always to be displayed.

XmNscrollBarExtend

Specifies the extent of the scroll bars. When set to True, scroll bars occupy the full length of the Scroll widget window. When set to False, scroll bars are aligned with the Scroll widget viewport.

XmNscrollBarPlacement

Specifies the position of the horizontal and vertical ScrollBars. You must use one of the following defined constants when specifying the value of this resource:

Resource Value	Description
XintBOTTOM_RIGHT (default)	Specifies that the horizontal ScrollBar will be placed below the scrolled widget and the vertical ScrollBar will be placed to the right of the scrolled widget.
XintBOTTOM_LEFT	Specifies that the horizontal ScrollBar will be placed below the scrolled widget and the vertical ScrollBar will be placed to the left of the scrolled widget.
XintTOP_LEFT	Specifies that the horizontal ScrollBar will be placed above the scrolled widget and the vertical ScrollBar will be placed to the left of the scrolled widget.
XintTOP_RIGHT	Specifies that the horizontal ScrollBar will be placed above the scrolled widget and the vertical ScrollBar will be placed to the right of the scrolled widget.

XmNshadowThickness

Specifies the shadow thickness (in pixels) for the Frames around the viewing areas for the scrolled child widget, its horizontal annotation and its vertical annotation. Usually you will specify a positive integer as the value of this resource. Alternatively, specifying 0 causes the widget to ignore the value of XmNshadowType. In this case, no shadow is drawn around the annotation areas and a line of width 1 pixel is drawn around the scrolled child.

XmNshadowType

Specifies the display characteristics of the Frames around the viewing areas for the scrolled child, its horizontal annotation and its vertical annotation. You must use one of the following defined constants when specifying the value of this resource:

Resource Value	Description
XintSHADOW_NONE	Specifies that no shadow is to be drawn around the annotation areas and a line of width **XmNshadowThickness** (pixels) will be drawn around the scrolled child.
XintSHADOW_IN (default)	Specifies that the Frames are to be drawn so that they appear to be inset
XintSHADOW_OUT	Specifies that the Frames are to be drawn so that they appear to be outset.
XintSHADOW_ETCHED_IN	Specifies that the Frames are to be drawn with a double line etched into the window.
XintSHADOW_ETCHED_OUT	Specifies that the Frames are to be drawn with a double line coming out of the window.

XmNtitleMargin

Specifies the margin height (in pixels) between the title and the horizontal annotation window. If there is no horizontal annotation window, then the title margin is the distance between the title and the scrolled child widget’s window.

XmNtopAnnotationHeight

Specifies the height (in pixels) of the window containing the horizontal annotation displayed above the scrolled child widget. If the child of the Scroll widget has resource **XmNautoMarginAdjust** set to XintADJUST_TOP, this resource will be ignored and the viewing area height will be calculated automatically.

XmNverticalAutoSized

When set to True, the Scroll widget will try to automatically adjust its height, so that its viewport height exactly matches the Scroll child height.

XmNverticalInsideMargin

Specifies the margin width (in pixels) between the scrolled child widget’s window border and the vertical annotation viewing areas.

XmNverticalOutsideMargin

Specifies margin width (in pixels) between vertical annotation’s viewing area and

Scroll widget's window border (or possibly, the vertical ScrollBar).

XmNverticalScrollBar

Specifies the widget ID of a vertical ScrollBar to be used by the Scroll widget. If this resource is set to NULL at widget creation time, the Scroll widget will create a new vertical ScrollBar widget to control the vertical scrolling of its scrolled child. This resource provides a mechanism to share a vertical scrollbar among several Scroll widgets. Note: The value of this resource can be set after widget creation time, only if the application had specified an application created scrollbar at widget creation time.

Scroll Components

The following table lists the named widgets contained within a Scroll widget which are accessible to the application programmer. The appearance and behavior of the internal widgets can be specified in a resource file using the listed widget names.

Name	Class	Description
bottom_w	XmDrawingArea	The window where the bottom horizontal annotation is drawn.
hsb	XmScrollBar	The horizontal ScrollBar.
left_w	XmDrawingArea	The window where the left vertical annotation is drawn.
right_w	XmDrawingArea	The window where the right vertical annotation is drawn.
top_w	XmDrawingArea	The window where the top horizontal annotation is drawn.
viewport	XmDrawingArea	The window where the current view of the scrolled child is drawn.
vsb	XmScrollBar	The vertical ScrollBar.

