Carnegie Mellon University
Software Engineering Institute

Serpent

Slang Reference Manual

User's Guide

CMU/SEI-91-UG-5
January 2008

Serpent: Slang Reference Manual

User Interface Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

The ideas and findings should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

JOHN S. HERMAN, Capt, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

The Software Engineering Institute is not responsible for any errors contained in these files or in their printed versions, nor for any problems incurred by subsequent versions of this documentation.

Table of Contents

List of Examples

List of Figures

List of Tables

1 Introduction

Serpent is a user interface management system (UIMS) that supports the development and execution of the user interface of a software system from the prototyping phase through production and maintenance. Serpent encourages a separation of functionality between the user interface and application portion of a system. Serpent is also easily extended to support additional input/output (I/O) toolkits.

This manual describes the model, syntax, and semantics of the Slang dialogue language, the language within Serpent used for the specification of user interfaces.

1.1 This Manual

This manual serves two purposes: to provide an introduction to Slang and to provide a reference manual for Slang. Readers should be familiar with general UIMS concepts, have a working knowledge of programming languages, and understand the concepts described in *Serpent Overview* and *Serpent: System Guide*.

1.1.1 Organization

The first three chapters provide an introduction to Slang. The remainder of the manual provides syntactic and semantic information about Slang.

1.1.2 Typographical Conventions

The following conventions are observed in this manual.

Code examples	`Courier typeface`
Variables, attributes, etc.	`Courier typeface`
Syntax	`Courier typeface`
Warnings and Cautions	***Bold, italics***

1.2 Documentation

This reference manual describes the model, syntax, and semantics of the Slang dialogue language. The following publications address other aspects of Serpent.

Serpent Overview
Introduces the Serpent system.

Serpent: System Guide
Describes installation procedures, specific input/output file descriptions for intermediate sites and other information necessary to set up a Serpent application.

Serpent: Saddle User's Guide
Describes the language that is used to specify interfaces between an application and Serpent.

Serpent: Dialogue Editor User's Guide
Describes how to use the editor to develop and maintain a dialogue.

Serpent: C Application Developer's Guide
Serpent: Ada Application Developer's Guide
Describe how the application interacts with Serpent. These guides describe the runtime interface library, which includes routines that manage such functions as timing, notification of actions, and identification of specific instances of the data.

Serpent: Guide to Adding Toolkits
Describes how to add user interface toolkits such as various Xt-based widget sets to Serpent or to an existing Serpent application. Currently, Serpent includes bindings to the Athena Widget Set and the Motif Widget Set.

The following figure shows Serpent documentation in relation to the Serpent system:

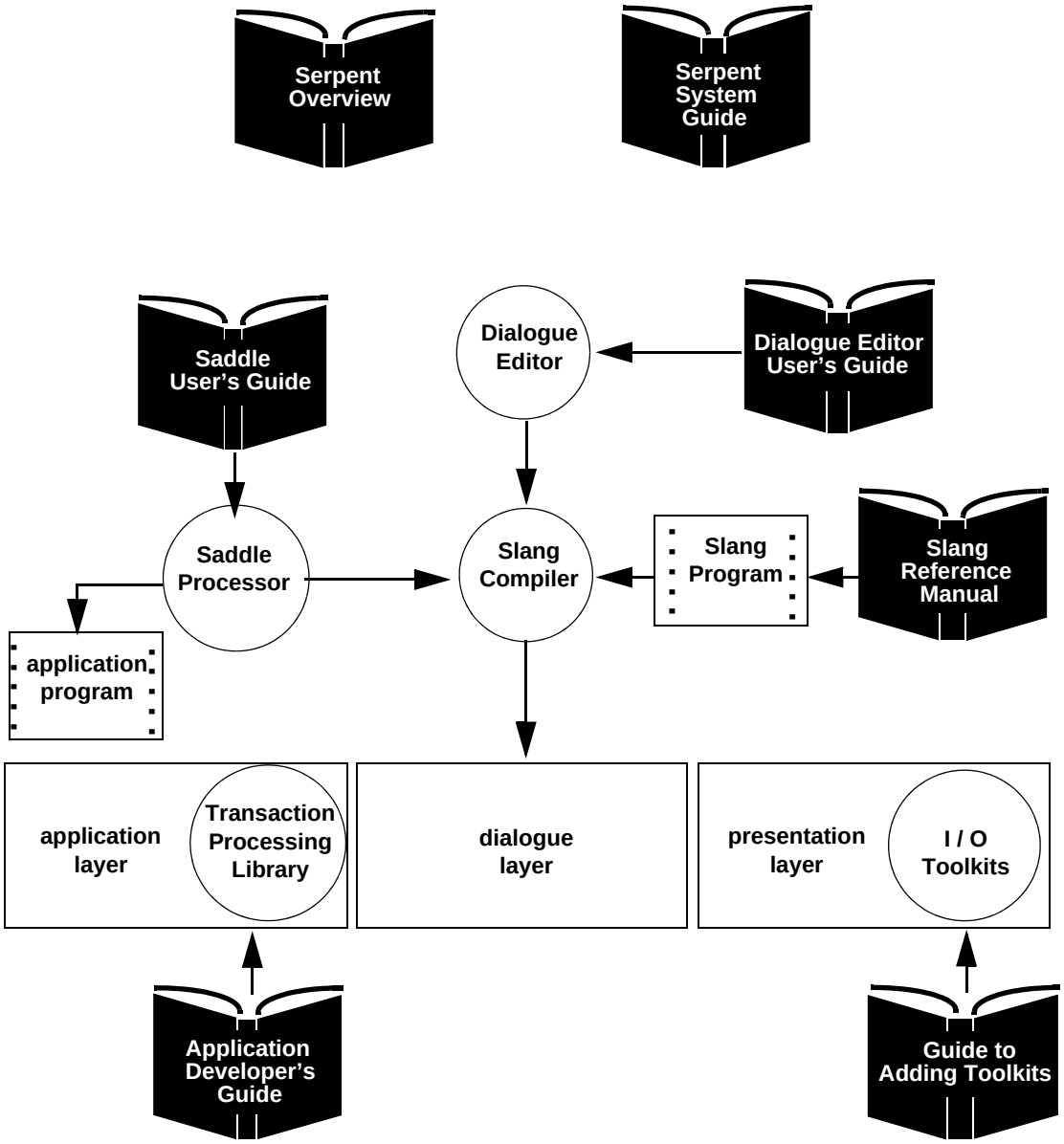

Figure 1-1 Serpent Documents

2 Serpent Overview

Serpent manages the total dynamic behavior of an interface and allows an application to be separated from the details of the user interface. Serpent is designed to manage relatively arbitrary toolkits. There is a language to describe the interface and an editor to build it. Serpent provides a runtime system that enables communication between the application and the end user.

Serpent supports the incremental development of the user interface from prototyping through production and maintenance. Serpent can be used either with an application in a production environment or without an application in a prototyping environment.

A primary goal of Serpent is to encourage the separation of a software system into an application portion and a user interface portion. This provides the application developer with a fixed application programmer's interface. One benefit of a fixed application programmer's interface is that the application programmer is insulated from the modifications to the user interface that are the most likely modifications to a completed system. Another benefit of using Serpent is that it provides the tools to develop and modify the user interface. A system developed using Serpent can be migrated to new toolkits without time-consuming reengineering of the application portion.

2.1 Serpent Architecture

Serpent is implemented using the standard UIMS architecture defined by the Seeheim working group on graphical interfaces (see *User Interface Management Systems*, G. E. Pfaff, ed, Eurographics Seminars, Springer-Verlag, 1985). The architecture consists of three layers: presentation, dialogue, and application (Figure 2-1).

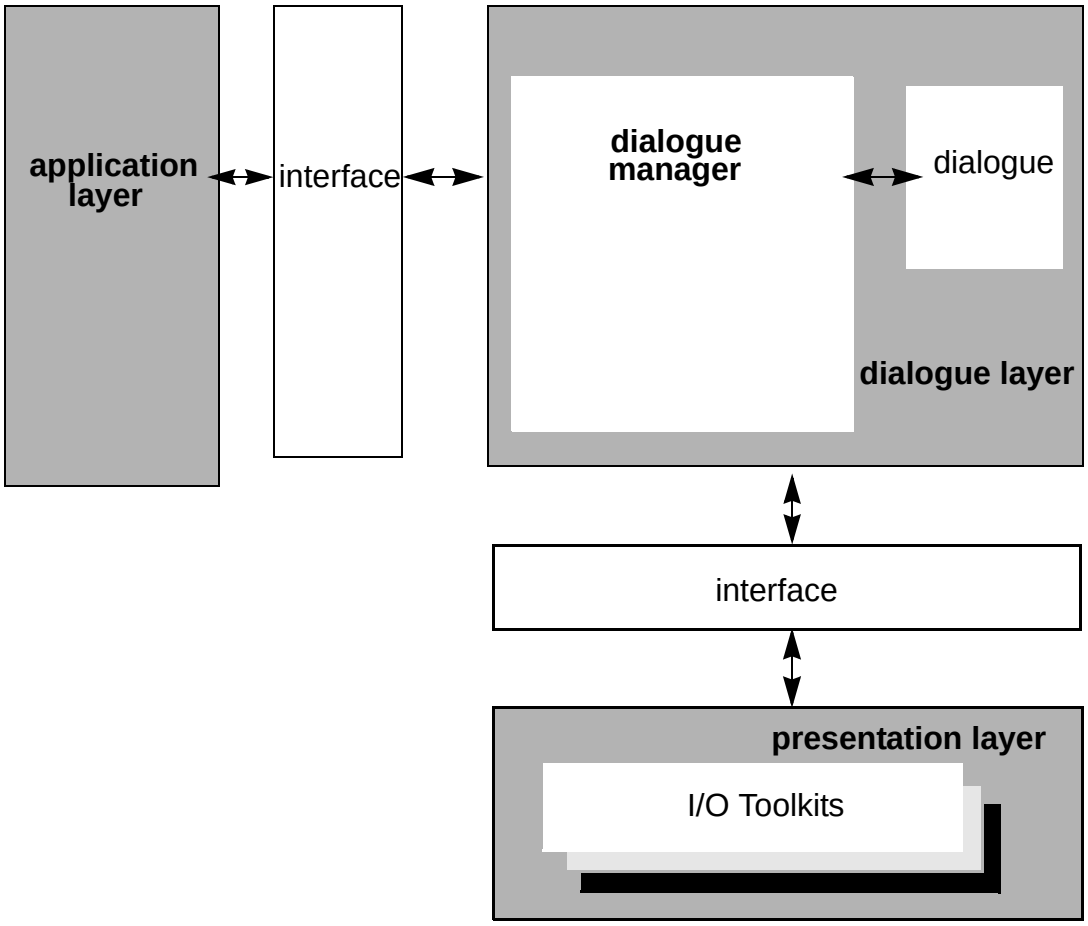

Figure 2-1 Serpent Architecture

The architecture is intended to encourage the proper separation of functionality between the application and the user interface portions of a software system. The three different layers of the standard architecture provide differing levels of control over user input and system output. The presentation layer is responsible for layout and device issues. The dialogue layer specifies the presentation of application information and user interactions. The application layer provides the functionality for the system.

The presentation layer consists of toolkits that have been incorporated into Serpent. Toolkits are existing hardware/software systems that perform some level of generalized interaction with the end user. Serpent is distributed with an interface to the X Window System (Version 11), to the MIT Athena Widget Set, and to the OSF/Motif Toolkit. Other toolkits based on the X Window System Intrinsics can be easily integrated into Serpent, and toolkits not based on the X Window System Intrinsics can be integrated without undue difficulty. Refer to the *Serpent: Guide to Adding Toolkits* for a discussion of how to do this.

The end-user interface for a software system is formally specified as a *dialogue* in Serpent. The dialogue specifies both the presentation of application information and end-user interactions.

The application provides the functional portion of the software system in a presentation-independent manner. It may be developed in C, Ada, or other programming languages (only C and Ada are supported at this time). The application may be either a functional simulation for prototyping purposes or the actual application for a delivered system. The actions of the application layer are based on knowledge of the specific problem domain. *Serpent: C Application Developer's Guide* and *Serpent: Ada Application Developer's Guide* describe how an application interacts with Serpent.

One way of viewing the three levels of the architecture is by the level of functionality provided for user input. The presentation layer is responsible for lexical functionality, the dialogue layer for syntactic functionality, and the application layer for semantic functionality. For example, in processing the selection of a menu item, the presentation layer is responsible for determining which menu item was selected and for presenting some indication to the end user of which item is currently selected. The dialogue layer is responsible for deciding whether another item is to be selected and for presenting it to the end user, or whether the choice requires action by the application. The application layer is responsible for executing those functions specific to the application. In another example, in processing a change in the status of the application, the application would detect the change in status and inform the dialogue layer of the new status. The dialogue layer would decide that the change in status requires a form to appear on the display, and the presentation layer would actually make the form visible to the end user.

2.2 Shared Database

Serpent provides an active database model for communication among the application, the dialogue, and the toolkits. In an active database, multiple processes are allowed to update a single database. Changes are then propagated to each user. This model is implemented in Serpent by a *shared database* that logically exists between the application and toolkits. The application can add, modify, query, or remove data from the shared database. Information provided to Serpent by the application is available for presentation to the end user. The application has no knowledge of the presentation media or user interface styles used to present this information.

The application and the Slang program exchange data through the shared database. Figure 2-2 illustrates the use of the shared database in Serpent. When the application modifies data in the shared database, the portions of the Slang program that depend upon that data are automatically executed. When the Slang program modifies data in the application portion of the shared database, the application is notified. The section of the shared database that is associated with the application layer, *application shared data*, is accessible only from the application layer and the dialogue layer.

The presentation layer also communicates with the dialogue layer via shared data. The section of the shared database that is associated with the presentation layer, toolkit shared data, is accessible only from the presentation layer and the dialogue layer.

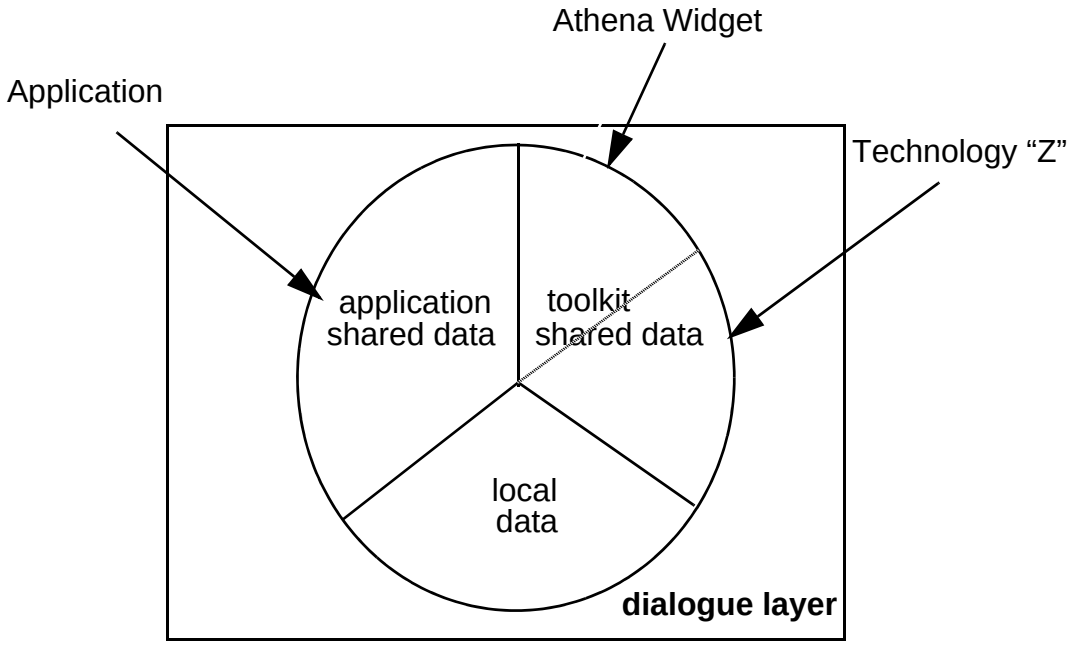

Figure 2-2 Serpent Shared Database

Serpent allows the specification of dependencies between elements in the shared database. These dependencies define a mapping between application data, presentation objects, and end-user input. The dialogue manager enforces the constraints implied by these dependencies by operating on the information stored in the shared database until the constraints are satisfied. Changes are then propagated to either the application or the toolkits as appropriate.

The type and structure of information that can be maintained in the shared database is defined externally in a *shared data definition* file. The structure in the shared data definition file corresponds to the database concept of schemata. A shared data definition file is required for each application*; Serpent: Saddle User's Guide* describes how to construct this file.

2.3 Serpent Components

Serpent consists of the following components:

- A language designed for the specification of user interfaces (Slang). This language is compiled into the dialogue layer of the Serpent architecture.

- A database-like schema language (Saddle) to define the interface between the application and Serpent. Applications and toolkits written in either C or Ada can be used with Serpent, although the interface description mechanism is designed to be extensible to other languages.
- A transaction processing library that is linked to the application layer to provide access to Serpent.
- An interactive editor for the specification of dialogues and for the construction and preview of displays.
- Input/output toolkits. The use of Serpent depends upon the types of toolkits that have been integrated.

3 Slang Overview

Serpent dialogues are specified in Slang. These dialogues define the presentation of application information to, as well as interactions with, the end user.

This chapter describes the overall dialogue model used by Serpent. The dialogue model provides the conceptual basis for dialogue specification. This model is based largely on the data-driven, rule-based approach used in production systems.

3.1 Interaction Objects

Interaction objects, which are defined by a given toolkit, allow user interaction with the application. Each interaction object has a particular appearance and behavior. The behavior determines how the object responds to end-user input. In a display technology, for example, text fields, circles, or rectangles may be defined as interaction objects. In voice technology, voice messages or recordings may be defined as interaction objects.

Interaction objects are specified by the designer in a dialogue and presented to the end user by the presentation layer. The objects used as examples in this document are based on Athena widgets. The Athena Widget Set is one of the initial toolkits supported by Serpent. A description of the supported Athena widget objects is provided in Appendix G. *A* description the OSF/Motif widgets is provided in Appendix H.

A dialogue is defined by a Slang program, and each Slang program enumerates a collection of objects to be available to the end user. Examples of interaction objects defined for the Athena Widget Toolkit are command widgets and text widgets.

Each object type has a collection of attributes that defines its presentation, as well as methods that determine the high-level interactions that the end user can have with the object. Objects are specified in a Slang program by listing the objects to be created and the attribute values to be assigned to each occurrence of the object. See Chapter 8 for details.

3.1.1 Attributes

Interaction objects have attributes to define the presentation characteristics. These attributes are defined by the toolkit integrator and their values are specified by the dialogue developer.

Example 3-1 provides the Slang specification for a label widget. A label widget is an Athena widget that provides a non-selectable rectangular label. Once defined, the interaction object is displayed to the end user using the toolkit that supports this object. The object defined in Example 3-1 causes a box (label widget) containing the value 0 to be displayed as illustrated in Figure 3-1.

```
display: XawLabel {
 Attributes:
    parent: main_background;
    height: 40;
    width: 60;
    label: 0;
    font: "9x15bold";
    vertDistance: 100;
    horizDistance: 175;
}
```

Example 3-1 Attributes

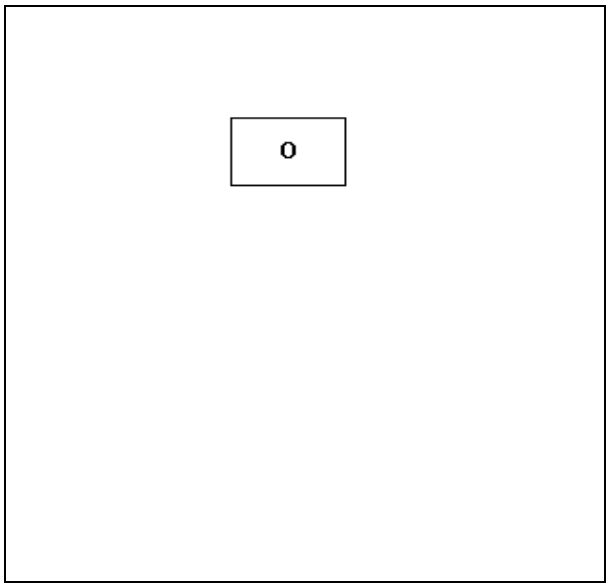

Figure 3-1 Label Widget

The label widget is generated as an independent window that the end user can position.

3.1.2 Methods

Interaction objects also have *methods* that are defined by the toolkit. Methods provide a way for handling end-user interactions in the dialogue by specifying actions to be performed for specific, end-user generated events. For example, the interaction object in Example 3-2 is declared as a command widget. A command widget is an Athena widget object that provides a display button that may be selected by the end user. The Athena Widget toolkit generates a `notify` event for the object when the command widget on the display is selected.

```
push_button: XawCommand {
 Attributes:
    parent: main_background;
    height: 40;
    width: 60;
    vertDistance: 200;
    horizDistance: 175;
    font: "9x15bold";
    label: "Push";
 Methods:
    notify: {
      display.label:= display.label + 1;
 }
}
```

Example 3-2 Methods

The button object defined in Example 3-2 causes a command widget to be displayed that contains the value `Press`. When selected, the command widget will generate a notify event that causes the `notify` method in the button definition to be executed. In this case, execution of the method causes the `text` attribute of the display object defined to be incremental. Figure 3-2 illustrates the display of both objects after the command widget has been pressed.

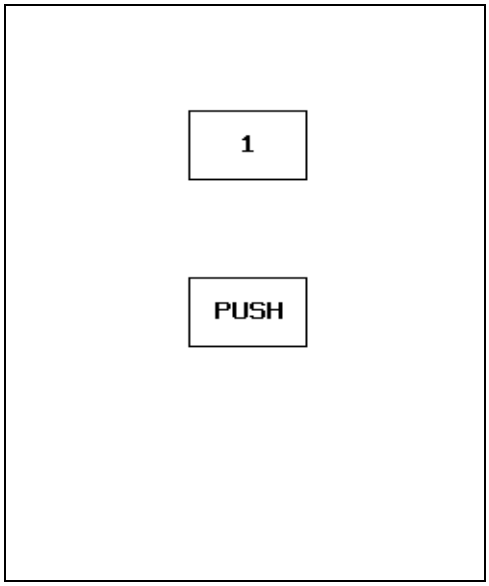

Figure 3-2 Command Widget Display

It is interesting to note that in the preceding example the `label` attribute of the `XawLabel` is actually defined as a string, while the numeric constant `1` is an integer. All type conversion is handled automatically by Slang.