The following components are created only when the child of the Scroll widget is an EditTable widget with frozen rows or columns.

Name	Class	Description
bottom_frozen_w	XmDrawingArea	The window where the bottom horizontal annotation for frozen rows/columns is drawn.
left_frozen_w	XmDrawingArea	The window where the left vertical annotation for frozen rows/columns is drawn.
right_frozen_w	XmDrawingArea	The window where the right vertical annotation for frozen rows/columns is drawn.
top_frozen_w	XmDrawingArea	The window where the top horizontal annotation for frozen rows/columns is drawn.

Scroll Functions

The following functions are defined for creating a Scroll widget, obtaining the widget IDs of the internal named children of a Scroll widget and scrolling to the specified position.

Function Name	Description
XintCreateScroll	Creates a dialog shell containing an unmanaged Scroll widget.
XintScrollGetChild	Returns the widget ID of a named child widget of the specified Scroll widget.
XintScrollScrollToPosition	Scrolls to the specified position.

XintCreateScroll

XintCreateScroll creates an unmanaged Scroll widget.

```
Widget XintCreateScroll (...)
```

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

XintScrollGetChild

This function returns the widget ID of a specified Scroll widget component.

```
Widget XintScrollGetChild (...)
```

Widget	widget	Scroll widget ID.
int	child	Specifies named child of interest as a defined integer constant.

You must specify *child* as one of the following defined integer constants.

Defined Constant	Description
XintBOTTOM_WINDOW	Bottom horizontal annotation DrawingArea widget.
XintLEFT_WINDOW	Left vertical annotation DrawingArea widget.
XintHORIZONTAL_SCROLLBAR	Horizontal ScrollBar widget.
XintRIGHT_WINDOW	Right vertical annotation DrawingArea widget.
XintTOP_WINDOW	Top horizontal annotation DrawingArea widget.
XintVERTICAL_SCROLLBAR	Vertical ScrollBar widget.
XintVIEWPORT	DrawingArea widget containing current view of scrolled child.

If the child of the Scroll widget is an EditTable widget, you can also specify *child* as one of the following defined integer constants.

Defined Constant	Description
XintLEFT_FROZEN_WINDOW	DrawingArea widget that can contain left row annotation for frozen rows
XintRIGHT_FROZEN_WINDOW	DrawingArea widget that can contain right row annotation for frozen rows
XintTOP_FROZEN_WINDOW	DrawingArea widget that can contain top column annotation for frozen columns
XintBOTTOM_FROZEN_WINDOW	DrawingArea widget that can contain bottom column annotation for frozen columns
XintFROZEN_COLUMN_VIEWPORT	DrawingArea widget containing frozen columns.
XintFROZEN_ROW_VIEWPORT	DrawingArea widget containing frozen rows.
XintFROZEN_INTERSECTION_VIEWPORT	DrawingArea widget containing intersection of frozen rows and columns.

XintScrollScrollToPosition

Function forces the Scroll widget to scroll to the specified position, in one or both directions. The function returns False if the arguments *user_x* or *user_y* are out of range.

```
Boolean XintScrollScrollToPosition (...)
```

Widget	*widget*	The Scroll widget ID.
float	*user_x*	The horizontal coordinate where to scroll (column number if child is an EditTable).
float	user_y	The vertical coordinate where to scroll (row number if child is an EditTable).
float	vp_x_percent	Where to scroll as a percentage (0.0 at the left, 50.0 in the middle, etc.) Specify a negative value to disable scrolling in that direction.
float	vp_y_percent	Where to scroll as a percentage (0.0 at the top, 50.0 in the middle, etc.) Specify a negative value to disable scrolling in that direction.

Examples

6

Overview

This chapter contains examples for the EditTable and Scroll widgets in the INT EditTable Widget Library. We recommend that you to read sections of this chapter after you reading the corresponding reference material in *Chapter 2—CompBase Widget Metaclass* through *Chapter 5—Scroll Widget Class*. The classes defined in the library are described in alphabetical order with descriptions of object classes following the descriptions of widget classes.

Note: There are additional examples distributed with the INT EditTable Widget Library that are not documented in this chapter. Please refer to the comments inside the source code files for those examples.