3.2 Variables

Slang also allows for the specification of dialogue variables. Variables can be used for defining object attributes and for defining control flow as described later in this chapter. The preceding example can be rewritten to make use of local variables.

```
Variables:
 counter: 0;
Objects:
display: XawLabel {
 Attributes:
    parent: main_background;
    height: 40;
```

```
    width: 60;
    vertDistance: 100;
    horizDistance: 175;
    font: "9x15bold";
    label: counter;
}
button: XawCommand {
 Attributes:
    parent: main_background;
    height: 25;
    width:   25;
    vertDistance: 200;
    horizDistance: 175;
    font: "9x15bold";
    label: "Push";
 Methods:
    notify: {
      counter := counter + 1;
    }
}
```

Example 3-3 Dialogue Variables

Note in the example that Slang supports dynamic typing. That is, the variable `counter` is treated as an integer in the initial assignment and in the method, but is converted to a string for the assignment to `label`. Serpent automatically decides upon the appropriate type for an operation or an assignment and changes the type of a value as necessary.

Note, also, that the `label` attribute of the `display` object is dependent on the `counter` variable. Whenever `counter` is modified (in this case, as the result of a `notify` event), the `label` attribute is automatically updated by Serpent. In this manner, local variables can be used to express complex relationships between interaction object attributes.

The dependency concept is fundamental to the structure of Slang programs. Flow of information between portions of a Slang program is handled automatically based on the names used in variables and attributes. Thus, in Example 3-3, no explicit action is required by the user to communicate to Serpent the information that the `label` attribute of the `display` object depends on `counter` and that `label` should be recomputed whenever `counter` changes.

3.3 Data Dependencies

An important and powerful aspect of Slang is that it automatically supports dependencies between data items in the program. Certain data items, such as variables and interaction object attributes, may be dependent on other variable data within the dialogue. Whenever the data upon which these data items are dependent is modified, the data items are automatically updated.

All variable declarations and object attribute definitions are automatically a portion of the dependency system. Whenever a data item on the right-hand side of a variable declaration or an attribute definition is modified, the declaration or definition is recalculated. Consequently, new values are assigned to items on the right-hand side of the computation. Computations performed either within methods or within creation and deletion actions are not recalculated when an independent value is modified.

The automatic recalculation has the potential to introduce an infinite loop into a program. That is, if variable `a` is defined in terms of variable `b` and variable `b` is defined in terms of variable `a`, then an infinite loop within a program is created and the program will not execute as expected.

Data dependencies are determined for the total calculation of a variable or an attribute. That is, if the computation of an attribute is a complex statement, then the total computation is redone as a result of the triggering of a data dependency.

3.4 View Controllers

Interaction objects, their methods and attributes, and dialogue variables provide a great deal of power in defining relatively static interfaces, or interfaces that contain a fixed set of interaction objects for which only the attributes may change. Although this is a very powerful mechanism, it is not adequate to describe more complex interfaces where it is necessary to present and remove interaction objects as a collection from the display.

The mechanism by which objects are collected is a *view controller.* A view controller is used to group interaction objects and to control the circumstances under which they are presented to the end user. Interaction objects and local dialogue variables are defined within the context of view controllers. A view controller maps specific data in the application into objects on the display and controls the existence of these objects.

A dialogue is specified in terms of *view controller templates*. A template maintains a watch on application shared data or local dialogue data for specific conditions. When data that satisfies a view controller template is placed into application shared data or when local dialogue data attains certain values, a view controller is created.

In a Slang program, a view controller template is specified that has a collection of objects associated with it. When a view controller is instantiated from the template, the associated objects are created and communicated to the presentation layer. The presentation layer makes the objects visible to the end user.

A simple example of this is a *drop-down menu*. A drop-down menu consists of a menu bar that contains a number of options. Selecting an option causes a submenu to appear directly below the menu bar. Figure 3-3 illustrates a sample drop-down menu.

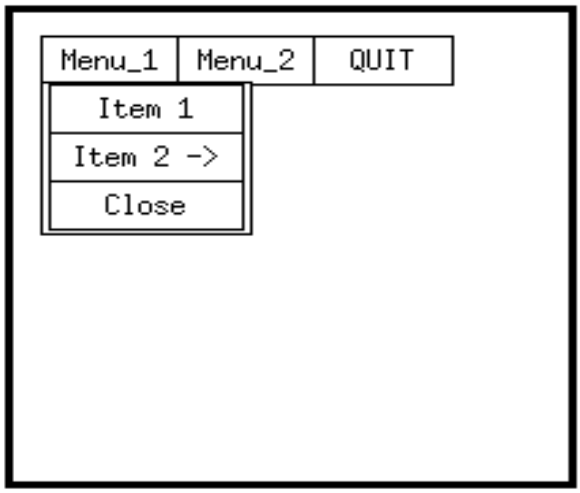

Figure 3-3 Drop-Down Menu

The drop-down menu can be implemented by using command widget interaction objects. Each menu item on the menu bar is represented by a command widget object. When a menu item is selected, additional command widget objects are displayed in order to produce the submenu.

View controllers have *creation conditions* that define the condition under which the view controller is instantiated. When a view controller is instantiated, the interaction objects defined for that view controller are displayed to the end user.

The Slang dialogue segment shown in Example 3-4 defines several interaction objects that make up the menu bar along with two dialogue variables.

The four objects that make up the menu bar are the: `menu_bar_form`, `Menu_1`, `Menu_2`, and `Quit` menu items. The `menu_bar_form` is a form widget that specifies the relative positions of objects within the form. The `Menu_1`, `Menu_2`, and `Quit` are command widgets that provide selectable menu headings for the menu bar. Besides these interaction objects, there are also three local variables defined: `display_menu1_submenu`, `display_menu2_submenu` and `display_sub_item_submenu`. These variables are initialized to `FALSE` and set to `TRUE` when their respective command buttons are selected.

```
#include "sat.ill"
|||
/*
** This demonstrates the use of Slang to produce
** a menu bar with two tear off menus.
** Initially, there is a menu bar presented to
** the user with two options: Menu_1 and Menu_2.
** Only Menu_1 is active. When the user selects
** Menu_1, a pull down menu will be displayed
** with additional items. When the user selects
** "Item 2 ->" another menu will be displayed.
** Each pull down has its own "Close" button and
** only affects that pull down menu. When the
** user selects the "Close" from the first pull
** down menu, the other pull down menu will
** remain on the display.
*/

Variables:

 display_menu1_submenu : FALSE;
 display_menu2_submenu : FALSE;
 display_sub_item_submenu : FALSE;

Objects:

menu_bar_form: XawBboard {
 Attributes:
    height:250;
    width: 250;
}

menu_bar: XawBboard {
 Attributes:
    parent: menu_bar_form;
    height:200;
    width: 200;
    vertDistance:20;
    horizDistance:20;
```

```
    borderWidth: 3;
}

menu1_item: XawCommand {
 Attributes:
    parent: menu_bar;
    vertDistance: 10;
    horizDistance: 10;
    height: 20;
    width: 50;
    label: "Menu_1";

 Methods:
    notify: {
      display_menu1_submenu := TRUE;
    }
}

menu2_item: XawCommand {
 Attributes:
    parent: menu_bar;
    vertDistance: 10;
    horizDistance: 60;
    height: 20;
    width: 50;
    label: "Menu_2";

 Methods:
    notify: {
       display_menu2_submenu := TRUE;
    }
}

quit_menu_item: XawCommand {
 Attributes:
    parent: menu_bar;
    height: 20;
    width: 50;
    vertDistance: 10;
    horizDistance: 110;
    label: "QUIT";

 Methods:
    notify: {
      exit ();
    }
}
/*
```

```
** menu1 view controller
*/

VC: menu1_submenu

Creation Condition: (display_menu1_submenu)

Objects:

menu1_form: XawBboard {
 Attributes:
    parent: menu_bar;
    vertDistance: 30;
    horizDistance:10;
    borderWidth: 1;
    height: 65;
    width: 76;
}

item1_menu_item: XawCommand {
 Attributes:
    parent: menu1_form;
    vertDistance: 0;
    horizDistance: 2;
    height: 20;
    width: 70;
    borderWidth: 1;
    label: "Item 1";
}

item2_menu_item: XawCommand {
 Attributes:
    parent: menu1_form;
    vertDistance: 21;
    horizDistance: 2;
    height: 20;
    width: 70;
    borderWidth: 1;
    label: "Item 2 ->";
 Methods:
    notify: {
       display_sub_item_submenu := TRUE;
   }
}

remove_menu_item: XawCommand   {
 Attributes:
    parent: menu1_form;
```

```
    vertDistance: 42;
    horizDistance: 2;
    borderWidth: 1;
    height: 20;
    width: 70;
    label: "Close";
 Methods:
    notify:   {
      display_menu1_submenu := FALSE;
   }
}

ENDVC menu1_submenu


/*
** sub_item_submenu view controller
*/

VC: sub_item_submenu

Creation Condition: (display_sub_item_submenu)

Objects:

sub_item_form: XawBboard {
 Attributes:
    parent: menu_bar;
    vertDistance: 53;
    horizDistance: 88;
    borderWidth: 1;
    height: 65;
    width: 76;
}

itema_menu_item: XawCommand {
 Attributes:
    parent: sub_item_form;
    vertDistance: 0;
    horizDistance: 2;
    height: 20;
    width: 70;
    borderWidth: 1;
    label: "Item A";
}

itemb_menu_item: XawCommand {
 Attributes:
```

```
    parent: sub_item_form;
    vertDistance: 21;
    horizDistance: 2;
    height: 20;
    width: 70;
    borderWidth: 1;
    label: "Item B";
}
remove_menu_item: XawCommand   {
 Attributes:
    parent: sub_item_form;
    vertDistance: 42;
    horizDistance: 2;
    borderWidth: 1;
    height: 20;
    width: 70;
    label: "Close";
 Methods:
    notify: {
      display_sub_item_submenu := FALSE;
    }
}


ENDVC sub_item_submenu
```

Example 3-4 Menu Bar

When a menu item is selected, the variable that caused the view controller to be instantiated is set to `FALSE`. Since the creation condition for the view controller is no longer `TRUE`, the view controller is destroyed and the interaction objects are removed from the display. Normally, the `notify` method for the menu items would perform some command-specific action, such as notifying the application of the selected command.

3.5 Shared Data

Often the dialogue specifier may need to display interaction objects to the end user based on the existence of a particular data item. An example is an application that provides a function to query an employee database. The end user may make multiple requests to view the data for several different employees at the same time.

To specify a user interface for this example, it is convenient to define a view controller that controls the presentation of an employee record. The creation condition for the view controller can then be based on the existence of a new employee shared data instance in the shared database. This is accomplished using the `new` function in the creation condition of the view controller.

Figure 3-4 illustrates the shared data definition for an employee record and three instances of the shared data element. These are added to the shared database directly from the dialogue or by the application as described in *Serpent: System Guide*.

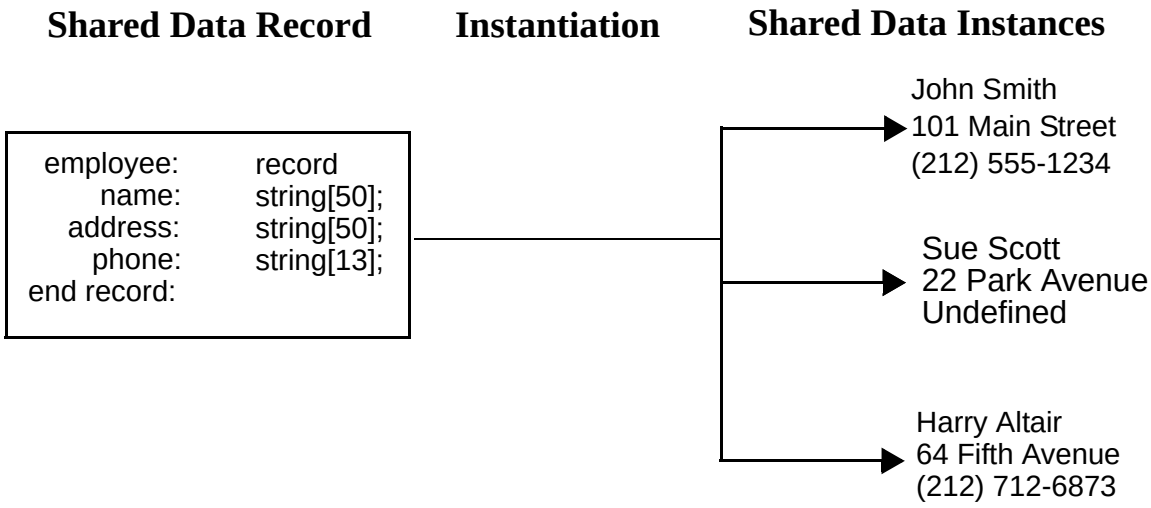

Figure 3-4 Shared Data Definition

The creation condition defined in the view controller template shown in Example 3-5 is based on the existence of a new employee shared data instance in the shared database. The creation condition uses the `new` function that serves only as a test for existence, not the action to create anything. A separate instance of the view controller is created for each of the three employee shared data instances and added to the shared database; a separate set of interaction objects is created for each view controller.

```
VC: employee_info
Creation Condition: (new("employee"))
Objects:

name_field: XawText {
 Attributes:
    height: 20;
    width: 200;
    label: employee.name;
}
.
.
.
END VC employee_info;
```

Example 3-5 View Controller Template

Each of these three view controller instances is *bound* to the shared data instance that caused it to be created. The dialogue specifier may then directly reference the specific values of each of the employee shared data instances within the scope of the view controller. For example, the `label` attribute of the `name_field` interaction object illustrated in Example 3-5 is set directly to the `name` component of the employee shared data instance.

This example illustrates how Slang provides a mapping between application shared data and interaction objects. Remember, there are actually three different employee records in the shared database and three different sets of interaction objects on the display. Serpent provides a mapping between shared data instances and their corresponding interaction objects and maintains the relationship between them.

3.6 Dialogue Shared Data

Dialogue shared data is a mechanism that allows a dialogue to create, modify, and destroy instances of data without informing an application or a toolkit of the actions.

Slang provides predefined routines that allow for the creation and destruction of shared data elements from within the dialogue. To create dialogue shared data, the type and structure of the shared data must first be defined in a shared data definition file. The shared data definition file must be named `dm.sdd`. Refer to *Serpent: Saddle User's Guide* for further information on creating a shared data definition.

Dialogue shared data is used in exactly the same fashion as application shared data. It is treated as shared data conceptually and with the same mechanisms, but the data actually is local and not shared with anything. It is possible, for example, to instantiate view controllers from new instances of dialogue shared data and to access this shared data in the same manner as application shared data.

The exact syntax and semantics of the shared data routines are defined in Appendix E.

3.7 Actions ON CREATE and ON DESTROY

To specify actions to be performed whenever a view controller is created or destroyed, use `ON CREATE` and `ON DESTROY` respectively. These mechanisms provide a means for executing procedural *code snippets* at defined instances within the dialogue model. Actions specified conditionally with `ON CREATE` and `ON DESTROY` may be used, for example, to increment and decrement counters.

Actions `ON CREATE` and `ON DESTROY` also may be used for calculating aggregate functions such as the average salary of all the employees in the employee database. The Slang dialogue illustrated in Example 3-6 performs this function.

In Example 3-6, the variable `average_salary` is defined to be `total_salary` divided by `employee_count`. Thus, `average_salary` depends upon both `total_salary` and `employee_count`. When either changes, `average_salary` is automatically changed. Thus, the view controller `employee_vc` has no associated objects. Its only function is to increase or decrease `employee_count` and `total_salary`. The actual presentation to the user is through the object `salary_field`.

```
Variables:
 employee_count: 0;
 total_salary: 0;
 average_salary: total_salary / employee_count;
/*
** if employee_count is 0, then average_salary
** becomes undefined
*/

Objects:
salary_field: XawText {
 Attributes:
    horizDistance: 100;
    vertDistance : 100;
    height: 20;
    width: 200;
    label: average_salary;
```

```
}
VC: employee_vc
Creation Condition: (new(”employee”))
On Create: {
 employee_count := employee_count + 1;
 total_salary := total_salary + employee.salary;
}
On Destroy: {
 employee_count := employee_count – 1;
 total_salary := total_salary – employee.salary;
}
END VC employee_vc;
```

Example 3-6 Calculating Aggregate Functions

3.8 Dialogue Structure

A Slang dialogue contains the specification of a user interface for a single application. The dialogue contains information about the application shared data (and by implication, the application) and the toolkits that are associated with the dialogue. The structure of the dialogue is:

- a list of application and toolkit shared data
- a list of view controllers used in the dialogue

The dialogue itself is always the outermost view controller and, consequently, all of the components of a view controller can be used in the dialogue as well as in explicitly declared view controllers.

The components of a view controller are:

- variables
- objects
- actions `ON CREATE`
- actions `ON DELETE`
- nested view controllers

Variable declarations serve two purposes: naming the variables for future reference and giving a definition of the variable which will be reevaluated whenever the independent variables in the definition are modified. The declaration of `average_salary` in Example 3-6 shows the use of variable declarations as a constraint definition. Whenever `total_salary` or `employee_count` are modified, `average_salary` is automatically reevaluated.

Objects declarations define the interaction objects that will be instantiated whenever a view controller instance is created. Each object that is declared has a collection of attributes that defines its presentation and a collection of methods that defines the response of the dialogue to end-user actions. Attributes are dependent on the variables that are used to define them (and consequently are reevaluated whenever the independent variables change), but methods are not a portion of the dependency analysis.

Actions `ON CREATE` and `ON DELETE` are code snippets that are executed once whenever a view controller is created (deleted). They are not subject to dependency analysis and are only evaluated when the relevant events occur.

4 Lexical Elements

This chapter describes the lexical components of Slang, including the characters that may appear in a Slang dialogue and the lexical units or *tokens* that they may form.

4.1 Character Set

Characters used in a Slang dialogue may consist of any characters from the standard ASCII character set. Slang is not case sensitive, except for characters written within string constants, preprocessor commands, and comments.

There are also special characters that are used in Slang either to separate adjacent tokens for the preprocessor or to format Slang dialogue text in a string. These characters include a blank space, end of line, formfeed, and horizontal tab.

Line termination is also a special character, but is generally ignored in Slang. It is important, however, in the recognition of preprocessor control lines, where the first character of the line must be a “#” character. The character following a line break character is considered the first character of the next line.

4.2 Comments

Comments begin with the characters /* and end with the first subsequent occurrence of the characters */. Comments, which are removed from the Slang program by the Slang preprocessor before a Slang dialogue is compiled, cannot be nested. For example, the following line:

```
/* /* Slang Source Commentary */
```

would be treated as a single comment and would generate a compiler error.

Another way to comment out dialogue lines is to use the preprocessor’s conditional commands. For example the following lines:

```
#if 0
 ...
#endif
```

would effectively comment out any section of the dialogue, including nested comments. For more information about the preprocessor refer to Chapter 12.

4.3 Tokens

The characters making up a Slang program are collected into lexical units called *tokens*. Tokens are the smallest lexical units that are recognized by the Slang compiler. When collecting characters into tokens, the Slang compiler always forms the longest token possible. For example, `vc_fred` is interpreted as a single identifier instead of the reserved word `VC` followed by the identifier `_fred`.

Adjacent tokens must be separated by white space or comments. In a macro body, separating tokens with comments rather than white space will cause the tokens to be merged. For example,

```
#define concat(x,y)
x/**/y
concat(A,B) => AB
```

Slang tokens are case insensitive, except for string constants. This means, for example, that the identifiers `Fred, fred`, and `FRED` are the same.

There are five classes of tokens in Slang: operators, special characters, identifiers, reserved words, and constants. Each of these classes is discussed in the following sections.

NOTE: Tokens that appear together in Slang, such as `Creation` and `Condition`, are accepted with or without a separator character. For example, `Creationcondition` and `Creation Condition` are both legal.

4.3.1 Operators and Special Characters

Slang has operators and special characters. Each operator has an associated precedence level that controls the order of operations in an expression. If parentheses are not used to indicate the grouping of operands with operators, the operand is grouped with the operator that has the higher precedence. For example, a + b * c would be grouped as a + (b * c).

Relative Precedence	Operator	Description
9	names, literals	simple tokens
8	func()	function call
8	.	component selection
7	not	logical negation
7	–	arithmetic negation
6	**	exponentiation
5	*, /	multiplicative

4	+, –	additive
3	<, >, <=, >=	relational
2	=, <>	equality/inequality
1	and, or	logical
0	:=	assignment

Slang supports the following special characters;

- # delineates preprocessor instructions
- \ identifies an escape character inside of text used for output
- { } delineates blocks of code
- , separates parameters in a function call
- ; terminates a statement
- : declares an attribute or variable

4.3.2 Identifiers

An identifier is a sequence of letters, digits, and underscore characters. An identifier cannot begin with a digit or an underscore and cannot have the same spelling as a reserved word. Identifiers may be of any length.

4.3.3 Reserved Words

The following are reserved words in Slang and may not be used as identifiers:

- AdiMethods
- AndNot
- AttributesNull
- Boolean Objects
- BufferOn Create
- CreateOn Destroy
- Creation ConditionOr
- DoReal
- ElseSelf
- ElsifString

- End IfThen
- End VcTrue
- End WhileUndefined
- ExitVariables
- FalseVc
- IdVoid
- IfWhile
- Integer

Table 4-1 Reserved Words

4.3.4 Constants

Constants are lexical elements that are characterized by having both a value and a type. There are four types of user-definable constants in Slang: boolean, integer, real, and string. There is also a system-defined type of constant which is an id.

There are also five predefined constants: `Undefined`, `Null`, `True`, `False`, and `Self`.

4.3.4.1 Integer Constant

An integer constant consists of a sequence of digits, 0-9. An integer constant may optionally include a leading + or –. Integer constants may not include commas or other non-digit characters. Examples of integer constants are: 1989, 0, and 1.

4.3.4.2 Real Constant

A real constant consists of a sequence of digits containing a single decimal point character. A real constant may optionally include a leading + or –. Examples of real constants are: .5, 1989., and 3.14.

4.3.4.3 String Constant

String constants consist of any arbitrary text delimited by double quotes. Examples of string constants are: “Fred,” “1989,” “string examples,” and “$%^&”.

String constants may also contain character or numeric escape codes. Escape characters are used to either represent characters that would be awkward to enter directly or to represent some particular formatting.

Escape characters consist of the backslash character followed by a character escape codes. Table 4-2 lists the character escape codes used in Slang.

\b	backslash
\f	form feed
\n	new line
\x	carriage return
\t	horizontal tab
\\	backslash

Table 4-2 Character Escape Codes

Numeric escape codes allow any character to be expressed by writing its octal code in the target character set. For instance, using the ASCII character set, the character a may be written as \141 and the `NULL` character as \0. Numeric escape codes terminate when either three characters have been read or a non-octal character is encountered. For example, the string \0111 consists of two characters: the character corresponding to octal code 011 and the character 1. The string \080 consists of three characters: the character corresponding to octal code 0 and the literal characters 8 and 0.

Predefined Constants

The `UNDEFINED` constant is used to represent a distinguished value which is recognized as undefined.

The `NULL` constant is an ID constant. It is used to specify that a given ID, while not undefined, does not identify any specific data item. The `Null` constant may be used, for example, to identify the end of a list.

The `Self` constant is an ID constant that is used to specify the ID of the current view controller or object, whichever is the local context.

5 Data

A Slang dialogue can be thought of as a mapping between application data and interaction objects. As such, the ability to declare and manipulate data in Slang is crucial to developing a dialogue. This chapter describes the different kinds of data that can exist in a Slang dialogue and how to declare and reference those data.

5.1 Different Forms of Data

Slang recognizes three forms of data:

1. Entities. These are data items such as shared data, `Vc`, `Object`, `Variable`, `Attributes`, and `Methods`, which have a predefined meaning within a Slang dialogue. Some entities have names (shared data, `Vc`, `object`, and `Variable`) and some have components that have names (shared data components, `Attributes`, and `Methods`).

2. Object type. This is the declaration type of an object entity. Each object is declared to be of a type. Admissible types are defined by the particular toolkits that have been included into a Slang dialogue. For example, an object can be declared to be a `XawCommand` widget. `XawCommand` is a type which is defined within the Athena Widget binding to Serpent.

3. Data type. This is the type of the value of shared data components, attributes, and variables. Components of application shared data, components of dialogue shared data, attributes, and constants all have fixed type. The type of a shared data component is specified in the shared data definition, the type of an attribute component is specified by the toolkit integrator, and the type of a constant is determined at compile time. Each variable has a dynamic data type that is determined at runtime for each assignment of value to the variable. The data value of a data type is its current value.

5.2 Assignment of Values to Data Types

When a data value is assigned to something with a fixed data type, its value is converted (if possible) to the fixed type. Appendix C presents the allowable conversions. When a data value is assigned to a variable (with dynamic type), the new type of the variable becomes the type of the data value.

For example, if `year` has a fixed data type of `integer`, the following assignments will yield equivalent values for `year`.

```
year := ”1984”;    year := 1984;
```

Example 5-1 Automatic Conversion

Alternatively, if `year` has dynamic type, the first assignment will result in a value typed as `string` and the second will result in a value typed as `integer`.

5.3 Object Types

An object is the only data item that has an object type. Variables, attributes, and shared data items have data types. Data types are discussed in more detail in the following paragraphs.

5.4 Data Types

Data items in Slang have either fixed or dynamic data type. For fixed data type, the type of the data item is determined at compile time. For dynamic data types, the type is determined dynamically at runtime. Examples of data items with fixed type are dialogue, application shared data, and object attributes. The only data items with dynamic type are dialogue variables.

Although fixed data types cannot change type, it is possible to assign data of a different type to a fixed data type if there is an appropriate conversion. This automatic conversion is illustrated in Example 5-2 The value `1984` is automatically converted to an integer so that it can be stored as the fixed typed integer `1984`.

```
year : integer;  (described in Saddle file)
year := “1984”;
```

Example 5-2 Fixed Data Type

Alternately, dynamic type data items can change type dynamically at runtime. In Example 5-3, the variable `year` is set to the integer value `1`. This means that the type of the variable is set to integer and the value assigned is `1`. In the following line, when the value `1984` is assigned to the variable, the type of the variable is set to string and the value is assigned `1984`.

```
year := 1;
year := “1984”;
```

Example 5-3 Dynamic Data Type

5.5 Base Types

There are six base data types supported in Serpent. These types are defined in Table 5-1. For more detailed information refer to *Serpent: Saddle User's Guide*.

Type	Description
`boolean`	true, false
`integer`	32 bits, from -2^{31} to $2^{31}-1$
`real`	64 bits, approximately 15 significant figures
`string`	variable length to maximum specified in Saddle description
`id`	data item identifier
`buffer`	n bytes of data together with length and type identifier

Table 5-1 Base Types

5.6 Dependency

An extremely important concept in Slang, dependency is one of the main ways in which the state of a Slang dialogue is modified at runtime.

A data item in Slang is dependent on another data item when the expression or code snippet (see Chapter 7) corresponding to the former data item references the latter. For example, if the variable `x` were assigned the value `2 * y` in the declaration statement, `x` would be automatically reevaluated whenever `y` changed. Expressions that are used in the evaluation of attributes of objects or in the declarations of variables are reevaluated whenever the independent variables in those expressions are modified. The independent variables can be either variables, attributes, or shared data components.

Data dependencies are determined dynamically at runtime. In Example 5-4, if the value of `w` is true, `x` is dependent on `y` and the expression will only be reevaluated if `y` changes or if `w` changes. If the value of `w` becomes false, then the expression would be dependent on `z` and will only be reevaluated if `z` or `w` changes. Determining dependencies dynamically helps to optimize runtime performance by reducing the situations in which the reevaluation of variables and attributes is necessary.

```
x: {
If (w) Then
 x := y;
Else
 x := z;
End If;}
```

Example 5-4 Dependency Propagation

It is possible when doing runtime dependency propagation to have infinite loops in dependent calculation. Potentially infinite loops within a Slang program are dealt with in one of two ways, depending upon how many snippets are used in the loop. The granularity of dependency calculations is at the snippet or expression level. That is, a snippet is determined to be dependent upon particular data items during execution. An infinite loop is suspected to exist (and execution is terminated) when a particular snippet is evaluated a number (currently 10) of times for the same data items. In Example 5-5, where variable `var1` is dependent upon variable `var2` and vice versa, the definitions are two independent snippets.

```
var1: var2;
var2: var1;
```

Example 5-5 Multiple Snippet Infinite Loop

There are cases in which it is possible to have a snippet dependent upon a variable that is also defined within the snippet. In these cases, the runtime system does not propagate dependencies. In Example 5-4, the snippet used in the definition of `x` depends upon the value of `x` (from the boolean expression "`x = y`"). The snippet also modifies the value of `x`. Thus, if snippets were always reevaluated when the independent value is modified, the definition of `x` would cause an infinite loop. Because, however, dependency granularity is at the snippet level, this snippet is treated as a dependency on `x`, but not as a modification of `x`. If `x` is modified from outside this snippet, the declaration is reevaluated but it is not reevaluated as a result of the modification of `x` from within the snippet.

```
x: {
If (x = y) Then
   x := y + 1;
Else
 x := y;
End If;}
```

Example 5-6 Snippet Dependent On Defined Variable

Example 5-7 shows another potential infinite loop. In this case, the variable `temp` is referenced within the snippet (from the second statement). Thus, if the granularity were at the statement rather than the snippet level, an infinite loop would result the second time the snippet is executed (the snippet depends upon the variable `temp` and `temp` is modified by the first statement in the snippet). Because the granularity is at the snippet level, this case is treated as a modification of `temp` but not as a use of `temp`. That is, if `temp` is modified, the snippet is not reevaluated and no infinite loop results.

```
x: {
 temp := 3 * var1 + var2;
 x := temp * temp;
}
```

Example 5-7 Snippet Dependent On Variable Both Modified and Used

5.7 Scope and Visibility

The *scope* of a data item in Slang is the set of statements and expressions in which the use of the identifier is associated with that particular data item. Slang supports block structured scoping in a fashion similar to that of programming languages such as Pascal.

Scopes are defined by view controllers and interaction objects. This could be alternately stated by saying that view controllers and interaction objects provide context for data items. Within the context in which they are defined, data items are said to have local scope. With respect to the other children of the parent, data items are said to have global scope. All data items declared in the same context must have unique names.

Data items are said to be *visible* if the identifier for that data item can be associated with the value. Data items are typically visible within their scope unless they are hidden. Hiding occurs when a data item having local scope has the same name as a data item with global scope.

The dialogue itself can be thought of as a view controller with a creation condition of true. Data items declared at the topmost level of the dialogue are then local to the dialogue and global to all other view controllers defined within the dialogue.

The following example illustrates the concepts of scope and visibility within Slang in terms of an abstract block structured language. The blocks can be thought of as being either view controllers or interaction objects in Slang.

```
VC: A
Variables:
 x;
 y;
 VC: B
 Variables:
    x;
    z;
 On Create: {
    x := 1;              /* assigns B.x */
    y := 2;              /* assigns A.y */
    z := 3;              /* assigns B.z */
 }
 END VC B;
 VC: C
 On Create: {
    z := 1; /* illegal */
 }
 END_VC C;
END_VC A;
```

Example 5-8 Scope and Visibility in an Abstract Block Structured Language

In this example, view controllers `B` and `C` are defined inside the context of view controller `A`. Since both `x` and `y` are declared within view controller `A`, they are considered to be local to `A` and global to both `B` and `C`. Data items declared within view controller `B` or `C` cannot be referenced from view controller `A` since they are outside the scope of `A`.

There are three assignment statements within view controller `B`. The first assignment assigns the value of `1` to data item `x` declared in view controller `B`. This is because the name of the data item (`y` in this case) is hiding the identically named data item declared in view controller `A`. The second assignment assigns the value of `2` to the data item `y` declared in view controller `A`. The assignment of the value of `3` to data item `z` is a simple assignment to the data item declared locally in view controller `B`.

The assignment in view controller `C` of the value of `1` to data item `z` would be an error condition since data item `z` is outside the context of view controller `C` and will result in a compiler error message.

Data items can be referenced outside their scope if they exist and the exact path is known. A *path* is a description of the location of a data item in the dialogue structure. Referencing data using a path is described in Section 5.9.

5.8 Extent

The *extent* of a data item refers to the period of time for which storage is allocated for the data item. Data items in Slang may have either local or dynamic extent.

A data item with local extent is created at the same time as either a view controller or an interaction object. Examples of data items with local extent are variables and object attributes.

Data items with dynamic extent may be created or destroyed at any time by either the dialogue or application program. Only shared data has dynamic extent.

5.9 Data Access

Each instance of a named entity in a Slang program (view controller, variable, object, and shared data element) is assigned a Serpent identifier when the instance is created. This identifier is of type ID. The identifier, for the most part, is unnecessary for the Slang programmer since the data items are within the scope of their use and can be referenced directly by name. It is possible, however, to reference data items by using their identifier. This is a useful feature when it is necessary to access a value that is out of the current context. Values of variables, attributes of objects, and shared data components can be set or retrieved through the use of the data access routines described in Appendix E.

Several different methods exist to determine the identifier for a data object:

At creation time: The creation of a shared data element through use of the `create_sd_instance` returns an identifier for the shared data element that has been created.

Special functions:

The function `get_parent_vc` returns the identifier of a view controller.

The function `get_object` returns the identifier of an object within a view controller.

The special `SELF` constant can be used to refer to the identification of objects, variables, and view controllers:

- When used within an object attribute or method specification, `SELF` refers to the ID for that object.
- When used within a variable specification, `SELF` refers to the ID for that variable.

- When used within the actions on create or destroy code snippets, `SELF` refers to the ID of the enclosing view controller.

The use of data access routines to set or retrieve identifiers or values requires explicit knowledge of the access path to the data item from the current context and, consequently, should only be used when absolutely necessary. For example, the addition of an intermediate view controller may change the access path from one compilation to another and a section of code that worked correctly prior to the addition may no longer work correctly.

5.10 Declared Data

This section describes data types that are declared within the dialogue.

5.10.1 View Controllers

View controllers serve two functions. At runtime, they provide a visibility mechanism for a collection of objects. When an instance of a view controller is created, those objects declared within the view controller are created and are thus made visible to the end user.

At compile time, view controllers provide context for variables, interaction objects, and bound shared data instances. A shared data instance is bound to a view controller when the creation condition for the view controller references the shared data element and the creation condition becomes true.

View controller instances are data items within a dialogue that have unique instance IDs. View controller IDs can be obtained using either the `SELF` constant or data access routines and can then be used to reference data items within the dialogue using the data access routines.

5.10.1.1 Declaration

A view controller may be referenced directly within its scope using the name of the view controller.

5.10.1.2 Extent

Variables have local extent. This means they exist only for the period of time in which the view controller instance that contains them exists.

5.10.1.3 Scope

Variables have local scope within the view controller in which they are declared and global scope within the subview controllers of that view controller. Being in scope within a view controller (whether local or global) means that the variable can be referenced within objects, subview controllers or variables declared within the view controller. Thus, in Example 5-8, variable `y` can be referenced from anywhere within View Controller `A`.

The order of declaration of variables is unimportant. Assume `x` is declared to be `y + 1` and `y` is declared to be `10`. If the declaration of `x` precedes the declaration of `y, x` is initially evaluated to be undefined and then `y` is evaluated to be `10`. The evaluation of `y` changes an independent variable in the declaration of `x` and, consequently, `x` is reevaluated to be given the value `11`.

5.10.2 Objects

Object declarations within Slang are templates that are instantiated when a particular view controller is instantiated.Object instances are data elements and have unique instance identifiers. Object identifiers can be determined using the `SELF` constant or data access routines. Objects provide context for attributes.

An object may be referenced directly within its extent, using its name.

Objects have the extent of the containing view controller. When the view controller is instantiated, an instance of the object is created and when the view controller is destroyed, the instance is destroyed.

5.10.3 Attributes

Attributes are used to define the presentation characteristics of interaction objects. Attributes are similar to shared data components in that they have fixed type. The type of the attributes is defined by the toolkit integrator.

An object attribute is declared to be either an expression or a code snippet in which the attribute is assigned (similarly to variables). The declaration generates a set of dependencies that are associated with the attribute. Every time any of the data items referenced in the declaration is modified, the attribute is reevaluated.

5.10.3.1 Type

An object attribute has a fixed type, either boolean, integer, real, string, buffer, or ID. Since

attributes are used to communicate with input/output toolkits, the values of attributes also have semantic meaning that is defined by the toolkit.

5.10.3.2 Extent

Attributes are created and destroyed with the object in which they are declared.

5.10.3.3 Scope

The scope of an attribute is limited to the object in which it is defined. Attributes may also be accessed from outside their scope if the exact path is known.

5.10.4 Methods

Methods provide a way of handling end-user interactions by specifying actions to be performed for specific end-user generated events. Each object type has a fixed collection of valid methods declared by the toolkit integrator. Methods are executed once for each end-user generated event.

5.10.5 Shared Data

Shared data is information that is communicated between the application and dialogue, solely within the dialogue, or between the dialogue and a toolkit. It consists of instances of shared data templates. Shared data is segmented: each application process communicating with Serpent has a segment, the dialogue has a segment, and each toolkit being used has a segment. The application segments are called *application shared data*, the dialogue segment is called *dialogue shared data*, and the toolkit segments are called *toolkit shared data.* The Slang programmer does not access toolkit shared data directly. Instead, the Slang programmer manipulates objects, object attributes, and methods; Serpent handles these objects internally through shared data.

Shared data resides in the *shared database*, is defined externally in a shared data definition file, and is instantiated at runtime by either the application or dialogue. Application and dialogue shared data instances can be bound to a view controller. A shared data instance is bound to a view controller when two conditions exist:

1. The creation condition for the view controller references the shared data element.
2. The creation condition is true.

5.10.5.1 Type

Shared data is different from dialogue internal data items in that it is declared externally to the dialogue. The type and structure of shared data is defined in a shared data definition file. A separate shared data definition file must be created for each Serpent application as well as for dialogue shared data.

A shared data definition file consists of aggregate data structures. The aggregate structures are referred to as *shared data elements*. Elements have components, each of which is declared to be one of the primitive types of Slang.

See *Serpent: Saddle User's Guide* for a complete description of shared data definitions.

5.10.5.2 Extent

Shared data has dynamic extent. It is created by either the application or dialogue and exists until it is explicitly destroyed.

5.10.5.3 Scope

The scope of a shared data instance includes any bound view controllers and any interaction objects and/or subview controllers defined within the scope of that view controller. Shared data items may be referenced directly within its scope using the name of the element or component.

Shared data can also be referenced from anywhere in the dialogue, if the ID is known, using the shared data routines defined in Appendix E.

5.10.6 Dialogue Shared Data

It is often useful in dialogue specification to be able to instantiate multiple instances of local data. For example, it is possible to create a dialogue without an application for prototyping purposes and, subsequently, to add the application. In such a situation, application shared data cannot exist because there is no application.

Dialogue shared data is like application shared data in that it is defined externally in a shared data definition file, can have view controllers bound to it, and must be explicitly instantiated and destroyed. It differs from application shared data only in that it is kept locally in the dialogue and not communicated to any external process.

The shared data definition file used to declare dialogue shared data must be named `dialogue_name.sdd`, where `dialogue_name` is the name of the file containing the dialogue, and the file name must be included in the prologue section of the Slang dialogue (see Section 9.5.1 for a discussion of prologues).

Dialogue shared data can be instantiated at runtime using the `create_sd_instance` function described in Appendix E. The `create_sd_instance` function returns a unique Serpent identifier that is used to reference, and later destroy, the shared data instance. In Example 5-9, the shared data element `dial_elem_sd` is instantiated and the identifier stored in `dial_elem_id`.

```
dial_elem_id := create_sd_instance("elem_name", "DM_BOX");
```

Example 5-9 Dialogue Shared Data Creation

The dialogue shared data element may be destroyed using the `destroy_sd_instance` function show in Example 5-7.

```
destroy_sd_instance(dial_elem_id);
```

Example 5-10 Dialogue Shared Data Destruction

5.10.7 Application Shared Data

Application shared data is used to communicate between the dialogue and an application. Shared data instances can be created by the application and communicated to the dialogue or created in the dialogue with the `create_sd_instance` function (described in Appendix E) and then communicated to the application.

Application shared data is specified using Saddle and is specific to an application. These concepts are discussed in detail in *Serpent: C Application Developer's Guide*, *Serpent: Ada Application Developer's Guide*, and *Serpent: Saddle User's Guide*.

5.11 Data Reference

In order to understand how data is referenced in a Slang dialogue, it is important to understand the dialogue structure.

5.11.1 Dialogue Structure

Dialogues have both a static specification-time structure and a dynamic, runtime structure. At specification time, the dialogue structure is a hierarchical tree of view controller and object templates. This static structure can be thought of as being the dialogue tree skeleton. At runtime, some of the templates have been instantiated (possibly multiple times). The runtime dialogue structure then consists of this tree skeleton plus whichever nodes on the tree skeleton have been instantiated. Since view controllers can instantiate multiple times, there may be multiple instances of the various nodes associated with a particular node on the skeleton. Each node in the dynamic structure has a unique path from the base of the structure.

Data reference can be performed in Slang using both direct and indirect referencing. The following subsections describe how each of these data referencing mechanisms are used.

5.11.2 Direct Reference

Data items in the dynamic structure can be referenced directly by path name from within their scope. The scope rules can be thought of as allowing upward referencing within the instance tree (as long as a name is not overloaded).

A path name consists of a series of tokens separated by the “.” symbol. The first symbol is considered the anchor, and determines the location from which the remainder of the path is determined. The anchor is bound to the first data item encountered in the dialogue structure, starting with the current context and working towards the dialogue base. Each of the following tokens must be defined in the context of the preceding token or a syntax error will occur. The rightmost token is the name of the actual data item being referenced.

The following are examples of referencing data using pathnames:

```
a_vc.a_subvc.a_variable
visible_vc.presentation_object.attribute
employee.name
```

Example 5-11 Direct Reference

5.11.3 Indirect Reference

Occasionally, it is necessary to reference other than upward in the instance tree. In this case, the Serpent identifier of the particular data item to be referenced must be determined. Once the identifier has been determined, the value of the data item can be set or retrieved using the data access and shared data routines defined in Appendix E.

In order to use indirect referencing, the identifier of the data item must be determined. A technique to use is to have the data item retrieve its identifier and save it in a variable within scope of the location where it is to be referenced. Example 5-14 shows a use of this technique.

5.11.4 Examples of Data Reference

To better illustrate data reference in Slang, some further examples from the employee database application are presented in this section. These examples assume two different shared data elements: one contains basic employee information and the other has information from the employees' previous reviews. These shared data records, specified in Saddle, are illustrated in Example 5-12.

```
employee: record
name:            string[50];
address:         string[50];
phone:           string[10];
salary:          integer;
end record;
review: record
name:            string[50];
ptr:             id_type; /*id of employee data element*/
year:            integer;
last_raise:      string[50];
level:           string[20];
end record;
```

Example 5-12 Employee Shared Data Definitions

The requirement is to provide a dialogue that will examine the database of employees and review information and display an employee bonus. Two dialogues (Example 5-13 and Example 5-14) show how to accomplish this function.

Example 5-13 is written using nested view controllers. The creation condition of `employee_vc` is based on the existence of a new employee shared data instance in the database. The sub-view controller `review_vc` is then created when a review shared data instance is found for the specified employee with a date field value of 1989. In this example, it is possible to directly reference shared data information from both the employee and review shared data instances from within the `review_vc` view controller to determine the employee bonus. Data references are automatically bound to the correct shared data instance of both shared data instances. The top level view control is instantiated whenever a new employee is added to shared data. The nested view controller is instantiated whenever a new review is added to the shared data that describes the employee referenced in the top most view controller. Thus, in the second view controller there are two conditions: one to add a new review to the shared data and the other to insure that the correct employee is referenced.

```
VC: employee_vc
Creation Condition: (new(”employee”))
Variables:
                 /*local variable is initialized*/
 name: employee.name;
 VC: review_vc
 Creation Condition:
    (new(“review”and name = review.name)
 Objects:
 due_for_raise: label_widget {
 /*
 ** omit most attributes
 */
 label {
    If (employee.salary < 20000)
    And
      review.level = ”superstar”)
    Then
      label := ”Merry Xmas.”;
    Else
      label := ”Forget it.”;
    End If;
    }
 }
 END VC review_vc;
END VC employee_vc;
```

Example 5-13 Direct Data Referencing

In the second version of this problem, Example 5-14, a single view controller is used for the solution. The assumption is that the application places the Serpent identifier for the correct employee shared data instance into the `ptr` component within the review element. The creation condition for the single view controller causes a new view controller to be created whenever a new record element is added with the desired year. The `ptr` component within the `review` element is used to reference the correct employee. Data access routines must be used to reference the values.

```
VC: review_vc
Creation Condition:
 (new(”review”) And review.date = 1989)
Objects:
due_for_raise: XawLabel {
 Attributes:
/*
** omit most attributes
*/
label: {
 If(employee[review.ptr].salary < 20000)
 And
    review.level = ”superstar”)
 Then
    label := ”Merry Xmas.”;
 Else
    label := ”Forget it.”;
 End If;
 }
}
END VC review_vc;
```

Example 5-14 Indirect Data Referencing

6 Expressions

Slang supports a standard set of unary and binary operators. Expression operands may have any underlying type (see Chapter 5). The resulting type and value of the expression is a function of both the operator and the type and value of each operand. The type results and conversions tables in Appendix C define the exact conversions that are performed at runtime. In general, operations on different types first try to coerce the types to integer, real, and string, in that order, in order to perform the operation. If no conversion can be performed then a runtime error occurs.

6.1 Undefined Values

Slang allows values of data items to be undefined. When an undefined value is used in an expression, the undefined value propagates outward during expression processing. For most operations, whenever an operand value is undefined, the resulting value is also undefined. The only exception is the equality operation in which two undefined operands are treated as being equal.