This chapter includes the following sections:

How To Make and Run Example Programs

Example programs in source code form are distributed to you along with the INT EditTable Widget Library. The following examples demonstrate how simple it is to create tables with EditTable. These examples show how to implement a variety of features in a common range of applications, but these are not intended to provide an exhaustive survey of EditTable options.

Makefile

The directory containing the example programs also has a makefile that works for SPARC platforms. You may need to edit the makefile if you have a different platform. You may also need to edit the locations of the include files and the libraries in the makefile. A portion of the makefile that makes the EditTable examples is as follows:

```
INTDIR = ..
XLIBS = -lXm -lXt -lX11

CFLAGS = -I$(INTDIR) -D_NO_PROTO
LIBS = $(INTDIR)/lib/libINT.a $(XLIBS) -lm

all: edittable_1

edittable_1: edittable_1.o $(INTDIR)/lib/libINT.a
 cc -O -o edittable_1 edittable_1.o $(LIBS)
```

Running an Example Program

To run an example program after the example’s source has been made into an executable program, just type the name of the example at the operating system prompt.

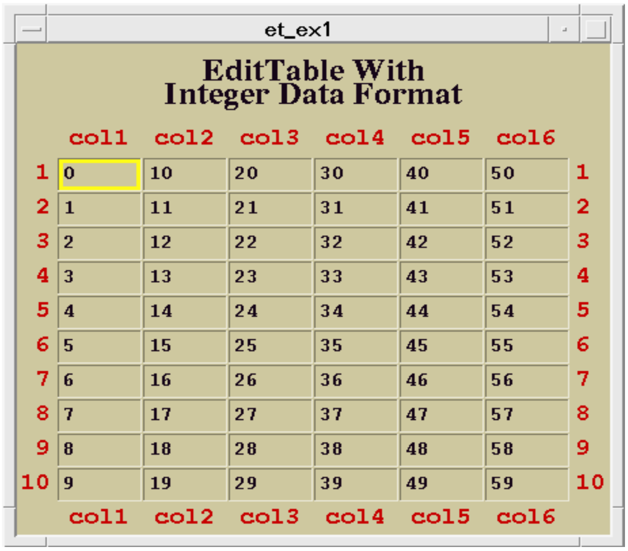
Note: Some of the example programs use data files that must be in your current directory when you execute the example program. If an example program requires a data file to run, then the name of the file is the same as the example program’s name with a file extension of `.dat`.

Example 1: Creating a Simple Table

The following example shows how to build a simple table with integer data, user-defined column headings, and automatic row annotation. as illustrated in the figure below. In particular, this example demonstrates:

- How to create the EditTable widget.
- How to set the size of the table.
- How to set the column annotation.
- How to let EditTable automatically generate row annotation.
- How margins can be calculated automatically.
- How to fill the table with data using the FillCell function.
- How to manage the table and exit to the event loop.

The following pages contain a complete example of the C code required to generate this table.



et_ex1

EditTable With Integer Data Format

	col1	col2	col3	col4	col5	col6	
1	0	10	20	30	40	50	1
2	1	11	21	31	41	51	2
3	2	12	22	32	42	52	3
4	3	13	23	33	43	53	4
5	4	14	24	34	44	54	5
6	5	15	25	35	45	55	6
7	6	16	26	36	46	56	7
8	7	17	27	37	47	57	8
9	8	18	28	38	48	58	9
10	9	19	29	39	49	59	10
	col1	col2	col3	col4	col5	col6	

Figure 22. Example of a Simple Table with Integer Data

Code The following code listing shows how to create the table shown in Figure 22.

```
/*********************************************************
 * EXAMPLE 1:  A SIMPLE TABLE USING THE EDITTABLE WIDGET *
 *********************************************************/

#include <stdio.h>
#include <Xm/Xm.h>
#include <Xint/EditTable.h>

main(argc, argv)
int argc;
char *argv[];
{
  XtAppContext app_context;
  Widget top_level;
  Widget edit_table;
  Arg arg[20];
  int col, row, count, n;
  static char *column_annotation[] = {"col1", "col2", "col3",
                      "col4", "col5", "col6"};

 /* INITIALIZE THE TOOLKIT */

  n = 0;
  top_level = XtAppInitialize (&app_context, "Example", NULL,
                        0, &argc, argv, NULL, arg, n);

/* CREATE THE EDITTABLE WIDGET UNDER THE TOP LEVEL SHELL */

  n = 0;
  XtSetArg(arg[n], XmNtitleString,
                "EditTable With\nInteger Data Format"); n++;

  /* SIZE THE TABLE */

  XtSetArg(arg[n], XmNnumberOfColumns, 6); n++;
  XtSetArg(arg[n], XmNnumberOfRows,    10); n++;

  /* SET THE COLUMN ANNOTATION */

  XtSetArg(arg[n], XmNautomaticColumnAnnotation, False); n++;
  XtSetArg(arg[n], XmNcolumnAnnotationData,
                   column_annotation); n++;
```