6.2 Logical AND and OR Operators

The logical AND and OR operators are used to provide logical operations on boolean values. Logical operators are not defined for anything other than boolean values and UNDEFINED.

A complete formal syntax for expressions is given in Appendix A. The following is the formal syntax for logical AND and OR expressions:

```
expression ::=
[expression logical_operator] boolean_expression
logical_operator ::=
And
| '&'
| Or
| '|'
```

Logical operations can return one of three values: true, false, or undefined. The following tables contain the values of logical operations.

AND	false	true	undefined
false	false	false	undefined
true	false	true	undefined
undefined	undefined	undefined	undefined

Table 6-1 AND Operations

The value of the first argument is read down the left column, the value of the second argument is read across the top, and the value of the result can be determined by finding the indicated entry in the table.

OR	false	true	undefined
false	false	true	undefined
true	true	true	undefined
undefined	undefined	undefined	undefined

Table 6-2 OR Operations

The value of the first argument is read down the left column, the value of the second argument is read across the top, and the value of the result can be determined by finding the indicated entry in the table.

6.3 Equality Operators

The equality operators are used to determine whether two data items are equal or not equal to each other. Data items can be compared for equality if both data items are of the same type or can be converted to the same type. Data items are considered equal if they have the same value in type to which they are converted. The following operators are used for comparing expressions

<=	equal
< >	not equal

Table 6-3 Equality Operators

The following is the formal syntax for equality expressions:

```
boolean_expression ::=
[boolean_expression equality_operator] relational_expression
equality_operator ::= '=' | '<>'
```

Equality operations can return one of two values: true or false. The following are some examples of equality comparisons

Operation	**Value**
1 = 5	false
“7” = 7	true
3.4 = 3	false
3 = 3.0	true
5<> UNDEFINED	true
“Fred” = “Tom”	false

Example 6-1 Equality Comparison

All operands of the same type can be compared for equality or inequality. In general, operands of mixed type may be compared when there is a defined conversion between the types. ID operands can only be compared with other ID operands. The exact semantics of equality operations is defined in Appendix C.

6.4 Relational Operators

Relational operators are used to compare two data items to determine their relative values; that is, if one data item has greater or lesser value than the other. Relational operators can only be used to make numerical comparisons. They cannot be used, for example, to alphabetize a list. Strings can only be compared if they can first be converted to a numerical type.

The following operators are used for comparing expressions:

<	less than
<=	less than/equal
>	greater than
>=	greater than/equal

Table 6-4 Relational Operators

The following is the formal syntax for relational expressions:

```
relational_expression ::=
[relational_expression relational_operator]
arithmetic_expression
relational_operator ::= '<' | '<=' | '>' | '>= '
```

Relational operations can return one of three values: true, false, or undefined. The following are some examples of relational comparisons.

Operation	Value
1 < 5	false
"7"> = 7	true
3.4 > = 3	true
3 < = 3.0	true
5 > UNDEFINED	UNDEFINED

Example 6-2 Relational Comparison

Operands used in relational operations must be either integers or real numbers or convertible to these two types. When one or both operands used in a relational operation is `UNDEFINED`, the result of the operation is also `UNDEFINED`. The exact semantics of relational operations are defined in Appendix C.

6.5 Arithmetic Operators

Arithmetic operators provide basic math functions. Arithmetic operations are only defined for numerical values. They cannot be used, for example, to add or concatenate strings. Arithmetic operations on strings are performed by converting the strings to a numeric type and then performing the arithmetic.

The following arithmetic operators are used for performing arithmetic operations on expression pairs.

Operator	Operation
`+`	`addition`
`-`	`subtraction`
`*`	`multiplication`
`/`	`division`
`**`	`exponentiation`

Table 6-5 Arithmetic Operators

Addition and subtraction have lower precedence than multiplication and division, which in turn have lower precedence than exponentiation. The following is the formal syntax for arithmetic expressions:

```
arithmetic_expression ::=
[arithmetic_expression addition_operator] term
term ::=
 [term multiplication_operator] factor
factor ::=
 [factor '**'] signed_id
arithmetic operator ::= '+' | '-'
multiplication_operator ::= '*' | '/'
```

Arithmetic operations can return values of two types: integer or real. They may also return `UNDEFINED` if either of the operands is `UNDEFINED`. The following tables give the type of arithmetic operations:

+,-,*,**	real	integer	undefined
real	real	real	undefined
integer	real	integer	undefined
undefined	undefined	undefined	undefined

Table 6-6 Plus, Minus, Multiple and Exponential Operations

/	real	real	undefined
real	undefined	undefined	undefined
integer	real	real	undefined
undefined	undefined	undefined	undefined

Table 6-7 Divide Operation

The exact semantics of arithmetic operations are defined in Appendix C. The following are some examples of arithmetic operations:

Operation	**Value**
1 + 5	6
"7" * 8	56
3.4 - 3	0.4
3 + 3.0	6.0
5/ UNDEFINED	UNDEFINED

Example 6-3 Arithmetic Operation

6.6 Unary Operators

There are two unary operators in Slang: unary negation and logical complement. Unary negation is used to negate a single, numeric operand. Logical complement is used to negate a single, logical operand. It is only defined for boolean values and undefined.

```
unary_operator ::=
     NOT
| '-'
```

Table 6-8 Unary Operators

The following is the formal syntax for unary expressions:

```
 signed_id ::=
[ unary_operator ] id

unary_operator ::=
NOT | '-'
```

Unary negation can return values of two types: integer or real. Unary negation also returns an UNDEFINED value when the operand is undefined. Logical complement can return one of three values: true, false or UNDEFINED. The following are examples of the unary operations:

Unary Operations	**Types**
-5	integer
- “3.14”	real
-UNDEFINED	UNDEFINED
not true	false
not UNDEFINED	UNDEFINED

Table 6-9 Unary Operations

7 Code Snippets and Statements

A *code snippet* is a fragment of procedural code within a dialogue specification that is executed in response to specific conditions. Code snippets are used to specify actions on create, actions on destroy, object attributes, object methods, and variable declarations. A code snippet consists of a collection of Slang statements.

The following is the formal syntax for code snippets:

```
code_snippet :: = '{' [statements] '}'
statement :: =
function_call |
assignment_statement |
conditional_statement |
loop_statement
```

Slang statements are all terminated by the semicolon character.

7.1 Function Call

A function call is a transfer of control to a procedure that exists outside of the Slang language but within the Serpent runtime system. Functions in Slang may or may not return a value but must not modify any of the actual parameters to the function. A function call may be a statement by itself (an imperative statement) or it may be a component of an expression. Predefined functions are defined in Appendix D and Appendix E.

The following is the formal syntax for imperative statements and functions:

```
function_call ::=
 function_name '(' [expressions] ')'
expressions ::= {expression ','} expression
```

The following are some examples of function call statements:

```
destroy_sd_instance(id);
put_variable_value(vc_id, "fred", 17);
```

Example 7-1 Function Call Statements

7.2 Assignment Statement

The assignment statement is used to set the value, and possibly type, of the data item on the left-hand side of the assignment operator to the value and type of the expression on the right-hand side of the operator.

The following is the formal syntax for assignment statements:

```
assignment_statement ::=
 qualifed_name ':=' expression
```

The following are some examples of assignment statements.

```
a := "string";
a := 7;
id := create_sd_instance(element_name);
```

Example 7-2 Assignment Statements

When the named identifier on the left-hand side of the assignment statement corresponds to a dialogue variable, both the type and value of the variable are set to the type and value of the expression. When the identifier corresponds to a shared data item or an object attribute, the value of the expression is converted to the type of the shared data item or attribute. The exact semantics of the assignment operation when the left-hand side of the statement is a shared data element or object attribute is defined in Table 12-3.

7.3 Conditional Statement

The conditional statement is used to provide optional execution of statements.

The following is the formal syntax for If statements:

```
conditional_statement ::=
If boolean_condition Then
            statements
{Elsif boolean_condition Then
            statements}
[Else
            statements]
End If
```

The following are examples of If statements:

```
If (a > 0) Then
 value = ”positive”;
Elsif (a = 0) Then
 value = “zero”;
Else
 value = “negative”
End If;
If (b > 5) Then
 printf(“b greater than five.”);
Elsf (b > 0) Then
 printf(“B greater than zero but less than five.”);
End If;
```

Example 7-3 Conditional Statements

When evaluating a conditional statement, the condition after the `IF` part and any conditions after the `ELSIF` parts are evaluated in lexical order (treating the final `ELSE` as an `ELSIF` true `THEN`) until one of the conditions is true or all the conditions are false. If one of the conditions evaluates to true the corresponding statements are executed; otherwise, none of the statements is executed.

7.4 Loop Statement

The loop statement is used to perform controlled iteration of Slang dialogue statements. All of the statements contained within the while loop are executed as long as the boolean condition evaluates to true.

The following is the formal syntax for the while statement.

```
loop_statement ::=
While boolean_condition [DO]
            statements
End While
```

The following is an example of a while loop that calculates the value of 5! (5 factorial).

```
i := 1;
x := 1;
While (i < 5) DO
 i := i+1;
 x := x * i;
End While;
```

Example 7-4 While Statement

The boolean condition of a loop statement is checked before the loop is entered. This means that the statements contained within the loop may not execute at all if the condition is not initially true.

8 Interaction Objects

Interaction objects are the means by which an end user visualizes and interacts with an application system. Interaction objects are instances, in the toolkit layer, of objects specified in a Slang dialogue. Objects in a Slang specification are object templates (for example, a label widget in the Motif toolkit binding). When the specification is executed, object instances are created within the dialogue. These object instances are passed to the toolkit layer and become interaction objects. Object instances are created when the view controller containing the object template is instantiated. An instance of an object is assigned an identifier when it is created and the identifier can be retrieved using the data access routines described in Appendix D.

Each object template has a collection of attributes that define the presentation of instances created from it, as well as methods that determine the high level interactions that the end user can have with the object. The syntax for object templates is:

```
objects ::=
Objects ‘:’ object_declaration { object_declaration }

object_declaration ::=
object_name ’:’ object_type_name
 ‘{ ’
   [ Attributes ‘:’ { attribute_value } ]
   [ Methods ‘:’ { method_handler } ]
‘ }’
```

8.1 Attributes

Attributes are used to define the presentation characteristics of object templates. Object attributes have fixed types of boolean, integer, real, string, buffer, or ID. Since attributes are used to communicate with toolkits, they also have semantic meaning that is defined by the toolkit integrator.

Attributes are created and destroyed along with the object instance in which they are defined. The scope of an attribute is identical to the scope of the object template in which it is defined. Attributes may be accessed from outside their scope if the exact path is known. Access is accomplished using the data access routines described in Appendix D.

Object attributes allow the specification of either code snippets or expressions that are associated with the attribute. Each specification is executed initially when the object instance is created and every time any of the data items (if any) referenced in the specification is modified. That is, the value of an attribute that depends upon certain data items is recalculated whenever those data items are modified.

The following is the formal syntax for attributes:

```
attribute_value ::= attribute_name ':' av_choice
av_choice ::=
code_snippet
| expression
```

The following are some examples of attribute definitions.

```
x: 5;
y: (x + 10) / 2;
label: temperature
color: {
 If (temperature < 32) Then
      color:="blue";
 Elsif (temperature < 100) Then
      color := "white";
 Else
      color := "red";
 End If;
}
```

Example 8-1 Attributes

Both the color and the label attributes, in the above example, are reevaluated whenever temperature changes. In the latter case, the integer value of temperature is automatically converted to the type of the label attribute (string).

8.2 Methods

Methods provide a way for handling end-user interactions in the dialogue by specifying actions to be performed for specific end-user generated events. Methods are executed once for each generated event and are procedural in nature. The number of and names of the methods for a given interaction object are defined by the toolkit integrator.

The following is the formal syntax for methods:

```
method_handler ::= method_name ':' code_snippet
```

The following are some examples of method definitions:

```
Methods:
 notify: {
   display_sizes_submenu := True;
 }
 send: {
   file_name := text_buffer;
   new_file_name := True;
 }
```

Example 8-2 Methods

9 View Controllers

A view controller template is used to control which interaction objects are presented to the end user under which circumstances. A view controller is an instance of a view controller template. View controllers provide a means of logically grouping interaction objects that are displayed and removed as a unit. When a view controller template is instantiated, the associated object templates are instantiated as object instances and passed to the toolkit layer as interaction objects. An example of this is a form containing multiple fields. Each field in the form is represented by a separate object template, but the entire form would most likely either be displayed or removed as a unit. View controllers may contain other view controllers nested to an arbitrary depth.

The following is the formal syntax for view controllers:

```
vc ::=
VC ‘:’ vc_name creation_condition
component_list
{ vc }
ENDVC vc_name [;]
component_list ::=
[variables]
[objects]
[actions_on_create]
[actions_on_destroy]
```

View controller instances are assigned identifiers in the same fashion as shared data element instances. The identifiers can be retrieved by means of the data access routines described in Appendix D.

9.1 Creation Conditions

Each view controller has a creation condition that defines the condition under which the view controller is instantiated.

The following is the formal syntax for creation conditions:

```
creation_condition ::=
Creation Condition ’:’ boolean_condition
```

There are two different classes of creation conditions in Slang: free and bound. Free creation conditions are boolean conditions that may reference existing data items. View controllers with a free creation condition can only be instantiated as many times as the surrounding view controller. The following are examples of free creation conditions:

```
Creation Condition: (True)
Creation Condition: (a < 5)
Creation Condition: (x = 1)
```

Example 9-1 Free Creation Conditions

Both `a` and `x` must be either variables or previously bound shared data items.

The second class of creation conditions causes the instantiated view controller to be bound to an instance of shared data. A bound creation condition must reference a single shared data item. It causes the view controller to be instantiated when an instance of the shared data element is created that is not currently bound to an instance of the view controller. This is accomplished by using the `new` function within the creation condition. The `new` function does not create new shared data; rather, it returns `true` when a new shared data item is created and becomes `false` when the shared data item is deleted.

A binding creation condition can also reference a component of a shared data element to cause the view controller to be instantiated only when a particular component assumes a particular value.

Creation conditions are within the scope of the enclosing view controller. Thus, only data items within the scope of the parent view controller can be used in a creation condition. References to shared data elements that have been bound in a parent view controller are not bound by the current creation condition.

The following are examples of bound creation conditions:

```
Creation Condition: (new("employee"))
Creation Condition: (new("employee") and display=true)
Creation Condition: (employee.location = "Pittsburgh")
```

Example 9-2 Bound Creation Conditions

Creation conditions cannot reference more than one non-bound shared data element. They can, however, reference any number of constants, variables, or attributes as long as they exist at the time the creation condition is evaluated. This does not include variables and attributes defined within the view controller for which the creation condition applies.

A view controller can only be instantiated if its parent already exists. In other words, in order for the creation condition for any view controller to be true, the creation condition for all the view controller's ancestors must also be true.

A view controller is destroyed when its creation condition becomes false, or the shared data instance that caused it to be instantiated is removed from the shared database. When a view controller is destroyed, all view controllers nested below it and child interaction objects of that view controller are also destroyed.

9.2 Actions "On Create"

Actions "on create" are used to specify a code snippet to be executed upon the creation of a view controller. Actions on create are executed once when the view controller is created; they are not reevaluated if the variables on which the code snippet is dependent change. The specification of an action `on create` code snippet is optional.

The following is the formal syntax for actions `on create`:

```
actions_on_create ::= On Create ':' [code_snippet]
```

The following is an example of an action `on create`:

```
On Create:   {
 counter := counter + 1;
}
```

Example 9-3 Actions on Create

9.3 Actions on Destroy

Actions `on destroy` are used to specify a code snippet to be executed upon the destruction of a view controller. Actions on destroy are executed once, when the view controller is destroyed. The specification of an actions `on destroy` code snippet is optional.

The following is the formal syntax for actions `on destroy`:

```
actions_on_destroy ::= On Destroy ':' [code_snippet]
```

The following is an example of an action `on destroy`:

```
On Destroy:   {
 counter := counter - 1;
}
```

Example 9-4 Actions on Destroy

9.4 Dependency Considerations

The interaction of three concepts within Slang can be important and confusing in some dialogues. The three concepts are: automatic reevaluation of some snippets (those used in variable declarations and those used in attribute definitions), the one-time evaluation of other snippets (those used within `on create` and `on delete`), and the timing of the creation of variables and objects within an instantiated view controller. In particular:

- Variables are evaluated before actions `on create` are executed. Thus, an action on creation can use the value of a variable set in its specification.
- Variable declarations are reevaluated when any independent variable within it is changed. Thus, a variable can be defined in terms of an attribute of an object created within the same view controller. When the attribute is modified or given a value, the variable declaration will be re-executed and a new value of a variable calculated.
- Actions `on create` are executed prior to the evaluation of attributes for objects. Thus, an action on create may depend on a variable declared in the same view controller, as indicated above. If that variable depends only on items outside the scope of the current view controller, or on other variables within the scope of the current view controller, then the expected behavior will be observed. On the other hand, if the variable depends upon an attribute of an object declared in the current view controller, the action `on create` will not yield the expected result. The sequence that causes the problem is:
    - Evaluate variable (since attributes of objects are not yet defined, the variable will use an `UNDEFINED` value for the attribute).
    - Perform the actions `on create`.
    - Evaluate the attributes of the object.

Since the variable depends on one of the attributes, it is reevaluated. Since actions `on create` are only performed once, they are not reevaluated.

9.5 Dialogue Structure

A Slang dialogue contains the specification of a user interface for a single application. The dialogue may be either a prototype or the user interface of a final product. Dialogues cannot be called or included from other dialogues.

The following is the formal syntax for a Slang dialogue:

```
Slang_dialogue = prologue [externals] component_list {vc}
```

The dialogue, as such, consists of the prologue, external declarations, a list of top level components, and a list of view controllers. Each of these top-level components is discussed in the following sections.

9.5.1 Prologue

Each Slang program must indicate the shared data files and toolkits that it uses. This is done in a prologue to the actual Slang program.

The prologue section of the dialogue is used to include the various .ill files required by the dialogue. An .ill file is a file that is generated when an application or toolkit shared data definition file is run through the Saddle preprocessor. *Serpent: Saddle User's Guide* contains more information on this process.

The prologue lists the shared data definitions and toolkit shared data. The form of this list is:

```
#include "name.ill"
#include "dm.ill"
#include "tech1.ill"
.
.
.
#include "techn.ill"
|||
```

where `name` is the name of the application shared data `.sdd` file. This can be omitted if there is no application shared data. The special name `dm.ill` is the name of the dialogue shared data. This can also be omitted if there is no dialogue shared data. The special delimiter "|||" is used to indicate the end of the prologue.

Each toolkit being used requires its own entry in the list. For the Athena X toolkit, the entry is:

```
#include "sat.ill"
```

For the Motif toolkit, the entry is:

```
#include "smo.ill"
```

The dialogue must include the .ill files for each toolkit referenced in the dialogue, the application .ill file if communication is required between the dialogue and application, and the `dm.ill` file if dialogue shared data is to be used.

The following is the formal syntax for the prologue section of the dialogue:

```
prologue ::= ill_file_contents { ill_file_contents } '|||'
```

The following lines provide a sample prologue section of a dialogue.

```
#include "app.ill"
            /* application shared data definition
            */
#include "dm.ill"
            /* dialogue manager shared data*/
#include "sat.ill"
            /* Serpent Athena Toolkit */
#include "tech_y.ill"
            /* Second I/O toolkit*/
|||
```

Prologue

9.5.2 Component List

The top-level dialogue structure can also be thought of as a view controller with a creation condition of true, with certain exceptions—the inclusion of both a prologue and externals section. As such, the dialogue can have almost all the same components as a view controller.

The following is the formal syntax for the component list:

```
component_list ::= [variables] [objects] [actions_on_create]
[actions_on_destroy]
```

Objects declared directly in the top level of the dialogue are always visible. Actions `on create` are executed immediately during execution of the dialogue. Actions `on destroy` are executed on dialogue exit.

10 User-Defined Functions

Slang provides a mechanism for invoking externally defined C functions from within a dialogue. The C function must be declared as an external in the Slang dialogue. The limitations on the C functions are:

1. They must be true functions; that is, a function should not modify any of its parameters. Thus, for example, the C library function `strcat` cannot be used directly since it returns the result in the first argument.

2. Functions that return arguments of type `string` or `real` must allocate static memory to hold the results.

10.1 External Functions

The following is the formal syntax for the externals section of the dialogue:

```
externals ::= {Externals ':' {external_declaration}}
external_declaration ::= external_type
function_name '(' [parm_list] ')'
external_type ::= Boolean
|                 Buffer
|                 Id
|                 Integer
|                 Real
|                 String
|                 Void
parm_list ::= {parm ','} parm
parm ::=          Boolean
|                 Buffer
|                 Id
|                 Integer
|                 Real
|                 String
```

The `function_name` can be any valid Slang name. Note that Slang can only bind to external functions whose names consist entirely of lower case letters and digits. The declaration `void FUNC (string, string);` is identical to `void func (string, string);` and maps into external function `func`. This restriction exists primarily because Slang is case insensitive while C is case sensitive.

The number of parameters accepted by the function should match the number and types given in the `parm_list`. It is possible to have functions with no parameters. For example, there is no character type in Slang. C functions that accept or return values of type character cannot be used.

Object files for external functions may be placed in a UNIX archive library (see UNIX ar(1) command) and the library may be passed to the Serpent command using the –L option to force the routines to link with the dialogue. Object files for external functions may also be linked directly using the -L option of the Serpent command. See *Serpent: System Guide* for a description of the Serpent command.

Note: not all valid C types have a defined conversion.

Example 10-1 shows external declarations in a Slang dialogue.

```
   /* string processing */
integer strlen(string);
integer strcmp(string, string);
   /* mathematical functions */
integer abs(integer);
real cos(real);
real pow(real, real);
```

Example 10-1 External Declarations

Once declared, the function can be used anywhere a function statement is valid in a Slang dialogue.

10.2 Existing External Functions

An external library of Slang-callable functions is supplied with Serpent. The serpent command ensures that this library is linked with Slang programs that require it. (See *Serpent: System Guide* for more information on the serpent command.) The definitions for each set of functions are available by including the appropriate header file. These header files, described in this section, reside in the directory externs/include under the installed Serpent directory; each one specifies the `EXTERNALS` keyword followed by a series of documented function definitions.

Each function package is described more completely in this section. Note that Slang forces references to a function to lower case and searches for the lower case version of the name.

10.2.1 Slang String Functions

The Slang string functions provide useful string manipulation routines. These functions are made available by including header file `sstring.ext`. In Slang, as opposed to C, strings are indexed starting from 1. The following is a list of the available functions, with a brief description of each one. A more complete description follows the list with examples showing how one might use the functions.