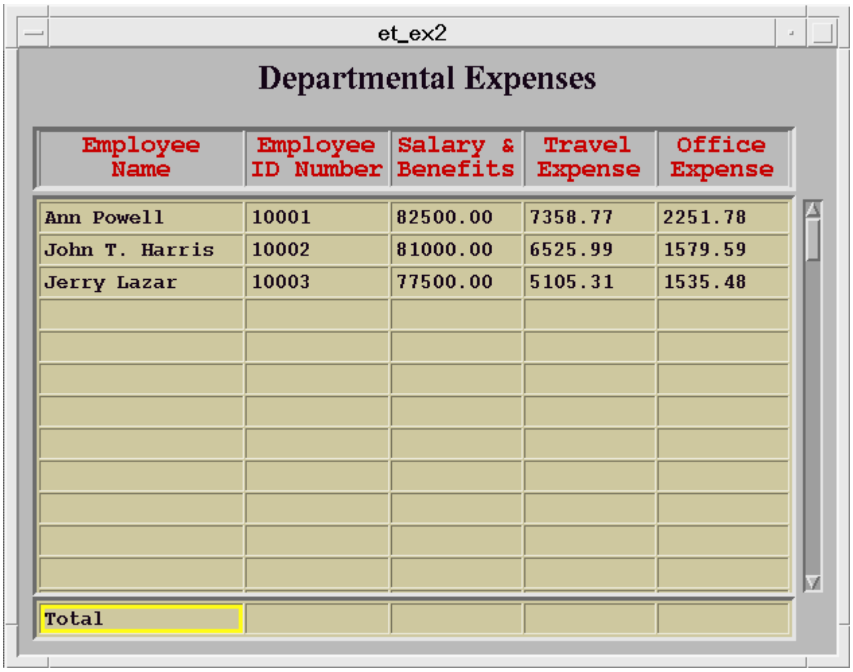
```
    /* LET EDITTABLE GENERATE THE ROW ANNOTATION */

    XtSetArg(arg[n], XmNautomaticRowAnnotation, True); n++;
   XtSetArg(arg[n], XmNgridLineStyle, XintGRID_LINE_SHADOW_IN);
n++;

    /* MARGINS WILL BE CALCULATED AUTOMATICALLY */

    XtSetArg(arg[n], XmNautoMarginAdjust, XintADJUST_ALL); n++;
    edit_table = XintCreateEditTable(top_level, "edit_table", arg, n);

   /* FILL THE TABLE WITH DATA USING THE FILLCELL FUNCTION */

    count = 0;
    for (col=1; col<=6; col++) {
      for (row=1; row<=10; row++) {
        XintEditTableFillCell(edit_table, col, row, (caddr_t) &count);
        count++;
      }
    }

   /* MANAGE THE TABLE AND GO TO THE EVENT LOOP */

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

Example 2: Scrollable Table with User Data

In the following example, we have created a table that accepts user input. and which also uses a scroll bar. In particular, this example demonstrates:

- How to initialize the toolkit and create a Scroll widget.
- How to create the EditTable widget under the Scroll widget.
- How to define the column annotation and display it on top.
- How to hide row annotation.
- How to have margins calculated automatically.
- How to define the column format and size.
- How to define a Totals column placed at the bottom.
- How to add a scroll bar.

The following page contains a complete example of the C code required to generate this table.



Departmental Expenses

Employee Name	Employee ID Number	Salary & Benefits	Travel Expense	Office Expense
Ann Powell	10001	82500.00	7358.77	2251.78
John T. Harris	10002	81000.00	6525.99	1579.59
Jerry Lazar	10003	77500.00	5105.31	1535.48
Total				

Figure 23. Example of Scrollable Table with User Data

Code

The following code listing shows how to create the table shown in Figure 23.

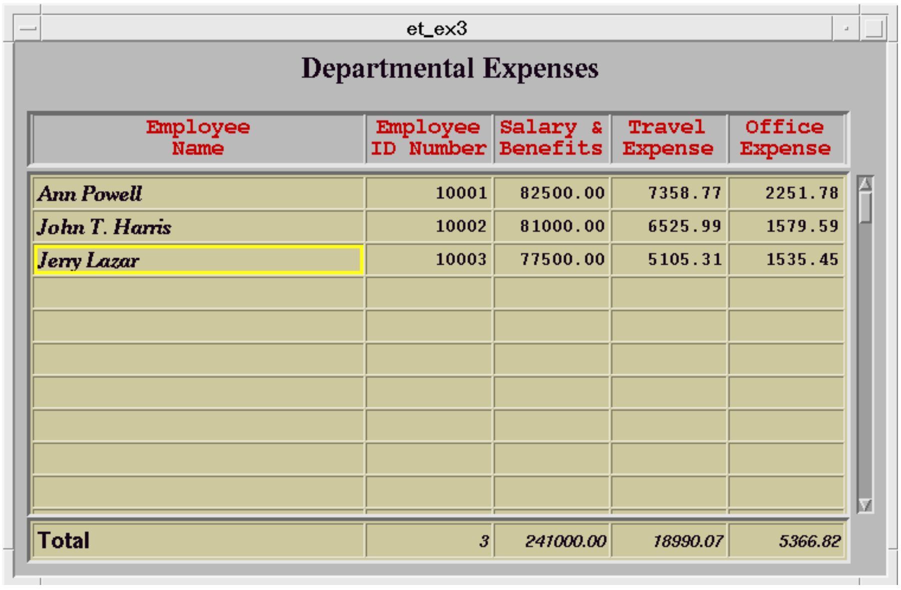
```
/*************************************************
 * EXAMPLE 2:  A SCROLLABLE TABLE WITH USER DATA *
*************************************************/
#include <stdio.h>
#include <Xm/Xm.h>
#include <Xint/EditTable.h>
#include <Xint/Scroll.h>

main(argc, argv)
int argc;
char *argv[];
{
  XtAppContext app_context;
  Widget top_level;
  Widget scroll;
  Widget edit_table;
  Arg arg[20];
  int n;
  static char *column_annotation[] = { "Employee\nName",
                    "Employee\nID Number",
                    "Salary &\nBenefits",
                    "Travel\nExpense",
                    "Office\nExpense"};
  static int column_width[] = {16, 6, 10, 10, 10};
  static char *column_format[] = {"%s", "%d", "%.2f", "%.2f", "%.2f"};
  static int column_data_type[] = {XintTYPE_STRING,
                  XintTYPE_INTEGER, XintTYPE_FLOAT,
                  XintTYPE_FLOAT, XintTYPE_FLOAT};
 /*
  * INITIALIZE THE TOOLKIT
  */
  n = 0;
  XtSetArg(arg[n], XmNallowShellResize, True); n++;
  top_level = XtAppInitialize (&app_context, "Expense Example", NULL,
                  0, &argc, argv, NULL, arg, n);
 /*
  * CREATE THE SCROLL WIDGET UNDER THE TOP LEVEL SHELL
  */
  scroll = XtVaCreateWidget("scroll",  xintScrollWidgetClass,
             top_level,
             XmNhorizontalAutoSized, True,
             XmNheight,              450,
             XmNscrollBarExtend,     False,
             XmNhorizontalInsideMargin, 3,
             NULL);
/*
 * CREATE THE EDITTABLE WIDGET UNDER THE SCROLL
 */
edit_table = XtVaCreateManagedWidget("table",
              xintEditTableWidgetClass,scroll,
             XmNtitleString,         "Departmental Expenses",
              XmNnumberOfColumns,      5,
              XmNnumberOfRows,         100,
             XmNgridLineStyle,       XintGRID_LINE_SHADOW_IN,

         /* DEFINE OUR OWN COLUMN ANNOTATION, DISPLAY IT ON TOP */
              XmNautomaticColumnAnnotation,     False,
             XmNcolumnAnnotationData,         column_annotation,
             XmNhorizontalAnnotationPlacement, XintPLACEMENT_TOP,

           /* NO ROW ANNOTATION DISPLAYED */
              XmNautomaticRowAnnotation,        False,
             XmNverticalAnnotationPlacement,   XintPLACEMENT_NONE,