Function	Description
`string_append`	Appends one string to another, returning the new string.
`string_count_chars`	Counts the number of occurrences of a single character or a set of characters in a string.
`string_delete`	Deletes a substring from a string.
`string_index`	Finds the first occurrence of a substring within another string, returning the position of the substring.
`string_insert`	Inserts one string inside another.
`string_is_integer`	Determines whether a string represents a valid integer.
`string_is_real`	Determines whether a string represents a valid real number.
`string_length`	Returns the length of a string.
`string_lower`	Converts each upper–case character in a string to lower case.
`string_upper`	Converts each lower–case character in a string to upper case.
`substring`	Returns the specified “slice” from a string.

Table 10-1 Slang String Functions

Function

string_append

Description The string_append function appends one string to the end of another string.

Syntax

```
string_append (initial_string,terminal_string)
```

Parameters

initial_string The string on which to append terminal_string.

terminal_string The string to append to initial_string.

Returns A new string is returned containing the result. The original strings are left unchanged.

Function

string_count_chars

Description

The string_count_chars function counts the number of occurrences of a single character or a set of characters in a string.

Syntax

```
string_count_chars (the_string, char_set)
```

Parameters

the_string	The string to be checked.
char_set	A string containing characters to be counted. The function counts the number of times any character from this string appears in the_string. That is, to count only occurrences of a single character, this string should only contain one character.

Returns

A count (integer) of the number of times that characters from char_set were found in the_string. 0 means none were found.

Function

string_delete

Description

The string_delete function deletes a substring from a string.

Syntax

```
string_delete
(target_string,starting_position,
substring_length)
```

Parameters

target_string	The string from which to delete.
starting_position	The start of the substring to be deleted. This value must be between 1 and the length of target_string.
substring_length)	The length of the substring to be deleted. Cannot extend past the end of target_string.

Returns

A new string is returned containing the result. If any error is detected, (e.g.,starting_position or substring_length is incorrect) a null string is returned. The original string is left unchanged.

Function

string_index

Description

The string_index function finds the first occurrence of a substring within another string, returning the position of the substring.

Syntax

```
string_index (source_string, substring)
```

Parameters

source_string	The string in which to search.
substring	The string to locate.

Returns

If the substring exists within source_string, its location, offset from 1, is returned. Otherwise, a 0 is returned.

NOTE: Specifying a substring that is longer than the source string is not treated as an error. Since the substring can never be found, an 0 is returned in this case.

Function

string_insert

Description

The string_insert function inserts one string inside another at the specified position.

Syntax

```
string_insert
(base_string,insert_string,position)
```

Parameters

base_string	The string in which to insert.
insert_string	The string to be inserted.
position	The position (integer) in base_string at which to insert. The restrictions on this argument are: 1. Position must be greater than 0. 2. If position is 1 past the last character in base_string, insert_string is appended to base_string. (See string_append.) 3. Position cannot be greater than string_length (base_string + 1).

Returns

A new string is returned containing the result. If position is in error, a null string (“ ”) is returned. The original strings are left unchanged.

Function

string_is_integer

Description

The string_is_integer function determines whether a string contains only decimal digits.

Syntax

```
string_is_integer ( the_string)
```

Parameters

the_string	The string to be checked.

Returns

FALSE – The string does not represent a valid decimal integer.

TRUE – The string represents a valid decimal integer.

Function

string_is_real

Description

The string_is_real function determines whether a string represents a valid real number.

Syntax

```
string_is_real ( the_string)
```

Parameters

the_string	The string to be checked.

Returns

FALSE – The string does not represent a valid real number.

TRUE – The string represents a valid real number.

Function

string_length

Description The string_length function returns the length of a string.

Syntax

```
string_length (target_string)
```

Parameters

target_string	The string to examine.

Returns The length (integer) of the string. 0 means that the string was empty.

Function

string_lower

Description The string_lower function converts every uppercase alphabetic character in a string to lower case, leaving all other characters untouched.

Syntax string_lower (the_string)

Parameters the_string The string to be converted.

Returns A new string is returned containing the result. The original string is left unchanged.

Function

string_upper

Description

The `string_upper` function converts every lowercase alphabetic character in a string to uppercase, leaving all other characters untouched.

Syntax

`string_upper ( the_string)`

Parameters

`the_string`	The string to be converted.

Returns

A new string is returned containing the result. The original string is left unchanged.

Function

substring

Description

The substring function returns the specified portion of a string.

Syntax

```
substring (source_string, position, length)
```

Parameters

source_string	The string from which to extract a substring
position	The starting position (integer) of the substring. This value must be between 1 and string_length (source_string) .
string_length	The length (integer) of the substring. This value must be greater than position and must not result in a value greater than string_length source_string) when added to position.

Returns

A new string is returned containing the result. If the length values are inconsistent, a null string (“ ”) is returned. The original string is left unchanged.

10.2.2 Extended Arithmetic Functions

The extended arithmetic functions provide additional arithmetic functionality not already available in Slang. The following is a list of the available functions, with a brief description of each one. A more complete description follows the list. These functions are available by including header file `arith.ext`.

Function	Description
`div`	A function-oriented integer division operator.
`make_integer`	Converts a Slang expression to an integer, forcing truncation if necessary.
`mod`	A function-oriented integer modulo operator.
`truncate`	Converts a Slang expression to an integer, forcing truncation if necessary.

Table 10-2 Extended Arithmetic Functions

Function

div

Description

The div function provides an integer division operator.

Syntax

div (dividend, divisor)

Parameters

dividend	The integer to be divided.
divisor	The integer by which to divide dividend.

Returns

The integer portion of the quotient. Any fractional portion is discarded. This is equivalent to make_integer(dividend/divisor).

Function

make_integer, truncate

Description

The make_integer function converts a Slang expression to an integer, forcing truncation if necessary. It is also callable as truncate.

Syntax

make_integer (operand) truncate (operand)

Parameters

operand	The expression, variable, or attribute to be converted to integer.

Returns

An integer version of the expression, variable, or attribute is returned. Real data is truncated, not rounded.

Function

mod

Description The mod function provides an integer modulo operator.

Syntax mod (dividend, divisor)

Parameters

dividend	The integer to be divided.
divisor	The integer by which to divide dividend.

Returns The remainder (integer) from dividing dividend by divisor.

10.3 Using External Functions

External functions can be called from Slang: either existing C functions or new ones created as described in Section 10.3.2.

10.3.1 Existing C functions

There are many reasons to use external C functions. C has a rich library of utility functions that are available. For example, many of the string routines translate directly into C library calls. The C libraries are automatically linked into the Slang runtime without a necessity of explicitly stating the libraries in the serpent command. As another example, when debugging Slang programs, it is often useful to print debug messages in the window from which Serpent was invoked. The simplest way to accomplish this is to use existing C I/O routines. It is possible, for example, to call the C function `printf` from within a Slang program.

Note that Slang assumes that each external function has a fixed number of parameters; further, each parameter is assumed to have a fixed type. While `printf` does not traditionally behave in this way, it is still possible to use it in a Slang program. Consider the following `Externals` section of a Slang program:

```
Externals:
void printf (string, integer);
void puts (string);
```

With these declarations in place, you can now make the following calls in a Slang code snippet:

```
printf (”counter = %d and input_string = ”, counter);
puts (input_string);
```

Some of the versatility of `printf` is lost, of course. Since the `EXTERNALS` declaration specifies that the second parameter to `printf` is an integer, you can’t use `printf` to display real numbers; if you do, they’ll be truncated to integers before `printf` is actually called. Further, this binding to printf specifies that it expects only two parameters; you cannot call it with more than two parameters in this case.

You can only bind to an external function once. In other words, the following `EXTERNALS` section is not legal Slang:

```
Externals:
void printf (string, integer);
void printf (string, real);
```

The easiest way around this dilemma is to create your own routines that, in turn, call `printf` with varying type and varying number of parameters. These routines can then be used from within a Slang dialogue.

Electronic examples of how to use external functions are included in the base Serpent distribution in the subdirectory `demos/saw/slang-ref`.

10.3.2 Creating New External Functions

This section discusses the writing of an external C function to be called from a Slang program.

10.3.3 Type Equivalences

Slang data types map into specific C data types, as outlined in the table below. Types marked with [S] are defined in the “serpent.h” header file (which resides in the “include” subdirectory in your Serpent directory tree); these types are briefly described following the table. Types marked with [M] have special memory allocation requirements which are described in following section.

Slang Data Type	**Corresponding C Data Type**
real	double * (i.e., pointer to double) [M]
integer	int
string	char * [M]
boolean	boolean [S]
id	iid_id_type [S]

The boolean type is an integer that can be set to a true or false value. The definition of boolean also supplies a series of true and false constants. A true value can be specified with any one of the following constants:

true TRUE True on

Likewise, a false value can be specified with any one of:

false FALSE False off

The id type is used within Serpent to identify items in the shared database. You are discouraged from writing external functions that manipulate ids; however, in the interests of completeness, the type is included here. Ids are currently typed as long integers in C,

although in future releases of Serpent that representation may change.

The string type maps into a standard C string. That is, a Slang string variable is assumed to be either a NULL pointer or a pointer to a series of characters terminated by a null (i.e., '\0') byte.

10.3.4 Memory Allocation Considerations

Inside Serpent, variables of type boolean, id, and integer are stored as immediate data, so no memory allocation is required. That is, if you write a function that returns a boolean, id, or integer, you do not have to allocate memory to return the value; you can return it on the stack, as illustrated by this function to add two integers:

```
int int_add (op1, op2)
int op1;
int op2;
{
return (op1 + op2);
}
```

However, as the table above indicates, strings and real numbers are stored internally as pointers. Serpent assumes that the memory a string or real pointer refers to has been allocated by a special memory allocation routine called `make_node()`.

The `make_node()` routine is similar to the conventional C `malloc()` routine: it takes a single parameter specifying the number of bytes to allocate and returns a pointer to the allocated area. The returned pointer should be cast to the appropriate pointer type. To make use of `make_node()`, you must include the following lines in your source file:

```
#define memoryPack
#include "memoryPack.h"
```

(Both lines are necessary; the "#define memoryPack" line must precede the include directive.) It is important that you use `make_node()` to allocate memory, rather than `malloc()` or some other allocation routine; whenever the Serpent runtime system frees memory, it assumes the memory was allocated with `make_node()`.

As an illustration, contrast the above function to add two integers with this function to add two Slang real numbers:

```
#include <stdio.h>

#define memoryPack

#include "memoryPack.h"

double *real_add (op1, op2)
double *op1;
double *op2;
{
 double *result;
   if (
     (result = make_node (sizeof (double))) !=
     (double*) NULL )
{
 *result = *op1 + *op2;
}
return (result);
}
```

For example, the code below implements the Slang `functionstring_upper`:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define memoryPack
#include "memoryPack.h"

char *string_upper (the_string)
char *the_string;
{
int length;
char *result = NULL;
char *s;

 if ( (length = strlen (the_string)) > 0 ) {

   if ( (result = (char *) make_node (length + 1))==NULL )

    (void) fprintf (stderr,
      "string_upper: memory allocation error for string %s\n",
       the_string);
   else {
    for ( s = result; *the_string != '\0'; s++, the_string++ )
     if ( islower (*the_string) )*s = toupper (*the_string);
```

```
      else
        *s = *the_string;
      *s = '\0';
    }
 }
return (result);
}
```

10.3.5 Linking External Functions To Slang Programs

If you only use the external functions supplied with Serpent, you can skip this section, since the serpent command automatically searches the Serpent externals library when it links a dialogue. However, if you plan to write your own external functions, this section describes how to link them into a dialogue executable.

After you've written your external function (or functions), you must load them into an archival library (see the documentation for the Unix ar(1) command for more information on archival libraries). For example, suppose your functions reside in file "myfuncs.c". After compiling the file and producing "myfuncs.o", you might load the object file into archival library "mylib.a" with these commands:

```
% ar r mylib.a myfuncs.o

% ranlib mylib.a
```

(The "%" is the Unix prompt.)

Next, when linking a dialogue that calls one of your functions, you have to tell Serpent to resolve that function reference from""mylib.a". Use the "–L" option on the serpent command to accomplish this. Suppose, for example, the dialogue in file "mydialog.sl" uses a function in "myfunc.c". You would compile and link "mydialog.sl" using the following serpent command:

```
% serpent –cl –L mylib.a mydialog.sl
```

Refer to the manual page for the serpent command for more information on the –L option.

11 Runtime System

The Slang runtime system, or *dialogue manager,* is based on a system production model. That is, Slang statements are interpreted as rules to the runtime system and these rules are triggered whenever a data item is created, deleted, or modified. The runtime system keeps track of those data items that depend on other data items. Whenever the independent data items are created, deleted, or modified, the dependent data items are reevaluated.

The reliance of Slang on dependency is one of the most powerful features of the language, but it has several implications of importance to the dialogue specifier. These are: determination of cycles, the time that data is available to application and toolkits, and avoiding the dependency mechanism.

11.1 Cycles

Dependencies among data items are detected at runtime. This allows a dependent data item to depend upon the minimum possible number of independent data items but it also implies that cycles in dependencies can only be detected at runtime. A cyclic dependency occurs when two (or more) variables or object attributes are mutually dependent on each other. The following example illustrates a cyclic dependency:

```
x: y – 1
y: x + 1
```

This dialogue specification causes a cycle that the dialogue manager detects by counting the number of times a particular snippet is executed. Whenever that number exceeds a threshold, a cycle is determined to have occurred, execution is terminated, and an error message generated.

11.2 Timing of Data Transfers to Application and Toolkit

Activities within the dialogue manager are triggered by an external event. That event is the receipt of data from either the application or one of the toolkits. Once the dialogue manager is triggered, it processes all of the data dependencies that exist until no more data remains unchanged. At this time it informs the application and the toolkits of all changes in their respective shared data. The processing of all data dependencies occurs in a process called a minor cycle.

There are several implications of the fact that the dialogue manager processes all changes prior to informing the application or the toolkit of any changes:

- If a single data item is modified multiple times during a collection of minor cycles, only one change is sent out.
- The end user sees all changes to an interaction object at one time. If the dialogue generates some portion of the output and then calls an external function prior to generating the rest, the toolkit will not be informed of any changes until the external function exits. Thus, looping in an external function that returns one line of output at a time will not produce the expected results.

In general, the automatic runtime propagation of data is a very powerful mechanism, but occasionally it produces some behavior that is difficult to understand without a more in–depth explanation of the system.

11.3 Implications of Dependencies

It is important when writing dialogues to understand when dependencies are automatically propagated and when they are not. In general, computations associated with variable declaration and object attribute declaration are targets of propagation. Computations associated with methods, actions `on create,` and actions `on delete` are not targets of propagation. Some implications of this follow.

Counters are handled by declaring them in the variable section, incrementing them in the `on create` section and decrementing them in the `on delete` section.

Shared data items cannot be declared to be dependent upon other data items. Thus, shared data items need to be explicitly modified when they should be changed. On the other hand, other data items can be dependent upon shared data items and are automatically recalculated when necessary.

12 Slang Preprocessor

Serpent uses the C preprocessor. The C preprocessor is used to make macro substitutions, conditional compilations, and inclusions of named files. All C preprocessor commands begin with the # symbol. The # symbol must be located in column one followed by one of the preprocessor commands. The character following a line break character is considered the first character of the next line.

Preprocessor control lines may also be extended on the following line by inserting a backslash "\" character in the last position in a line. This causes the backslash character to remove itself and the following line break character. Backslash characters in normal Slang text are illegal.

12.1 Macros and Conditional Compilation

Slang uses the same preprocessor as the C programming language. This preprocessor is a simple macro processor that processes the source text of a Slang program before the compiler processes the source program.

The preprocessor supports the following commands:

Command	Description
`#define` *`identifier token-string`*	Define a preprocessor macro.
`#undef` *`name`*	Remove a macro definition.
`#include` *`"filename"`*	Insert text from another file.
`#if` *`expression`*	Conditionally perform some action, based on the value of a constant expression.
`#ifdef` *`identifier`*	Conditionally include some text, if preprocessor identifier is defined.
`#ifndef` *`identifier`*	Conditionally include some text, if preprocessor identifier is not defined.
`#else`	Alternately include some text, if the previous #if, #ifdef, or #ifndef test failed.
`#endif`	Terminate conditional text.
`#line` *`constant identifier`*	Supply a line number for compiler messages.

NOTE: The preprocessor commands must be in lower case. If lowercase letters are not used, the preprocessor will return an error message.

Please refer to *A C Reference Manual*, Samuel P. Harbison, Guy L. Steele Jr., Second Edition, pages 26-48 or *The C Programming Language*, Brian W. Kernighan and Dennis M. Ritchie, pages 207-208 for more information about the C preprocessor.

Appendix A Glossary of Terms

application layer
Those components of a software system that implement the "core" application functionality of the system.

application shared data
The section of the shared database associated with the application. This data acts as the interface between the functional portion of the application system and Serpent.

atomic data item
A shared data component or a variable.

attribute
A characteristic of an interaction object that may be defined by the dialogue specifier.

bound
Associated with. A view controller instance is *bound* to the shared data instance for which it was created.

code snippet
"Islands" of procedural code that are executed at certain defined times in the execution of a dialogue or as a result of changes in the state of the system.

creation condition
The conditions under which a view controller template is instantiated.

data item
Anything that can be declared or specified in Slang.

dialogue
A specification of the presentation of application information to, and interactions with, the end user.

dialogue layer
Serpent layer that controls the dialogue between the application and the end user of the application.

dialogue manager
Serpent component that executes the dialogue.

dialogue model
The dialogue model provides the conceptual basis for dialogue specification. The dialogue model is primarily based on a data-driven, rule-based production model.

dialogue shared data
Mechanism that allows a dialogue to create, modify, and destroy instances of data without informing an application or toolkit of the activities.

dialogue variables
Variables defined in view controllers within a Slang dialogue.

drop-down menu
Menu that consists of a menu bar that contains a number of options. Selecting an option causes a submenu to appear directly below the menu bar.

extent
Refers to the period of time for which storage is allocated for a data item.

ID
Unique instance identifier of those Slang data items that may have multiple instances.

I/O toolkits
Existing hardware/software systems that perform some level of generalized interaction with the user.

interaction object
Objects that exist in a given toolkit and can be used to interact with the end user.

lexical structure
The characters that may appear in a Slang dialogue and the lexical units or tokens that they may form.

method
Provides a way for handling end-user interactions in the dialogue by specifying actions to be performed for specific end-user generated events.

path
Description of the location of a data item in the dialogue structure.

presentation layer
Serpent layer concerned with low level interaction with the user. This layer consists of the various I/O toolkits.

presentation independent
Independent of the user interface of the system.

scope
The scope of a data item is the set of statements and expressions in which the declaration of the identifier associated with that data item is valid.

shared data
Data that is managed by Serpent except for variables declared within view controller templates.

shared database
Data managed by Serpent. The database consists of application data, presentation data, and global dialogue data.

shared data definition
A shared data structure that may be instantiated at runtime; may be either a record or a scalar.

shared data instance
An instance of a shared data element.

shared data item
A component of a shared data record instance or a shared data scalar instance.

tokens
The smallest lexical units that are recognized by Slang.

transaction
A collection of updates to the shared database that is logically processed at one time.

user interface
Those components of a software system that specify the presentation of application information to, and interaction with, the end user.

view controller
Mechanism for defining control flow and existence of interaction objects in dialogues.

view controller instance
An instantiation of a view controller template that is bound to specific application data and interaction objects.

view controller template
A view controller specification.

visible
A data item is said to be visible if the identifier for that data item can be associated with the value.

white space characters
Characters that are used to separate adjacent tokens or format Slang dialogue text. These characters include: blank (space), end of line, vertical tab, form feed, horizontal tab, and comments.

Appendix B Slang BNF Grammar

This appendix defines the BNF grammar for Slang. The following conventions are used in the specification:

Uppercase letters are used to indicate terminals, as does anything enclosed in single quotes. Uppercase letters are used for reserved words, and singe quotes are reserved for punctuation literals.

- Lowercase letters are used to indicate a non-terminal (rule).
- Spaces between items in a rule indicate that they are separate, lexically different terms. A carriage return, line feed, or tab between two items in a rule has the same meaning as a space (except for reserved words).
- A vertical bar (|) separates choices, one of which must be used.
- Items enclosed in square brackets ([]) are optional.
- Items enclosed in curly brackets ({}) may be executed any number of times, including zero.
- Keywords are in boldface type.

Note: Reserved words consisting of two words can have white space between the words.

```
Slang_program ::=
        prologue {externals} component_list {vc}

prologue ::=
        <ill file contents> {Note: <ill file contents>
        are handled by the C preprocessor} '|||'

externals ::=
 EXTERNALS {external_type function_name
     '(' [parm_list] ')' eos}

external_type ::=
                   BOOLEAN
                | BUFFER
                | ID
                | INTEGER
                | REAL
                | STRING
```

```
                | VOID

parmlist ::=
        {parm ‘,’} parm

parm ::=
                 BOOLEAN
                |BUFFER
                |ID
                |INTEGER
                |REAL
                |STRING

vc ::=
        VC ‘:’ vc_name
        creation_condition
        component_list
        {vc}
        END VC vc_name

component_list ::=
        [variables]
        [objects]
        [actions_on_create]
        [actions_on_destroy]

creation_condition ::=
        CREATION CONDITION ‘:’ boolean_condition

variables ::=
        VARIABLES ‘:’ {variable_declaration}

variable_declaration ::=
        variable_name ’:’ v_choice

vvchoice ::=
         code_snippet
        | [expression] eos

objects ::=
        OBJECTS ‘:’ object_declaration
        {object_declaration}

object_declaration ::=
        object_name ‘:’ object_type_name ‘{ ’
        [ATTRIBUTES ‘:’ {attribute_value}]
        [METHODS ‘:’ {method_handler}] ’ }’

attribute_value ::=
        attribute_name ‘:’ av_choice
```

```
av_choice ::=
         code_snippet
        | expression eos

method_handler ::=
        method_name ':' code_snippet

actions_on_create ::=
        ON CREATE ':' [code_snippet]

actions_on_destroy ::=
        ON DESTROY ':' [code_snippet]

code_snippet ::=
        '{ ' [statements] ' }' [eos]

statements ::=
        statement eos {statement eos}

statement ::=
           conditional_statement
        | assignment_statement
        | imperative_statement
        | loop_statement

conditional_statement ::=
        IF boolean_condition
        THEN statements
        {ELSIF boolean_condition THEN statements}
        [ELSE statements]
        END IF

assignment_statement ::=
        qualified_name ':=' expression

imperative_statement ::=
        function_call

loop_statement ::=
        WHILE boolean_condition [DO]
        statements
        END WHILE

boolean_condition ::=
        '(' expression ')'


expression ::=
        [expression logical_operator]
        boolean_expression
```

```
boolean_expression ::=
        [boolean_expression equality_operator]
        relational_expression

relational_expression ::=
        [relational_expression relational_operator]
        arithmetic_expression

arithmetic_expression ::=
        [arithmetic_expression addition_operator] term

term ::=
        [term multiplication_operator]factor

factor ::=
        [factor '**'] signed_id

signed_id ::=
        [ unary_operator ] id

id ::=
           qualified_name
        | function_call
        | '(' expression ')'
        | constant

qualified_name ::=
        name { '.' name }

function_call ::=
        function_name '(' [ expressions ] ')'

expressions ::=
        expression { ',' expression }

logical_operator ::=
             AND
        | '&'
        | OR
        | '|'

equality_operator ::=
            '='
        | '<>'

relational_operator ::=
            '<'
        | '<='
        | '>'
        | '>='
```

```
addition_operator ::=
            '+'
          | '-'

multiplication_operator ::=
            '*'
        | '/'

unary_operator ::=
             NOT
        | '-'

constant ::=
            integer_constant
        | real_constant
        | boolean_constant
        | string_constant
        | UNDEFINED
        | NULL
        | SELF

integer_constant ::=
        digit {digit}

real_constant ::=
            digit {digit} '.' {digit}
        | '.'digit {digit}

boolean_constant ::=
            TRUE
        | FALSE

string_constant ::=
        ' " ' <any valid ASCII text> ' " '
        function_name ::= name

name ::=
        alphabetic_character_or_underscore
        [{alpha_or_digit_or_underscore}
        alpha_or_digit {alpha_or_digit}]

eos ::=
        ';' { ';' }

comment ::=
        '/*' <any valid ASCII text> '*/'
        Note: comments are handled by the C
        preprocessor and may occur anywhere in a Slang
        program
```

Appendix C Runtime Conversions

Each of the following tables defines the type results and conversions based on the types (and values) for a specified class of runtime operations. Each table defines the type coercions that are legal and the results of the coercions. Each shared data component and attribute has a type declared at specification time. Each variable value has a type determined at runtime. The first table gives the type of the result when a binary arithmetic operation (+, –, *, /, **) is performed.

C.1 Binary Arithmetic

argument 2: **argument 1**	**boolean**	**integer**	**real**	**string**	**id**	**buffer**	**undefined**
boolean	(4)	(4)	(4)	(4)	(4)	(4)	(4)
integer	(4)	integer	(3)	(1)	(4)	(4)	undefined
real	(4)	(3)	real	(2)	(4)	(4)	undefined
string	(4)	(1)	(2)	(5)	(4)	(4)	undefined
id	(4)	(4)	(4)	(4)	(4)	(4)	(4)
buffer	(4)	(4)	(4)	(4)	(4)	(4)	(4)
undefined	(4)	undefined	undefined	undefined	(4)	(4)	undefined

Table 12-1 Binary Arithmetic

The preceding is valid for all binary arithmetic operators (+, –, *, /, **) except for division, which has the following exceptions:

- Dividing by zero results in undefined.
- An integer divided by an integer has a real result.

Notes:

1. If the string operand can be converted to integer, do the conversion and proceed with the operation. If the string operand cannot be converted to integer, try to convert it to real. If it can be converted to real, then convert the integer argument to real and the result is real. Otherwise, it is a runtime error.

2. If the string operand can be converted to real, do the conversion and proceed with the operation. If the string operand cannot be converted to real, it is a runtime error.
3. Convert the integer operand to real and proceed with the operation.
4. Runtime error.
5. If both string operands can be converted to integer, convert them and then perform the operation. If neither string operand can be converted to integer, try converting to real. If neither can be converted to real, it is a runtime error.

C.2 Relational Operations

The following table gives the type of the comparison when a relational operation (>, <) is performed.

argument 2 **argument 1**	**boolean**	**integer**	**real**	**string**	**id**	**buffer**	**undefined**
boolean	(3)	(3)	(3)	(3)	(3)	(3)	(3)
integer		integer	real	(1)	(3)	(3)	undefined
real			real	(2)	(3)	(3)	undefined
string				(5)	(3)	(3)	undefined
id					(3)	(3)	(3)
buffer						(3)	(3)
undefined							undefined

Table 12-2 Relational Operations

1. If the string operand can be converted to integer, do the conversion and proceed with the operation. If the string operand cannot be converted to integer, try to convert it to real. If it can be converted to real, then convert the integer argument to real and the comparison is real. Otherwise, it is a run time error.
2. If the string operand can be converted to real, do the conversion and proceed with the operation. If the string operand cannot be converted to real, it is a runtime error.
3. Runtime error.
4. If both string operands can be converted to integer, convert them and then perform the operation. If both string operands cannot be converted to integer, try converting them to real. If both cannot be converted to real, it is a runtime error.

C.3 Assignment Operations

The following table gives the results of assigning a value to either shared data or an object attribute.

To	boolean	integer	real	string	id	buffer
From						
boolean	valid	(4)	(5)	(8)	invalid	valid
integer	(6)	valid	valid	valid	invalid	valid
real	(7)	(1)	valid	valid	invalid	valid
string	(8)	(2)	(3)	valid	invalid	valid
id	invalid	integer	invalid	invalid	valid	valid
buffer	(9)	(9)	(9)	(9)	(9)	valid
undefined	valid	valid	valid	valid	valid	valid

Table 12-3 Assignment Operations

Note: The following conversions only occur with shared data.

1. Convert the real operand to integer by truncating the value.
2. Valid if the string operand can be converted to integer; otherwise, it is a runtime error.
3. If the string operand can be converted to real, do the conversion and then continue with the operation. If the string operand cannot be converted to real, it is a runtime error.
4. Boolean is converted to integer value 1 if true and 0 if false.
5. Boolean is converted to real value 1.0 if true and 0.0 if false.
6. An integer can be converted to a boolean if the value of the integer is 0 (converted to false) or 1 (converted to true). Any other value results in a runtime error.
7. A real can be converted to a boolean if the value of the real is 0.0 (converted to false) or 1.0 (converted to true). Any other value results in a run time error.
8. True is converted to the string “true,” false is converted to the string “false.” The reverse conversions occur also (the string values are case sensitive). Any other values result in a runtime error.
9. If the type of the buffer is a Serpent-defined type and the conversion is defined for that type, then the conversion is done; otherwise it is a runtime error.

C.4 Unary Arithmetic Operations

boolean	(2)
integer	integer
real	real
string	(1)
record	(2)
id	(2)
buffer	(2)
undefined	undefined

1. If the string operand can be converted to integer or real, do the conversion and proceed with the operation. If the string operand cannot be converted to integer or real, it is a runtime error.

2. Runtime error.

Table 12-4 Unary Arithmetic Operations

C.5 Equality Operations

The following table gives the type of comparison when applying the equality operator.

	boolean	**integer**	**real**	**string**	**id**	**buffer**	**undefined**
boolean	boolean	(8)	(8)	(8)	(4)	(9)	(5)
integer		integer	(3)	(1)	(4)	(9)	(5)
real			real	(2)	(4)	(9)	(5)
string				string	(4)	(9)	(5)
id					ID	(9)	(5)
buffer						buffer	(5)
undefined						TRUE	

Table 12-5 Equality Operations

1. If the string operand can be converted to integer, do the conversion and proceed with the operation. If the string operand cannot be converted to integer, the operands are considered unequal.

2. If the string operand can be converted to real, do the conversion and proceed with the operation. If the string operand cannot be converted to real, the operands are considered unequal.

3. Convert the integer operand to real and proceed with the operation.
4. Runtime error.
5. A variable of undefined type is equal to a shared data component of undefined value, regardless of the type of the shared data component.
6. A buffer is equal to another type if the buffer can be converted to that type and the values are equal.
7. A boolean can be compared to an integer or real value if the value can be converted to boolean.
8. If the type of the buffer is one of the Serpent types, the comparisons are done using the Serpent type contained in the buffer. If the buffer type is not a Serpent type, the result is not equal.

Appendix D Data Access Routines

The data access routines provide a means of accessing and modifying view controllers, variables, and objects within a Slang dialogue. The routines provide access to data items for those portions of the dialogue that are not within scope of the data item. The following is a list and short description of these routines. A more complete description immediately follows:

`get_bound_sd_instance`	Gets the ID of the shared data element bound to a given view controller.
`get_name`	Gets the symbolic name of a view controller, object, or variable that has the given ID.
`get_object`	Gets the ID of a named object associated with a view controller instance.
`get_parent_vc`	Gets the ID of the parent view controller of the specified view controller or object.
`get_variable_value`	Gets the value of a specified variable.
`get_vc`	Returns the ID of the named view controller created for a given shared data element.
`put_variable_value`	Puts a value into a variable.

Function

get_bound_sd_instance

Description The get_bound_sd_instance function gets the ID of the shared data instance bound to the specified view controller. The view controller is specified through its ID.

Syntax

```
function get_bound_sd_instance( vc_id );
```

Parameters

vc_id	The ID of the view controller.

Returns The ID of the shared data instance.

Function

get_variable_value

Description

The get_variable_value function gets the value for a specified variable within a view controller instance.

Syntax

```
function get_variable_value (vc_id , name);
```

Parameters

vc_id	The ID of the view controller instance in which the variable lives.
name	The name of the variable.

Returns

The value of the specified variable.

Function

get_name

Description The get_name function gets the name of the specified data item.

Syntax

```
function get_name( item_id );
```

Parameters

item_id	The ID of the data item whose name is to be retrieved. The data item can be either a view controller, an object, or a variable.

Returns The symbolic name of the item whose ID was passed.

Function

get_object

Description

The get_object function returns the ID of object instance created for a specified view controller instance.

Syntax

```
function get_object (vc_id , object_name );
```

Parameters

vc_id	The ID of the view controller instance.
object_name	The name of the object as a string.

Returns

The ID of the object instance.

Function

get_parent_vc

Description

The get_parent_vc function gets the ID of the parent view controller for the specified view controller or object.

Syntax

```
function get_parent_vc (vc_or_object_id);
```

Parameters

vc_or_object_id The ID of the view controller or object.

Returns

The ID of the parent view controller. If the argument is the ID of an object instance, the function returns the ID of the surrounding view controller instance. If the argument is the ID of a view controller instance, the function returns the ID of the parent view controller instance.

Note:if the argument is the ID of the top level view controller, the system aborts.

PROCEDURE

put_variable_value

Description

The put_variable_value procedure is used to assign a value to a specified variable.

Syntax

```
procedure put_variable_value (vc_id , name , value);
```

Parameters

vc_id	The ID of the view controller instance of the specified variable.
name	The name of the variable.
value	The value to be assigned to the variable.

Function

get_vc

Description The get_vc function gets the ID of view controller instance bound to a specified shared data element.

Syntax

```
function get_vc (sd_id , vc_template_name) ;
```

Parameters

sd_id	ID of the shared data element.
vc_template_name	Name of the view controller template.

Returns The ID of the view controller instance.

Appendix E Shared Data Routines

Shared data routines are used to create, destroy, and manipulate shared data elements from within a Slang dialogue. The following is a list and short description of these routines. A more complete description immediately follows:

`create_sd_instance`	Creates a shared data instance.
`destroy_sd_instance`	Destroys a shared data instance.
`get_sd_value`	Gets the value of a component of a shared data element instance.
`put_sd_value`	Puts a value into a component of a shared data element instance.

Function

create_sd_instance

Description

The create_sd_instance routine creates an instance for the specified shared data element and returns a unique ID.

Syntax

```
function create_sd_instance (element_name ,
sdd_mailbox );
```

Parameters

element_name	The name of the shared data element as a string.
sdd_mailbox	The name of the mailbox for the shared data definition file in which the element is defined. This name is constructed from the name of the shared data definition file by capitalizing the prefix of the file and adding "_BOX" (for example, app.sdd uses a mailbox named APP_BOX).

Returns

The ID of the newly created shared data instance.

Procedure

destroy_sd_instance

Description

The destroy_sd_instance routine destroys the specified shared data instance in the shared database.

Syntax

```
procedure destroy_sd_instance ( shared_data_id );
```

Parameters

shared_data_id	The ID of the shared data instance to be destroyed.

Function

get_sd_value

Description

The get_sd_value function gets the value of a component of a specified shared data element.

Syntax

```
function get_shared_data_value (sd_id,
component_name ) ;
```

Parameters

sd_id	The ID of the shared data element.
component_name	The name of a component within the shared data element as a string.

Returns

The value of the specified shared data element

Procedure

put_sd_value

Description

The put_sd_value procedure puts the specified value into a component of the specified shared data item.

Syntax

```
put_shared_data_value (sd_id , component_name ,
value );
```

Parameters

sd_id	The ID of the shared data element instance.
component_name	The name of a component within the shared data element.
value	Value to be assigned to shared data component.

Appendix F Utility Routines

Utility routines serve special purposes. They are used as functions within a dialogue but are not exclusively for shared data or for data access.

`exit`	Terminates the dialogue manager and sends a SIGINT signal to all the taps that were started by the current invocation of the Serpent command.
`id_exists`	Used to test for the existence of either a shared data element, a view controller, a variable, or an object.
`new`	Used in a view controller template creation condition to indicate that an instance of shared data should be considered for causing a new instance of that view controller.
`recording_on`	Turns on the Serpent transaction recording function.
`recording_off`	Turns off the Serpent transaction recording function.

Function

exit

Description	The `exit` function terminates execution of a dialogue and any related applications and I/O technologies.
Syntax	`procedure exit ();`
Parameters	None.

Function

id_exists

Description

The id_exists function tests whether an ID refers to a valid shared data item, view controller, variable, or object.

Syntax

```
function id_exists ( id ) ;
```

Parameters

id	The ID to test.

Returns

TRUE	if the ID specifies a valid shared data item, variable, view controller instance, or object; FALSE otherwise.

Example

The following Slang program shows how one might use id_exists. The shared data definition file is appended to the end of the program.

```
#include "dm.ill"
#include "saw.ill"
|||
VARIABLES:
some_id;
is_there : false;
OBJECTS:
button : command_widget
{ATTRIBUTES:
label_text :
{IF ( is_there ) THEN
label_text := "THERE";
ELSE
label_rtext := "NOT THERE";
END IF;
}
METHODS:
notify: {
IF ( id_exists (some_id) ) THEN
destroy_sd_instance (some_id);
is_there := false;
ELSE
some_id := create_sd_instance ("some_sdd",
```

```
"DM_Box");
is_there := true;
END IF;
}
}
shared data definition file
<<test>>
some_sd : shared data
some_sdd : record
some_component : integer;
end record;
end shared data;
```

Function

new

Description

The new function determines the existence of an unbound shared data element in the shared database. The new function can only be used as part of the creation condition for view controllers.

Syntax

```
function new (shared_data_item)
```

Parameters

`shared_data_item` The name of a shared data element as a string.

Returns

`TRUE` when an unbound shared data element of the appropriate name is found `False` when either no unbound shared data element is found or the currently bound instance is deleted.

Procedure

recording_on

Description

The recording_on procedure turns on the Serpent transaction recording function.

Syntax

```
procedure recording_on (recording_file,
header_message);
```

Parameters

recording_file	The name of a UNIX file to which the transactions are written.
header_message	An identification string included in the recording file.

Procedure

recording_off

Description The recording_off procedure turns off the Serpent transaction recording function.

Syntax