           /* MARGINS WILL BE CALCULATED AUTOMATICALLY */
             XmNautoMarginAdjust,              XintADJUST_ALL,
              XmNshowAnnotationGridLines, True,

           /* DEFINE THE COLUMNS FORMAT AND SIZE */
             XmNcolumnDataTypeData,           column_data_type,
             XmNcolumnDataFormatData,          column_format,
             XmNcolumnNumCharData,              column_width,

    /* FIXED TOTAL COLUMN PLACED AT THE BOTTOM */
    XmNfrozenRowPlacement,            XintPLACEMENT_BOTTOM,
    NULL);

/*
 * FREEZE THE LAST ROW
 */
 XintEditTableFreezeRow(edit_table, 100);

/*
 * SET STRING TOTAL
 */
 XintEditTableFillCell(edit_table, 1, 100, (caddr_t) "Total");

/*
/*
 * MANAGE THE SCROLL AND GO TO THE EVENT LOOP
 */
 XtManageChild(scroll);

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

Example 3: Change Fonts and Add Totals

At first glance, the following table is nearly identical to the previous one. However, we have added new fonts to some of the cells, and we have built in the callbacks for mathematical calculations so that the Totals row automatically calculates and validates the figures in each column. In particular, this example demonstrates the following:

- How to define and install a font table.
- How to adjust column alignments
- How to calculate and validate totals.
- How to update cells with calculated values.
- How to apply various fonts to selected table cells.

The coding examples on the following page show only the portions of code that were added to the previous example.



Departmental Expenses

Employee Name	Employee ID Number	Salary & Benefits	Travel Expense	Office Expense
Ann Powell	10001	82500.00	7358.77	2251.78
John T. Harris	10002	81000.00	6525.99	1579.59
Jerry Lazar	10003	77500.00	5105.31	1535.45
Total	3	241000.00	18990.07	5366.82

Figure 24. Example of Changing Fonts and Add Totals

Global Definitions

```
/*
 * FONT TABLE DEFINITION
 */
static void NumEmployee();
static void SumUp();

static char *font_table[] = {
  "*Times-bold-*18*",
  "*helvetica-bold-*14*",
  "*helvetica-bold-r*18*",
  NULL /* termination */
};

/*
 * ADJUST COLUMN ALIGNMENTS
 */
  static int column_alignment[] = {
XintALIGNMENT_BEGINNING_MIDDLE,
                          XintALIGNMENT_END_MIDDLE,
                          XintALIGNMENT_END_MIDDLE,
                          XintALIGNMENT_END_MIDDLE,
                          XintALIGNMENT_END_MIDDLE};
```

Additional Resources

The following additional resources are set when creating the EditTable widget.

```
/*
 * INSTALL FONT TABLE
 */
           XmNfontTable,      font_table,

/*
 *FIXED TOTAL COLUMN PLACED AT THE BOTTOM
 */
           XmNfrozenRowPlacement,   XintPLACEMENT_BOTTOM,
           NULL);
```

Additional Callbacks

The following callbacks are registered after the EditTable widget is created.

```
/*
 * ADD VALIDATION FOR NUMBER OF EMPLOYEES
 */
 XintEditTableAddLocalCallback(edit_table, 2, 0,
              XmNvalidateValueCallback,
              NumEmployee, NULL, XintLOCAL_CALLBACK_AFTER);

/*
 * ADD VALIDATION CALLBACKS TO LAST THREE
 * COLUMNS SO THEY DO SOMETHING ON USER INPUT
 */
 for (n = 3; n <= 5; n++) {
   XintEditTableAddLocalCallback(edit_table, n, 0,
              XmNvalidateValueCallback,
              SumUp, NULL, XintLOCAL_CALLBACK_AFTER);
 }
```

Set Table Font

The following code assigns the various fonts used in the example.

```
/*
 * USE HELVATICA-BOLD-14POINT FONT (SEE 'FONT_TABLE' ABOVE)
 * FOR THE TOTAL ROW
 */
 XintEditTableSetRowFont(edit_table, 100, 1, 1);
/*
 * USE TIMES-BOLD-18POINT FOR THE 'EMPLOYEE NAMES'
 */
 XintEditTableSetColumnFont(edit_table, 1, 1, 0);

/*
 * USE HELVATICA-BOLD-18POINT FONT FOR THE FIELD 'TOTAL'
 */
 XintEditTableSetCellFont(edit_table, 1, 1, 100, 1, 2);
```