```
procedure recording_off();
```

Parameters none

Appendix G Athena Widget Set

XawBboard

Serpent Name

XawBboard

include_file: X11/Xaw/Form.h
widget_type: widget
class: formWidgetClass

Description

The XawBboard widget is a form widget that does not perform geometry management for its children. (The XawBboard widget should be used instead of the form widget when creating a background form to parent other widgets.)

Attributes
Serpent

Name	X Type	Default
after	Widget	
allowUserMove	Boolean	false
allowUserResize	Boolean	false
toBottom	bottom	
focus	focus	
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
maintainSize	Boolean	
managedWhenCreated	Boolean	true
toTop	top	
widget	int	

Form Widget Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	white
backgroundPixmap	Pixmap	
borderColor	Pixel	black
borderPixmap	Pixmap	
borderWidth	Dimension	1
children	WidgetList	
colormap	Colormap	
defaultDistance	int	
depth	int	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	true
numChildren	Cardinal	
screen	Screen	
sensitive	Boolean	true
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Btn1Down>,<Btn1Up>: select() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Ctrl<Btn1Down>,<Btn1Up>: top() Ctrl<Btn2Down>,<Btn2Up>: bottom() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()	
width	Dimension	
x	Position	
y	Position	

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL

romVert	Widget	NULL
horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	unsigned_char	2
top	unsigned_char	0

Methods

Name	Parameters	Description
move	x, y, horizDistance, vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize this widget with the mouse, and sends the widget's x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse, and sends the location of that point to the dialogue in response to a user event (typically a shifted Btn1Down).
top		This method is sent to the dialogue when the widget is raised.
bottom		This method is sent to the dialogue when the widget is lowered.
select	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse, and sends the location of the point to the dialogue.

XawBox

Serpent Name

XawBox

include_file: X11/Xaw/Box.h
class: boxWidgetClass
widget_type: widget

Description

The XawBox widget provides geometry management of arbitrary widgets in a box of a specified dimension. The children are rearranged when resizing events occur either on the XawBox or on one of its children, or when its children are managed or unmanaged. The XawBox widget always attempts to pack its children as tightly as possible within the geometry allowed by its parent

Attributes

Serpent

Name	X Type	Default
after	Widget	
allowUserMove	Boolean	false
allowUserResize	Boolean	false
toBottom	bottom	
focus	focus	
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	NULL
selectedX	Position	0
selectedY	Position	0
toTop	top	
widget	int	

Box Widget Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
anbackground	Pixel	white
backgroundPixmap	Pixmap	
borderColor	Pixel	black
borderPixmap	Pixmap	
borderWidth	Dimension	1
children	WidgetList	
colormap	Colormap	
depth	int	
destroyCallback	CallbackList	
height	Dimension	
hSpace	Dimension	
mappedWhenManaged	Boolean	true
numChildren	Cardinal	
orientation	String	
screen	Screen	
sensitive	Boolean	true
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()	
width	Dimension	
x	Position	
y	Position	

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL
fromVert	Widget	NULL

horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	unsigned_char	2
top	unsigned_char	0

Methods

Name	Parameters	Description
move	x, y, horizDistance, vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize this widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XawCommand

Serpent Name

XawCommand

include_file: X11/Xaw/Command.h
class: commandWidgetClass
widget_type: widget

Description

The XawCommand widget is an area, often rectangular, that contains a text label or bitmap image. This area, which is selectable, is often referred to as a *button*. When the pointer cursor is on a button, it becomes highlighted by drawing a rectangle around its perimeter. This highlighting indicates that the button is ready for selection. When pointer button 1 is pressed, the XawCommand widget indicates its selection by reversing its foreground and background colors. When the button is released, the XawCommand widget's notify action will be invoked. If the pointer is moved out of the widget before the button is released, the widget reverts to its normal foreground and background colors, and releasing the button has no effect. This behavior allows the user to cancel an action.

Attributes

Serpent

Name	X Type	Default
after	Widget	
allowUserMove	Boolean	false
allowUserResize	Boolean	false
toBottom	bottom	
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	NULL
toTop	top	
selectedX	Position	0

selectedY	Position	0
widget	int	

Command Widget Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	white
backgroundPixmap	Pixmap	
bitmap	Pixmap	
borderColor	Pixel	black
borderPixmap	Pixmap	
borderWidth	Dimension	1
callback	CallbackList	Six_callback
colormap	Colormap	
cornerRoundPercent	Dimension	
cursor	Cursor	
depth	int	
destroyCallback	CallbackList	
font	XFontStruct	6x13
foreground	Pixel	black
height	Dimension	
highlightThickness	Dimension	
insensitiveBorder	Pixmap	
internalHeight	Dimension	2
internalWidth	Dimension	4
justify	unsigned_char	1
label	String	
mappedWhenManaged	Boolean	true
resize	Boolean	
screen	Screen	
sensitive	Boolean	true
shapeStyle	unsigned_char	
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize()

	Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() Ctrl<Btn1Down>,<Btn1Up>: top() Ctrl<Btn2Down>,<Btn2Up>: bottom() <Btn1Down>: set() <Btn1Up>: notify () unset() <EnterWindow>: highlight() <LeaveWindow>: unhighlight()
width	Dimension
x	Position
y	Position

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL
fromVert	Widget	NULL
horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	unsigned_char	2
top	unsigned_char	0

Methods

Name	Parameters	Description
move	x, y, horizDistance, vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
top		This method is sent to the dialogue when the widget is raised.

bottom		This method is sent to the dialogue when the widget is lowered.
notify		This method is sent to the dialogue in response to a user event (typically a Btn1Down).
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XawDialog

Serpent Name

XawDialog

include_file: X11/Xaw/Dialog.h
class: dialogWidgetClass
widget_type: widget

Description

An XawDialog widget, which is simply a special case of the Form widget, provides a convenient way to create a preconfigured form. The typical XawDialog widget contains three areas. The first line contains a description of the function of the XawDialog widget (for example, the string Filename); the second line contains an area into which the user types input; the third line contains buttons that allow the user to confirm or cancel the XawDialog input.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
toBottom	bottom	
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	
toTop	top	
selectedX	Position	0
selectedY	Position	0
widget	int	

Form Widget Resource Set

Name	X Type
accelerators	Accelerators
ancestorSensitive	Boolean
background	Pixel
backgroundPixmap	Pixmap
borderColor	Pixel
borderPixmap	Pixmap
borderWidth	Dimension
children	WidgetList
colormap	Colormap
defaultDistance	int
depth	int
destroyCallback	CallbackList
height	Dimension
icon	Pixmap
label	String
mappedWhenManaged	Boolean
numChildren	Cardinal
screen	Screen
sensitive	Boolean
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Ctrl<Btn1Down>,<Btn1Up>: top() Ctrl<Btn2Down>,<Btn2Up>: bottom() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()
value	String
width	Dimension
x	Position
y	Position

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL
fromVert	Widget	NULL
horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	unsigned_char	2
top	unsigned_char	0

Methods

Name	Parameters	Description
move	x, y, horizDistance, vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize this widget with the mouse and sends its x and y location and new width and height to the dialogue.
top		This method is sent to the dialogue when the widget is raised.
bottom		This method is sent to the dialogue when the widget is lowered.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XawForm

Serpent Name

XawForm

include_file: X11/Xaw/Form.h
class: formWidgetClass
widget_type: widget

Description

The XawForm widget can contain an arbitrary number of children, or *subwidgets*. The XawForm provides geometry management for its children, allowing individual control of the position of each child. Any combination of children can be added to an XawForm. The initial positions of the children may be computed relative to the positions of other children. When the XawForm is resized, it computes new positions and sizes for its children. This computation is based upon information provided when a child is added to the XawForm.

Attributes

Serpent

Name	X Type	Default
after	Widget	
allowUserMove	Boolean	false
allowUserResize	Boolean	false
toBottom	bottom	
focus	focus	
maintainSize	Boolean	true
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	NULL
selectedX	Position	0
selectedY	Position	0
toTop	top	

widget	int

Form Widget Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	white
backgroundPixmap	Pixmap	
borderColor	Pixel	black
borderPixmap	Pixmap	
borderWidth	Dimension	1
children	WidgetList	
colormap	Colormap	
children	WidgetList	
colormap	Colormap	
depth	int	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	true
numChildren	Cardinal	
sensitive	Boolean	true
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Ctrl<Btn1Down>,<Btn1Up>: top() Ctrl<Btn2Down>,<Btn2Up>: bottom() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <Btn1Down>,<Btn1Up>: select()
width	Dimension	
x	Position	
y	Position	

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL
fromVert	Widget	NULL
horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	unsigned_char	2
top	unsigned_char	0

Methods

Name	Parameters	Description
move	x, y, horizDistance, vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize this widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).
top		This method is sent to the dialogue when the widget is raised.
bottom		This method is sent to the dialogue when the widget is lowered.

XawLabel

Serpent Name

Xawlabel

include_file: X11/Xaw/Label.h
class: labelWidgetClass
widget_type: widget

Description

An XawLabel widget is a text string or bitmap displayed within a rectangular region of the screen. The label may contain multiple lines of Latin1 characters. The XawLabel widget will allow its string to be left, right, or center justified. Normally, this widget can be neither selected nor directly edited by the user. It is intended for use as an output device only.

Attributes

Serpent

Name	X Type	Default
after	Widget	
allowUserMove	Boolean	false
allowUserResize	Boolean	false
toBottom	bottom	
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	NULL
toTop	top	
selectedX	Position	0
selectedY	Position	0
widget	int	

Label Widget Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	white
backgroundPixmap	Pixmap	
bitmap	Pixmap	
borderColor	Pixel	black
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
cursor	Cursor	
depth	int	
destroyCallback	CallbackList	
font	XFontStruct	6x13
foreground	Pixel	black
height	Dimension	
insensitiveBorder	Pixmap	
internalHeight	Dimension	2
internalWidth	Dimension	4
justify	unsigned_char	1
label	String	
mappedWhenManaged	Boolean	true
resize	Boolean	true
screen	Screen	
sensitive	Boolean	true
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Ctrl<Btn1Down>,<Btn1Up>: top() Ctrl<Btn2Down>,<Btn2Up>: bottom() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()	
width	Dimension	
x	Position	

y	Position

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL
fromVert	Widget	NULL
horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	type unsigned_char	2
top	unsigned_char	0

Methods

Name	Parameters	Description
move	x, y, horizDistance, vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize this widget with the mouse and sends its x and y location and new width and height to the dialogue.
top		This method is sent to the dialogue when the widget is raised.
bottom		This method is sent to the dialogue when the widget is lowered.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse, and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XawMenuButton

Serpent Name XawMenuButton

include_file: X11/Xaw/MenuButton.h
class: menuButtonWidgetClass
widget_type: widget

Description The XawMenuButton widget is a (typically) rectangular area that contains a text label or bitmap image. When the pointer cursor is on the button, the button becomes highlighted by drawing a rectangle around its perimeter. Highlighting means that the button is ready for selection. When selected, the XawMenuButton will pop up the menu that has been named in the menuName resource.

Attributes

Serpent

Name	X Type	Default
after	Widget	
allowUserMove	Boolean	false
allowUserResize	Boolean	false
toBottom	bottom	
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	NULL
selectedX	Position	0
selectedY	Position	0
toTop	top	
widget	Widget	

Menu Button Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	white
backgroundPixmap	Pixmap	
bitmap	Pixmap	
borderColor	Pixel	black
borderPixmap	Pixmap	
borderWidth	Dimension	1
callback	CallbackList	Six_callback
colormap	Colormap	
cornerRoundPercent	Dimension	
cursor	Cursor	
depth	int	
destroyCallback	CallbackList	
font	XFontStruct	6x13
foreground	Pixel	black
height	Dimension	
highlightThickness	Dimension	
insensitiveBorder	Pixmap	
internalHeight	Dimension	2
internalWidth	Dimension	4
justify	unsigned_char	1
label	String	
mappedWhenManaged	Boolean	true
menuName	String	
resize	Boolean	
screen	Screen	
sensitive	Boolean	true
shapeStyle	unsigned_char	
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move()

	Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <BtnDown>: reset() notify () PopupMenu() <EnterWindow>: highlight() <LeaveWindow>: reset()
width	Dimension
x	Position
y	Position

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL
fromVert	Widget	NULL
horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	unsigned_char	2
top	unsigned_char	0

Methods

Name	Parameters	Description
move	x, y, horizDistance, vertDisttance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize this widget with the mouse and sends its x and y location and new width and height to the dialogue.
notify		This method is sent to the dialogue in response to a user event (typically a Btn1Down).
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XawMenuShell

Serpent Name

XawMenuShell

include_file: X11/ /SimpleMenu.h
class: simpleMenuWidgetClass
widget_type: override

Description

The XawMenuShell widget is an override shell which acts as a container for the menu entries. The XawMenuShell serves as the glue to bind the individual menu entries together into a menu.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	
method	MethodName	
parent	Widget	NULL
widget	int	

Shell

Name	X Type
accelerators	Translations
ancestorSensitive	Boolean
allowShellResize	Boolean
background	Pixel
backgroundPixmap	Pixmap
borderColor	Pixel
borderPixmap	Pixmap
borderWidth	Dimension

bottomMargin		Dimension
children		WidgetList
colormap		Colormap
createPopupChildProc		Boolean
cursor		Cursor
depth		int
destroyCallback		CallbackList
geometry		caddr_t
height		Dimension
label		String
labelClass		WidgetClass
mappedWhenManaged		Boolean
menuOnScreen		Boolean
numChildren		Cardinal
overrideRedirect		Boolean
popdownCallback		CallbackList
popupCallback		CallbackList
popupOnEntry		Widget
rowHeight		Dimension
saveUnder		Boolean
screen		Screen
sensitive		Boolean
topMargin		Dimension
translations	Translations:	<EnterWindow>: highlight() <LeaveWindow>: unhighlight() <BtnMotion>: highlight() <BtnUp>: MenuPopdown() notify() unhighlight() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() Ctrl<Btn1Down>,<Btn1Up>: top() Ctrl<Btn2Down>,<Btn2Up>: bottom()
width		Dimension
x		Position
y		Position

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL
fromVert	Widget	NULL
horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	unsigned_char	2
top	unsigned_char	0

XawPaned

Serpent Name XawPaned

include_file: X11/Xaw/Paned.h
class: panedWidgetClass
widget_type: widget

Description The XawPaned widget manages children in a vertically or horizontally tiled fashion. The user may dynamically resize the pane by using the grips that appear near the right or bottom edge of the border between two panes.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
toBottom	bottom	
focus	focus	
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
toTop	top	
widget	int	

Paned Widget Resource Set

Name	Type	Default
accelerators	Accelerators	

ancestorSensitive		Boolean
background:		Pixel
backgroundPixmap		Pixmap
betweenCursor		Cursor
borderColor		Pixel
borderPixmap		Pixmap
borderWidth		Dimension
children		WidgetList
colormap		Colormap
cursor		Cursor
depth		int
destroyCallback		CallbackList
gripCursor		Cursor
gripIndent		Position
gripTranslations		Translations
height		Dimension
horizontalBetweenCursor		Cursor
horizontalGripCursor		Cursor
icon		Pixmap
internalBorderColor		Pixel
internalBorderWidth		Dimension
leftCursor		Cursor
lowerCursor		Cursor
mappedWhenManaged		Boolean
numChildren		Cardinal
orientation		String
refigureMode		Boolean
rightCursor		Cursor
screen		Screen
sensitive		Boolean
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

upperCursor	Cursor
verticalBetweenCursor	Cursor
verticalGripCursor	Cursor
width	Dimension
x	Position
y	Position

Constraint Resource Set

Name	X Type
allowResize	Boolean
max	Dimension
min	Dimension
preferredPaneSize	Dimension
resizeToPreferred	Boolean
showGrip	Boolean
skipAdjust	Boolean

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL
fromVert	Widget	NULL
horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	unsigned_char	2
top	unsigned_char	0

Methods

Name	Parameter	Description
move	x, y, horizDistance, vertDistance	This method allows the user to move this widget with the mouse and sends the widget's new x and y location to the dialogue.

resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XawScreenObject

Serpent Name XawScreenObject

Description The XawScreenObject widget allows for the detection of screen and display IDs:
Display size contains height and width
Display type contains color or black & white

Attributes
Serpent

Name	X Type
color	Boolean
display	int
height	Dimension
screen	Screen
width	Dimension

XawScrollbar

Serpent Name

XawScrollbar

include_file: X11/Xaw/Scrollbar.h
class: scrollbarWidgetClass
widget_type: widget

Description

The XawScrollbar is a rectangular area containing a thumb that, when moved along one dimension, will cause the scrolling of a region inside a box widget. The XawScrollbar may be oriented horizontally or vertically. Each pointer button invokes a specific action.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
toBottom	bottom	
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	NULL
selectedX	Position	0
selectedY	Position	0
toTop	top	
widget	int	

Command Widget Resource Set

Name	X Type
accelerators	Accelerators
ancestorSensitive	Boolean

background		Pixel
backgroundPixmap		Pixmap
borderColor		Pixel
borderPixmap		Pixmap
borderWidth		Dimension
colormap		Colormap
depth		int
destroyCallback		CallbackList
foreground		Pixel
height		Dimension
jumpProc		CallbackList
length		Dimension
mappedWhenManaged		Boolean
minimumThumb		Dimension
orientation		unsigned_char
screen		Screen
scrollDCursor		Cursor
scrollHCursor		Cursor
scrollLCursor		Cursor
scrollProc		CallbackList
scrollRCursor		Cursor
scrollUCursor		Cursor
scrollVCursor		Cursor
sensitive		Boolean
shown		float
thickness		Dimension
topOfThumb		float
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() Ctrl<Btn1Down>,<Btn1Up>: top() Ctrl<Btn2Down>,<Btn2Up>: bottom() <Btn1Down>: StartScroll(Forward) <Btn2Down>: StartScroll(Continuous) MoveThumb()

	NotifyThumb() <Btn3Down>: StartScroll(Backward) <Btn2Motion>: MoveThumb() NotifyThumb() <BtnUp>: NotifyScroll(Proportional) EndScroll()
width	Dimension
x	Position
y	Position

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL
fromVert	Widget	NULL
horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	unsigned_char	2
top	unsigned_char	0

Methods

Name	Parameters	Description
move	x, y, horizDistance, vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
top		This method is sent to the dialogue when the widget is raised.
bottom		This method is sent to the dialogue when the widget is lowered.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XawSimpleMenu

Serpent Name XawSimpleMenu

include_file: X11/Xaw/Sme.h
class: smeObjectClass
widget_type: widget

Description The XawSimpleMenu widget is a container for menu entries. The XawSimpleMenu serves as the glue to bind the individual menu entries together into a menu.

Attributes

Serpent

Name	X Type	Default
after	Widget	
allowUserMove	Boolean	false
allowUserResize	Boolean	false
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	NULL
selectedX	Position	0
selectedY	Position	0
toBottom	bottom	
toTop	top	
widget	int	

Menu

Name	X Type	Default
ancestorSensitive	Boolean	
destroyCallback	CallbackList	

height		Dimension
sensitive		Boolean
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <BtnDown>: reset() notify () PopupMenu() <EnterWindow>: highlight() <LeaveWindow>: reset()
width		Dimension
x		Position
y		Position

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL
fromVert	Widget	NULL
horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	unsigned_char	2
top	unsigned_char	0

Methods

Name	Parameters	Description
move	x, y, horizDistance vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
notify		This method is sent to the dialogue in response to a user event (typically a Btn1Down).

`pick`	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XawSimpleMenuBSB

Serpent Name

XawSimpleMenuBSB

include_file: X11/ /SmeBSB.h
class: smeBSBObjectClass
widget_type: widget

Description

The XawSimpleMenuBSB widget is a container for the menu entries. It differs from a plain menu widget in that it can contain bitmaps on both sides of a menu entry.

Attributes

Serpent

Name	X Type	Default
after	Widget	
allowUserMove	Boolean	false
allowUserResize	Boolean	false
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	NULL
selectedX	Position	0
selectedY	Position	0
toBottom	bottom	
toTop	top	
widget	Widget	

BSB Object

Name	X Type	Default
ancestorSensitive	Boolean	
callback	CallbackList	Six_callback

destroyCallback		CallbackList	
font		XFontStruct	6x13
foreground		Pixel	black
height		Dimension	
justify		unsigned_char	
label		String	
leftBitmap		Pixmap	
leftMargin		Dimension	4
rightBitmap		Pixmap	
rightMargin		Dimension	4
sensitive		Boolean	
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <BtnDown>: reset() notify () PopupMenu() <EnterWindow>: highlight() <LeaveWindow>: reset()	
vertSpace		int	25
width		Dimension	
x		Position	
y		Position	

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL
fromVert	Widget	NULL
horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	unsigned_char	2
top	unsigned_char	0

Methods

Name	Parameters	Description
move	x, y, horizDistance, vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
notify		This method is sent to the dialogue in response to a user event (typically a Btn1Down).
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XawSmeLine

Serpent Name XawSmeLine

include_file: X11/Xaw/SmeLine.h n
class: smeLineObjectClass
widget_type: widget

Description The XawSmeLine widget is an object used to add a horizontal line to a menu, acting as a menu separator. This object is not selectable.

Attributes

Serpent

Name	X Type	Default
after	Widget	
allowUserMove	Boolean	false
allowUserResize	Boolean	false
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	NULL
selectedX	Position	0
selectedY	Position	0
toBottom	bottom	
toTop	top	
widget	int	

SmeLine

Name	X Type	Default
destroyCallback	CallbackList	
foreground	Pixel	black
height	Dimension	

lineWidth		Dimension	1
sensitive		Boolean	
stipple		Pixmap	
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <BtnDown>: reset() notify () PopupMenu() <EnterWindow>: highlight() <LeaveWindow>: reset()	
width		Dimension	
x		Position	
y		Position	

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL
fromVert	Widget	NULL
horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	unsigned_char	2
top	unsigned_char	0

Methods

move	x, y, horizDistance, vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
notify		This method is sent to the dialogue in response to a user event (typically a Btn1Down).

`pick`	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XawText

Serpent Name

XawText

include_file: X11/Xaw/AsciiText.h
class: asciiTextWidgetClass
widget_type: widget

Description

The text widget provides a modifiable, emacs-style, text editor interface in a widget that is used to allow arbitrary text input. XawText in this widget can also be modified under program control and displayed back to the user.

Attributes

Serpent

Name	X Type	Default
after	Widget	
allowUserMove	Boolean	false
allowUserResize	Boolean	false
toBottom	bottom	
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	NULL
selectedX	Position	0
selectedY	Position	0
sendBuffer	Boolean	false
toTop	top	
widget	int	

Text Widget Resource Set

Name	X Type	Default
ancestorSensitive	Boolean	
autoFill	Boolean	
background	Pixel	white
backgroundPixmap	Pixmap	
borderColor	Pixel	black
borderPixmap	Pixmap	
borderWidth	Dimension	1
bottomMargin	Position	
callback	CallbackList	
colormap	Colormap	
cursor	Cursor	
dataCompression	Boolean	
depth	int	
destroyCallback	CallbackList	
displayCaret	Boolean	
displayNonprinting	Boolean	
displayPosition	int	
echo	Boolean	
editType	unsigned_char	2
font	FontStruct	6x13
foreground	Pixel	black
height	Dimension	
insensitiveBorder	Pixmap	
insertPosition	int	
leftMargin	Position	
length	int	
mappedWhenManaged	Boolean	true
pieceSize	int	
rightMargin	Position	
screen	Screen	
scrollHorizontal	unsigned_char	false
scrollVertical	unsigned_char	false
selectTypes	TextSelectType_star	

sensitive		Boolean	true
string		String	textBuffer
textSink		Widget	
textSource		Widget	
topMargin		Position	
translations	Translations:	Ctrl<Key>A: beginning-of-line() Ctrl<Key>B: backward-character() Ctrl<Key>D: delete-next-character() Ctrl<Key>E: end-of-line() Ctrl<Key>F: forward-character() Ctrl<Key>G multiply(Reset) Ctrl<Key>H: delete-previous-character() Ctrl<Key>J: newline-and-indent() Ctrl<Key>K: kill-to-end-of-line() Ctrl<Key>L: redraw-display() Ctrl<Key>M: newline() Ctrl<Key>N: next-line() Ctrl<Key>O: newline-and-backup() Ctrl<Key>P: previous-line() Ctrl<Key>R: search(backward) Ctrl<Key>S: search(forward) Ctrl<Key>T: transpose-characters() Ctrl<Key>U: multiply(4) Ctrl<Key>V: next-page() Ctrl<Key>W: kill-selection() Ctrl<Key>Y: insert-selection(CUT_BUFFER1) Ctrl<Key>Z: scroll-one-line-up() Meta<Key>B: backward-word() Meta<Key>F: forward-word() Meta<Key>I: insert-file() Meta<Key>K: kill-to-end-of-paragraph() Meta<Key>Q: form-paragraph() Meta<Key>V: previous-page() Meta<Key>Y: insert-selection(PRIMARY,CUT_BUFFER0) Meta<Key>Z: scroll-one-line-down() Meta<Key>d: delete-next-word() Meta<Key>D: kill-word() Meta<Key>h: delete-previous-word() Meta<Key>H: backward-kill-word() Meta<Key>\\<: beginning-of-file() Meta<Key>\\>: end-of-file() Meta<Key>]: forward-paragraph() Meta<Key>[: backward-paragraph() ~Shift Meta<Key>Delete: delete-previous-word() Shift Meta<Key>Delete: backward-kill-word() ~Shift Meta<Key>BackSpace: delete-previous-word()	

	Shift Meta<Key>BackSpace: backward-kill-word() <Key>Right: forward-character() <Key>Left: backward-character() <Key>Down: next-line() <Key>Up: previous-line() <Key>Delete: delete-previous-character() <Key>BackSpace: delete-previous-character() <Key>Linefeed: newline-and-indent() <Key>Return: newline() <Key>: insert-char() <FocusIn>: focus-in() <FocusOut>: focus-out() Ctrl<Btn1Down>,<Btn1Up>: top() Ctrl<Btn2Down>,<Btn2Up>: bottom() Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <Btn1Down>: select-start() <Btn1Motion>: extend-adjust() <Btn1Up>: extend-end(PRIMARY, CUT_BUFFER0) <Btn2Down>: insert-selection(PRIMARY, CUT_BUFFER0) <Btn3Down>: extend-start() <Btn3Motion>: extend-adjust() <Btn3Up>: extend-end(PRIMARY, CUT_BUFFER0)
`type`	unsigned_char
`useStringInPlace`	Boolean
`width`	Dimension
`wrap`	unsigned_char
`x`	Position
y	Position

Constraint Resource Set

Name	X Type	Default
`bottom`	unsigned_char	0
`fromHoriz`	Widget	NULL
`fromVert`	Widget	NULL
`horizDistance`	int	0
`vertDistance`	int	0

left	unsigned_char	2
resizable	Boolean	false
top	unsigned_char	0

Methods

Name	Parameters	Description