Number of Employees Callback

The following callbacks counts the number of employees.

```
/*
 * PERFORM CALCULATIONS
 */
static void
NumEmployee(widget, unused, cb)
Widget widget;
caddr_t unused;
XintEditTableValidateValueCallbackStruct *cb;
{
  int total = 0;
  int *data;
  int i, nrows;

  data = (int *) XintEditTableGetColumnData(widget,
                               cb->column, &nrows);

  /* IF THE NEW VALUE OF THE CURRENT CELL IS NULL */

  if (cb->new_value_string && cb->new_value_string[0] != '\0')
{
    if (sscanf(cb->new_value_string, "%d", &i) == 1) {
      total = 1;
    } else {
      cb->cell_value.integer_value = XintUNDEFINED_INTEGER;
      total = 0;
    }
  } else {
    cb->cell_value.integer_value = XintUNDEFINED_INTEGER;
    total = 0;
  }

  for (i = 0; i < nrows - 1; i++) {
    if (i + 1 != cb->row && data[i] != XintUNDEFINED_INTEGER) total++;
  }
  if (total == 0) total = XintUNDEFINED_INTEGER;

  /* UPDATE THE NUMBER OF EMPLOYEES */

  XintEditTableFillCell(widget, cb->column, nrows, (caddr_t) &total);

  /* CLEAN UP */

  XtFree(data);
}
```

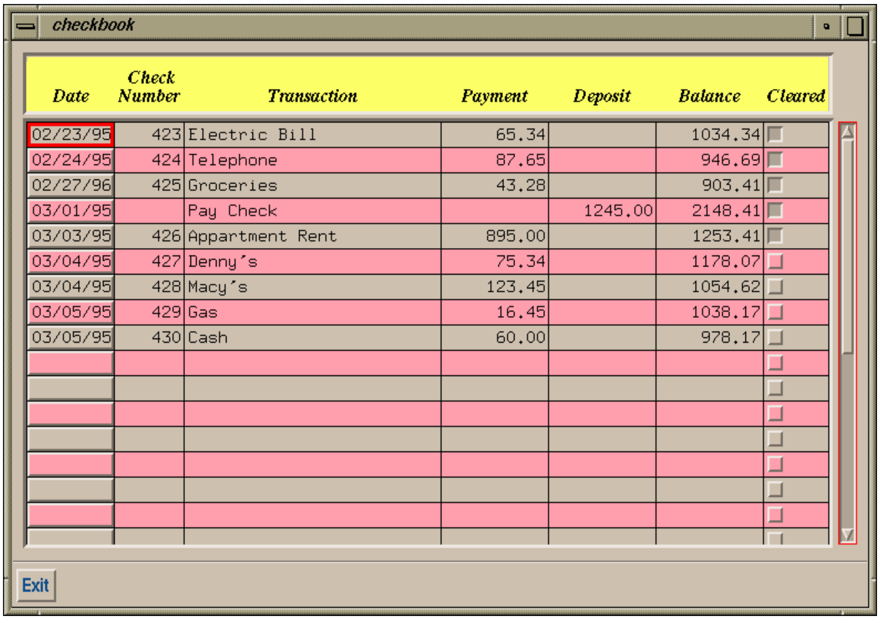
Summation Callback

The following callback is for summing up the columns of numbers

```
static void
SumUp(widget, unused, cb)
Widget widget;
caddr_t unused;
XintEditTableValidateValueCallbackStruct *cb;
{
  float total = 0;
  float *data;
  int i, nrows;

  data = (float *) XintEditTableGetColumnData(widget,
                                cb->column, &nrows);

  /* RETRIEVE THE NEW VALUE OF THE CURRENT CELL */

  if (cb->new_value_string) {
    if (sscanf(cb->new_value_string, “%f”, &total) != 1)
      cb->cell_value.float_value = XintUNDEFINED_FLOAT;
  } else
      cb->cell_value.float_value = XintUNDEFINED_FLOAT;

  /* ADD VALUES OF ALL CELLS EXCEPT CURRENT AND LAST CELL */
  for (i = 0; i < nrows - 1; i++) {
    if (i + 1 != cb->row && data[i] != XintUNDEFINED_FLOAT)
        total += data[i];
  }

  /* UPDATE THE TOTAL */
  XintEditTableFillCell(widget, cb->column, nrows, (caddr_t) &total);

  /* CLEAN UP */
  XtFree(data);
}
```

Example 4: Widget In A Cell

The following example illustrates how to include a simple widget and a more complicated widget in sets of cells in an edit table. The complete source code for the example will be found in checkbook.c in the examples directory.



Date	Check Number	Transaction	Payment	Deposit	Balance	Cleared
02/23/95	423	Electric Bill	65.34		1034.34	▨
02/24/95	424	Telephone	87.65		946.69	▨
02/27/96	425	Groceries	43.28		903.41	▨
03/01/95		Pay Check		1245.00	2148.41	▨
03/03/95	426	Appartment Rent	895.00		1253.41	▨
03/04/95	427	Denny's	75.34		1178.07	☐
03/04/95	428	Macy's	123.45		1054.62	☐
03/05/95	429	Gas	16.45		1038.17	☐
03/05/95	430	Cash	60.00		978.17	☐
						☐
						☐
						☐
						☐
						☐
						☐
						☐
						☐

Figure 25. Example of a Widget In A Cell Application

Inserting a PushButton Widget

The following code inserts PushButton widgets in column one. We set XmNcellWidgetSetResources to True. This causes button color, label, alignment and sensitivity to be set automatically. Thus, we do not need to register callback XmNcellWidgetDisplayCallback.

```
/*
 * Create a PushButton widget that will be used for column 1.
 */
 range.row = 1;
 range.rows = 0;
 range.column = 1;
 range.columns = 1;
 XtVaCreateWidget(“ “, xmPushButtonWidgetClass, Table,
                         XmNcellWidgetRange, &range,
                         XmNcellWidgetSetResources, True,
                      XmNcellWidgetOverrideTranslations, True,
                         NULL);
```

Note: The name of the PushButton is set to a blank (“ “) so that it appears to be empty in all of the buttons which have no value assigned.

Inserting a ToggleButton Widget

The following code creates ToggleButton widgets in column seven. Since the state of the ToggleButton is not set automatically, we register callback XmNcellWidgetDisplayCallback. We also must update the table whenever the ToggleButton is activated. This is done by registering callback XmNvalueChangedCallback on the ToggleButton.

```
/*
 * Add the cell widget callback to EditTable
 */
  XtAddCallback(Table, XmNcellWidgetDisplayCallback,
               (XtCallbackProc) CellWidgetCallback, (XtPointer)
                   NULL);

 /*
  * Create a ToggleButton widget that will be used for column 7.
  */
  range.row = 1;
  range.rows = 0;
  range.column = 7;
  range.columns = 1;
  toggle = XtVaCreateWidget(“ “, xmToggleButtonWidgetClass, Table,
                          XmNcellWidgetRange, &range,
                          XmNcellWidgetSetResources, False,
                        XmNcellWidgetOverrideTranslations, True,
                          NULL);

/*
 * Add the value changed callback to the “toggle” widget.
 */
  XtAddCallback(toggle, XmNvalueChangedCallback, ToggleCallback,
                Table);
```

Updating Resources

The following code illustrates a callback to update resources when the ToggleButton widget is drawn to a particular cell.

```
static void CellWidgetCallback(widget, data, cb)

  Widget widget;
  XtPointer data;
  XintEditTableCellWidgetCallbackStruct *cb;

{
  int *state_ptr;
  int state;

  if (XmIsToggleButton(cb->widget)) {

    state_ptr = (int *) XintEditTableGetCellData(widget, cb->column,
                   cb->row);

    if (state_ptr)
      state = *state_ptr;
    else
      state = False;

    XtVaSetValues(cb->widget,
                  XmNset, state,
                  XmNbackground, cb->resources->background,
                  NULL);
  }
}
```

Updating the Edit Table

The following callback updates the edit table when the ToggleButton is “toggled”.

```
static void ToggleCallback(widget, table, cb)

  Widget widget;
  Widget table;
  XmToggleButtonCallbackStruct *cb;

{
  int column, row;
  int value = cb->set;

 /*
  * Find the location of the toggle and update the table.
  */
  XintEditTableGetCellPointerPosition(table, &column, &row);
  XintEditTableFillCellNoUpdate(table, column, row, &value);
}
```

Index

Index

Index

Defined Constants

Index

Functions

Index

Index

Resources (continued)

Subjects

Index

Index

Subjects (continued)

Index