move	x, y, horizDistance, vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize this widget with the mouse and sends its x and y location and new width and height to the dialogue.
send	textBuffer	This method is returned in response to the send_buffer flag being set to true by the dialogue or to a translation table action.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).
top		This method is sent to the dialogue when the widget is raised.
bottom		This method is sent to the dialogue when the widget is lowered.

XawTextentry

Serpent Name

XawTextentry

include_file: X11/Xaw/AsciiText.h
class: asciiTextWidgetClass
widget_type: widget

Description

The XawTextentry widget is similar to the Text widget except a carriage return activates the send method. The XawTextentry widget was created for use in forms so that the text focus can shift automatically from item to item via carriage return.

Attributes

Serpent

Name	X Type	Default
after	Widget	
allowUserMove	Boolean	false
allowUserResize	Boolean	false
toBottom	bottom	
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
sendBuffer	Boolean	
toTop	top	
widget	int	

Text Widget Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
autoFill	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	
bottomMargin	Position	
callback	CallbackList	
colormap	Colormap	
cursor	Cursor	
dataCompression	Boolean	
depth	int	
destroyCallback	CallbackList	
displayCaret	Boolean	
displayNonprinting	Boolean	
displayPosition	int	
echo	Boolean	
editType	unsigned_char	2
font	FontStruct	6x13
foreground	Pixel	
height	Dimension	
insensitiveBorder	Pixmap	
insertPosition	int	
leftMargin	Position	
length	int	
mappedWhenManaged	Boolean	
pieceSize	int	
rightMargin	Position	
screen	Screen	
scrollHorizontal	unsigned_char	
scrollVertical	unsigned_char	

selectTypes		TextSelectType_star	
sensitive		Boolean	
string		String	textBuffer
textSink		Widget	
textSource		Widget	
topMargin		Position	
translations	Translations:	Ctrl<Key>A: beginning-of-line() Ctrl<Key>B: backward-character() Ctrl<Key>D: delete-next-character() Ctrl<Key>E: end-of-line() Ctrl<Key>F: forward-character() Ctrl<Key>G: multiply(Reset) Ctrl<Key>H: delete-previous-character() Ctrl<Key>J: newline-and-indent() Ctrl<Key>K: kill-to-end-of-line() Ctrl<Key>L: redraw-display() Ctrl<Key>M: newline() Ctrl<Key>N: next-line() Ctrl<Key>O: newline-and-backup() Ctrl<Key>P: previous-line() Ctrl<Key>R: search(backward) Ctrl<Key>S: search(forward) Ctrl<Key>T: transpose-characters() Ctrl<Key>U: multiply(4) Ctrl<Key>V: next-page() Ctrl<Key>W: kill-selection() Ctrl<Key>Y: insertselection(CUT_BUFFER1) Ctrl<Key>Z: scroll-one-line-up() Meta<Key>B: backward-word() Meta<Key>F: forward-word() Meta<Key>I: insert-file() Meta<Key>K: kill-to-end-of-paragraph() Meta<Key>Q: form-paragraph() Meta<Key>V: previous-page() Meta<Key>Y: insertselection (PRIMARY CUT_BUFFER0) Meta<Key>Z: scroll-one-line-down() Meta<Key>d: delete-next-word() Meta<Key>D: kill-word() Meta<Key>H: backward-kill-word() Meta<Key>\\<: beginning-of-file() Meta<Key>\\>: end-of-file() Meta<Key>]: forward-paragraph() Meta<Key>[: backward-paragraph() ~Shift Meta<Key>Delete: delete previous-word() Shift Meta<Key>Delete: backward-kill word() ~Shift Meta<Key>BackSpace: delete-previous-word()	

	Shift Meta<Key>BackSpace: backward-kill-word() Shift<Key>Tab: tab() <Key>Right: forward-character() <Key>Left: backward-character() <Key>Down: next-line() <Key>Up: previous-line() <Key>Delete: delete-previous-character() <Key>BackSpace: delete-previous-character() <Key>Linefeed: newline-and-indent() <Key>Return: send() <Key>: insert-char() <FocusIn>: focus-in() <FocusOut>: focus-out() Ctrl<Btn1Down>,<Btn1Up>: top() Ctrl<Btn2Down>,<Btn2Up>: bottom() Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <Btn1Down>: select-start() <Btn1Motion>: extend-adjust() <Btn1Up>: extend-end(PRIMARY, CUT_BUFFER0) <Btn2Down>: insert-selection(PRIMARY, CUT_BUFFER0) <Btn3Down>: extend-start() <Btn3Motion>: extend-adjust() <Btn3Up>: extend-end(PRIMARY, CUT_BUFFER0)
type	unsigned_char
useStringInPlace	Boolean
width	Dimension
wrap	unsigned_char
x	Position
y	Position

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_cha	0
fromHoriz	Widge	NULL
fromVert	Widge	NULL
horizDistance	in	0
vertDistance	in	0

left	unsigned_cha	2
resizable	Boolean	false
right	unsigned_cha	2
top	unsigned_cha	0

Methods

Name	Parameters	Description
move	x, y, horizDistance, vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
send	textBuffer	This method is returned in response to the send_buffer flag being set to true by the dialogue or to a translation table action.
tab		This method is sent to the dialogue in response to a shifted tab.
top		This method is sent to the dialogue when the widget is raised.
bottom		This method is sent to the dialogue when the widget is lowered.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XawToggleButton

Serpent Name

XawToggleButton

include_file: X11/Xaw/Toggle.h
class: toggleWidgetClass
widget_type: widget

Description

The Toggle widget is an area, often rectangular, containing a text label or bitmap image. This widget maintains a Boolean state (e.g., True/False or On/Off) and changes state whenever it is selected. When the pointer is on the button, the button may become highlighted by a rectangle around its perimeter. This highlighting indicates that the button is ready for selection. When pointer button 1 is pressed and released, the Toggle widget indicates that it has changed state by reversing its foreground and background colors, and its notify action is invoked, calling all functions on its callback list. If the pointer is moved out of the widget before the button is released, the widget reverts to its normal foreground and background colors, and releasing the button has no effect. This allows the user to cancel an action.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
toBottom	bottom	
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	NULL
toTop	top	
selectedX	Position	0
selectedY	Position	0

widget	int

Command Widget Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	white
backgroundPixmap	Pixmap	
bitmap	Pixmap	
borderColor	Pixel	black
borderPixmap	Pixmap	
borderWidth	Dimension	1
callback	CallbackList	Six_callback
colormap	Colormap	
cornerRoundPercent	Dimension	
cursor	Cursor	
depth	int	
destroyCallback	CallbackList	
font	XFontStruct	6x13
foreground	Pixel	black
height	Dimension	
highlightThickness	Dimension	
insensitiveBorder	Pixmap	
internalHeight	Dimension	2
internalWidth	Dimension	4
justify	unsigned_char	1
label	String	
mappedWhenManaged	Boolean	true
radioData	caddr_t	
radioGroup	Widget	
resize	Boolean	
screen	Screen	
sensitive	Boolean	true
shapeStyle	unsigned_char	
state	Boolean	

translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize(Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <Btn1Down>,<Btn1Up>: toggle() notify () <EnterWindow>: highlight(Always) <LeaveWindow>: unhighlight()
width		Dimension
x		Position
y		Position

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_char	0
fromHoriz	Widget	NULL
fromVert	Widget	NULL
horizDistance	int	0
vertDistance	int	0
left	unsigned_char	2
resizable	Boolean	false
right	unsigned_char	2
top	unsigned_char	0

Methods

Name	Parameters	Description
move	x, y, horizDistance, vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
notify		This method is sent to the dialogue in response to a user event (typically a Btn1Down).

`pick`	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XawTopLevelShell

Serpent Name XawTopLevelShell

include_file: X11/Shell.h
class: topLevelShellWidgetClass
widget_type: shell

Description Used for normal top level windows (for example, any additional top level widgets an application needs).

Attributes

TopLevelShell

Name	X Type
iconic	Boolean
iconName	String
iconNameEncoding	unsigned_char

WMShell

Name	X Type
baseHeight	int
baseWidth	int
heightInc	int
iconMask	Pixmap
iconPixmap	Pixmap
iconWindow	Window
iconX	int
iconY	int
initialState	int
input	Boolean

maxAspectX	int
maxAspectY	int
maxHeight	int
maxWidth	int
minAspectX	int
minAspectY	int
minHeight	int
minWidth	int
title	char_star
titleEncoding	unsigned_char
transient	Boolean
waitForWm	Boolean
widthInc	int
windowGroup	XID
wmTimeout	int

Shell Resource Set

Name	X Type
allowShellResize	Boolean
createPopupChildProc	Boolean
geometry	caddr_t
overrideRedirect	Boolean
popdownCallback	caddr_t
popupCallback	caddr_t
saveUnder	Boolean

XawViewport

Serpent Name

XawViewport

/include_file: X11/Xaw/Viewport.h
widget_type: widget

Description

The XawViewport widget consists of a frame window, one or two scrollbars, and an inner window. The size of the frame window is determined by the viewing size of the data that is to be displayed and the dimensions to which the XawViewport is created. The inner window is the full size of the data that is to be displayed and is clipped by the frame window. The XawViewport widget controls the scrolling of the data directly. No application callbacks are required for scrolling.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
toBottom	bottom	
focus	focus	
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
toTop	top	
widget	int	

Paned Widget Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
allowHoriz	Boolean	
allowVert	Boolean	
backgroun	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	
children	WidgetList	
colormap	Colormap	
depth	int	
destroyCallback	CallbackList	
forceBars	Boolean	
height	Dimension	
mappedWhenManaged	Boolean	
numChildren	Cardinal	
screen	Screen	
sensitive	Boolean	
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Ctrl<Btn1Down>,<Btn1Up>: top() Ctrl<Btn2Down>,<Btn2Up>: bottom() Shift<Btn3Down>,<Btn3Motion>: move()d Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()	
useBottom	Boolean	
useRight	Boolean	
width	Dimension	
x	Position	
y	Position	

Constraint Resource Set

Name	X Type	Default
bottom	unsigned_cha	0
fromHoriz	Widge	NULL
fromVert	Widge	NULL
horizDistance	in	0
vertDistance	in	0
left	unsigned_cha	2
resizable	Boolea	false
right	unsigned_cha	2
top	unsigned_cha	0

Methods

Name	Parameters	Description
move x, y,	horizDistance, vertDistance	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
top		This method is sent to the dialogue when the widget is raised.
bottom		This method is sent to the dialogue when the widget is lowered.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse, and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

Appendix H Motif Widget Set

XmArrowButton

Serpent Name

XmArrowButton

include_file: Xm/ArrowB.h
class: ArrowButtonWidgetClass
widget_type: widget

Description

The XmArrowButton widget consists of a directional arrow surrounded by a border shadow. When it is selected, the shadow moves to give the appearance that XmArrowButton has been pressed in. When XmArrowButton is not selected, the shadow moves to give the appearance that XmArrowButton has been released, or is out.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	
activateCallback	CallbackList	Six_callback
armCallback	CallbackList	
arrowDirection	unsigned_char	ARROW_UP
disarmCallback	CallbackList	

Primitive Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	

bottomShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightOnEnter	Boolean	false
highlightPixmap	Pixmap	
highlightThickness	short	0
shadowThickness	short	2
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
traversalOn	Boolean	false
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitived	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() <Btn1Down>: Arm()	

<Btn1Up>: Activate() Disarm()
<Key>Return: ArmAndActivate()
<Key>space: ArmAndActivate()
<EnterWindow>: Enter()
<LeaveWindow>: Leave()
Shift<Btn2Down>,<Btn2Motion>: resize()
Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>:
resize()
Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize()
Shift<Btn3Down>,<Btn3Motion>: move()
Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move()
Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
notify		This method is sent to the dialogue in response to a user event (typically a Btn1Down).
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmBulletinBoard

Serpent Name XmBulletinBoard

include_file: Xm/BulletinB.h
class: BulletinBoardWidgetClass
widget_type: widget

Description XmBulletinBoard is a composite widget that provides simple geometry management for children widgets. It does not force positioning on its children, but can be set to reject geometry requests that would result in overlapping children. XmBulletinBoard is the base widget for most dialogue widgets and is also used as a general container widget.

Attributes
Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
focus	focus	
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

BulletinBoard Resource Set

Name	X Type	Default
allowOverlap	Boolean	
autoUnmanage	Boolean	
buttonFontList	FontList	

cancelButton	Widget
defaultButton	Widget
defaultPosition	Boolean
dialogStyle	unsigned_char
dialogTitle	String
focusCallback	CallbackList
labelFontList	FontList
mapCallback	CallbackList
marginHeight	short
marginWidth	short
noResize	Boolean
resizePolicy	unsigned_char
shadowType	unsigned_char
stringDirection	StringDirection
textFontList	FontList
textTranslations	Translations
unmapCallback	CallbackList

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type	Default
insertPosition	OrderProc	
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() <Btn1Down>,<Btn1Up>: select() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()	

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.

`select`	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue.
`pick`	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmCascadeButton

Serpent Name XmCascadeButton

include_file: Xm/CascadeB.h
class: CascadeButtonWidgetClass
widget_type: widget

Description The XmCascadeButton widget links two MenuPanes or a MenuBar to a MenuPane. It is used in menu systems and must have a RowColumn parent with its rowColumnType resource set to MENU_BAR, MENU_POPUP, or MENU_PULLDOWN. It is the only widget that may have a pulldown MenuPane attached to it as a submenu. The submenu is displayed when this widget is activated within a MenuBar, a PopupMenu, or a PulldownMenu. Its visuals can include a label or pixmap and a cascading indicator when it is in a Popup or Pulldown MenuPane; when it is in a MenuBar, it can include only a label or a pixmap.

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

ArrowButton

Name	X Type	Default
activateCallback	CallbackList	Six_callback
cascadePixmap	Pixmap	
cascadingCallback	CallbackList	

mappingDelay	int
subMenuId	Widget

Label Resource Set

Name	X Type
accelerator	String
acceleratorText	String
alignment	unsigned_char
fontList	FontList
labelInsensitivePixmap	Pixmap
labelPixmap	Pixmap
labelString	String
labelType	unsigned_char
marginBottom	short
marginHeight	short
marginLeft	short
marginRight	short
marginTop	short
marginWidth	short
mnemonic	char
recomputeSize	Boolean
stringDirection	StringDirection

Primitive Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightOnEnter	Boolean	false
highlightPixmap	Pixmap	
highlightThickness	short	0
shadowThickness	short	2

topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
traversalOn	Boolean	false
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	<BtnDown>: MenuBarSelect() <EnterWindow>: MenuBarEnter() <LeaveWindow>: MenuBarLeave() <BtnUp>: DoSelect() <Key>Return: KeySelect() <Key>Escape: CleanupMenuBar() Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,t<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
notify		This method is sent to the dialogue in response to a user event (typically a Btn1Down).
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse, and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmCommand

Serpent Name XmCommand

include_file: Xm/Command.h
class: CommandWidgetClass
widget_type: widget

Description The XmCommand widget is a special-purpose, composite widget for command entry that provides a built-in mechanism for displaying command histories. XmCommand includes a field for input from the command line, a command line prompt, and a region for displaying the command history list.

Attributes
Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

Command

Name	X Type	Default
command	String	
commandChangedCallback	CallbackList	
commandEnteredCallback	CallbackList	
historyItems	StringTable	
historyItemCount	int	

historyMaxItems	int
historyVisibleItemCount	int
promptString	String

SelectionBox Resource Set

Name	X Type	Default
applyCallback	CallbackList	
applyLabelString	String	
cancelCallback	CallbackList	
cancelLabelString	String	
dialogType	unsigned_char	
helpLabelString	String	
listItemCount	int	
listItems	StringList	
listLabelString	String	
listVisibleItemCount	int	
minimizeButtons	Boolean	
mustMatch	Boolean	
noMatchCallback	CallbackList	
okCallback	CallbackList	
okLabelString	String	
selectionLabelString	String	
textAccelerators	Translations	
textColumns	int	
textValue	String	

BulletinBoard Resource Set

Name	X Type	Default
allowOverlap	Boolean	
autoUnmanage	Boolean	
buttonFontList	FontList	
cancelButton	Widget	
defaultButton	Widget	
defaultPosition	Boolean	

dialogStyle	unsigned_char
dialogTitle	String
focusCallback	CallbackList
labelFontList	FontList
mapCallback	CallbackList
marginHeight	short
marginWidth	short
noResize	Boolean
resizePolicy	unsigned_char
shadowType	unsigned_char
stringDirection	StringDirection
textFontList	FontList
textTranslations	Translations
unmapCallback	CallbackList

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize this widget with the mouse and sends its x and y location and new width and height to the dialogue.

pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmDrawingArea

Serpent Name

XmDrawingArea

include_file:Xm /DrawingA.h
class: DrawingAreaWidgetClass
widget_type: widget

Description

The XmDrawingArea widget is an empty widget that is easily adaptable to a variety of purposes. It does no drawing and defines no behavior except for invoking callbacks. Callbacks notify the application when graphics need to be drawn (exposure events or widget resize) and when the widget receives input from the keyboard or mouse. Applications are responsible for defining appearance and behavior as needed in response to XmDrawingArea callbacks.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean false	
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

DrawingArea

Name	X Type	Default
exposeCallback	CallbackList	
inputCallback	CallbackList	
marginHeight	short	

marginWidth	short
resizeCallback	CallbackList
resizePolicy	unsigned_char

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	

height		Dimension
mappedWhenManaged		Boolean
screen		Screen
sensitive		Boolean
width		Dimension
x		Position
y		Position
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <Btn1Down>: Arm() <Btn1Up>: Activate() <EnterWindow>: Enter() <FocusIn>: FocusIn()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmDrawnButton

Serpent Name XmDrawnButton

include_file: Xm/DrawnB.h
class: DrawnButtonWidgetClass
widget_type: widget

Description The XmDrawnButton widget consists of an empty widget window surrounded by a shadow border. It provides the application developer with a graphics area in which PushButton input semantics may be used.

Attributes
Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

DrawnButton

Name	X Type	Default
activateCallback	CallbackList	Sixcallback
armCallback	CallbackList	
disarmCallback	CallbackList	
exposeCallback	CallbackList	
pushButtonEnabled	Boolean	

resizeCallback	CallbackList
shadowType	unsigned_char

Resource Set

Name	X Type
accelerator	String
acceleratorText	String
alignment	unsigned_char
fontList	FontList
labelInsensitivePixmap	Pixmap
labelPixmap	Pixmap
labelString	String
labelType	unsigned_char
marginBottom	short
marginHeight	short
marginLeft	short
marginRight	short
marginTop	short
marginWidth	short
mnemonic	char
recomputeSize	Boolean
stringDirection	StringDirection

Primitive Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightOnEnter	Boolean	false
highlightPixmap	Pixmap	
highlightThickness	short	0
shadowThickness	short	2

topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
traversalOn	Boolean	false
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <Btn1Down>: Arm() <Btn1Up>: Activate() Disarm() <Key>Return: ArmAndActivate() <Key>space: ArmAndActivate() <EnterWindow>: Enter() <LeaveWindow>: Leave()	

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
notify		This method is sent to the dialogue in response to a user event (typically a Btn1Down).
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmErrorDialog

Serpent Name XmErrorDialog

include_file: XM/MessageB.h
class: MessageBoxWidgetClass
widget_type: widget

Description The XmErrorDialog widget is a MessageBox created with a convenience routine. This dialogue is used to warn a user about problem situations. The dialogue box comes with three buttons: OK, Cancel, and Help. The default symbol is an octagon with a diagonal slash.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
deactivate	Boolean	false
isComposite	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	
MessageBox	CallbackList	
cancelLabelString	String	
defaultButtonType	unsigned_char	
dialogType	unsigned_char	
helpLabelString	String	
messageAlignment	unsigned_char	

messageString	String
minimizeButtons	Boolean
okCallback	CallbackList
okLabelString	String
symbolPixmap	Pixmap

BulletinBoard Resource Set

Name	X Type
allowOverlap	Boolean
autoUnmanage	Boolean
buttonFontList	FontList
cancelButton	Widget
defaultButton	Widget
defaultPosition	Boolean
dialogStyle	unsigned_char
dialogTitle	String
focusCallback	CallbackList
labelFontList	FontList
mapCallback	CallbackList
marginHeight	short
marginWidth	short
noResize	Boolean
resizePolicy	unsigned_char
shadowType	unsigned_char
stringDirection	StringDirection
textFontList	FontList
textTranslations	Translations
unmapCallback	CallbackList

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
foreground	Pixel	

helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type	
insertPosition	OrderProc	
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	<EnterWindow>Enter() <FocusIn>: FocusIn() Shift<Btn1Down>,<Btn1Up>: pick() <Btn1Down>: Arm() <Btn1Up>: Activate() <Key>F1: Help() <Key>Return: Return() <Key>KP_Enter: Return()

Shift<Btn2Down>,<Btn2Motion>: resize()
Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize()
Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize()
Shift<Btn3Down>,<Btn3Motion>: move()
Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move()
Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmFileSelectionBox

Serpent Name

XmFileSelectionBox

include_file: Xm/FileSB.h
class: FileSelectionBoxWidgetClass
widget_type: widget
check_routine: check_FSB

Description

The XmFileSelectionBox widget traverses directories, views the files in them, and then selects a file. An XmFileSelectionBox widget has four main areas:

- a directory mask that includes a filter label and a directory mask input field used to specify the directory that is to be examined
- a scrollable list of file names
- a text input field for directly typing in a file name
- a group of PushButtons: OK, Filter, Cancel, and Help

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

FileSelectionBox

Name	X Type
dirMask	String

dirSpec	String
fileSearchProc	Proc
filterLabelString	String
listUpdated	Boolean

SelectionBox Resource Set

Name	X Type
applyCallback	CallbackList
applyLabelString	String
cancelCallback	CallbackList
cancelLabelString	String
dialogType	unsigned_char
helpLabelString	String
listItemCount	int
listItems	StringList
listLabelString	String
listVisibleItemCount	int
minimizeButtons	Boolean
mustMatch	Boolean
noMatchCallback	CallbackList
okCallback	CallbackList
okLabelString	String
selectionLabelString	String
textAccelerators	Translations
textColumns	int
textValue	String

BulletinBoard Resource Set

Name	X Type
allowOverlap	Boolean
autoUnmanage	Boolean
buttonFontList	FontList
cancelButton	Widget
defaultButton	Widget

defaultPosition	Boolean
dialogStyle	unsigned_char
dialogTitle	String
focusCallback	CallbackList
labelFontList	FontList
mapCallback	CallbackList
marginHeight	short
marginWidth	short
noResize	Boolean
resizePolicy	unsigned_char
shadowType	unsigned_char
stringDirection	StringDirection
textFontList	FontList
textTranslations	Translations
unmapCallback	CallbackList

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type	Default
insertPosition	OrderProc	

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	<EnterWindow>:Enter() <FocusIn>: FocusIn() Shift<Btn1Down>,<Btn1Up>: pick() <Btn1Down>: Arm() <Btn1Up>: Activate() <Key>F1: Help() <Key>Return: Return() <Key>KP_Enter: Return() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.

`resize`	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
`pick`	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmForm

Serpent Name

XmForm

include_file: Xm/Form.h
class: FormWidgetClass
widget_type: widget

Description

The XmForm widget is a container widget with no input semantics of its own. Constraints are placed on children of XmForm to define attachments for each of the child's four sides. These attachments can be to XmForm, to another child widget or gadget, to a relative position within XmForm, or to the initial position of the child. The attachments determine the layout behavior of XmForm when resizing occurs.

Attributes

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
focus	focus	
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

Form

Name	X Type	Default
fractionBase	int	
horizontalSpacing	int	
rubberPositioning	Boolean	
verticalSpacing	int	

Form Constraint

Name	X Type
allowOverlap	Boolean
bottomAttachment	unsigned_char
bottomOffset	int
bottomPosition	int
bottomWidget	Widget
leftAttachment	unsigned_char
leftOffset	int
leftPosition	int
leftWidget	Widget
resizable	Boolean
rightAttachment	unsigned_char
rightOffset	int
rightPosition	int
rightWidget	Widget
topAttachment	unsigned_char
topOffset	int
topPosition	int
topWidget	Widget

BulletinBoard Resource Set

Name	X Type
allowOverlap	Boolean
autoUnmanage	Boolean
buttonFontList	FontList
cancelButton	Widget
defaultButton	Widget
defaultPosition	Boolean
dialogStyle	unsigned_char
dialogTitle	String
focusCallback	CallbackList
labelFontList	FontList
mapCallback	CallbackList

marginHeight	short
marginWidth	short
noResize	Boolean
resizePolicy	unsigned_char
shadowType	unsigned_char
stringDirection	StringDirection
textFontList	FontList
textTranslations	Translations
unmapCallback	CallbackList

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	

borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() <Btn1Down>,<Btn1Up>: select() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	Parameter	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmFrame

Serpent Name

XmFrame

include_file: Xm/Frame.h
class: FrameWidgetClass
widget_type: widget

Description

The XmFrame widget is a very simple manager used to enclose a single child in a border drawn by XmFrame. It uses the Manager class resources to draw borders and performs geometry management such that its size will always match its child's size plus the margins defined for it.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

Frame

Name	X Type
marginWidth	short
marginHeight	short
shadowType	unsigned_char

Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	

width		Dimension
x		Position
y		Position
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>:move() <EnterWindow>: Enter() <FocusIn>: FocusIn() <Btn1Down>: Arm() <Btn1Up>: Activate()

Methods

Move	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmInformationDialog

Serpent Name

XmInformationDialog

include_file: Xm/MessageB.h
class: xmMessageBoxWidgetClass
widget_type: widget

Description

The XmInformationDialog widget is a XmMessageBox created with a convenience routine. This dialogue is used to provide a user with information. The dialogue box comes with three buttons: OK, Cancel, and Help. The default symbol is a lower case i.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
deactivate	Boolean	false
isComposite	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	
MessageBox	CallbackList	
cancelLabelString	String	
defaultButtonType	unsigned_char	
dialogType	unsigned_char	
helpLabelString	String	
messageAlignment	unsigned_char	
messageString	String	

minimizeButtons	Boolean
okCallback	CallbackList
okLabelString	String
symbolPixmap	Pixmap

BulletinBoard Resource Set

Name	X Type
allowOverlap	Boolean
autoUnmanage	Boolean
buttonFontList	FontList
cancelButton	Widget
defaultButton	Widget
defaultPosition	Boolean
dialogStyle	unsigned_char
dialogTitle	String
focusCallback	CallbackList
labelFontList	FontList
mapCallback	CallbackList
marginHeight	short
marginWidth	short
noResize	Boolean
resizePolicy	unsigned_char
shadowType	unsigned_char
stringDirection	StringDirection
textFontList	FontList
textTranslations	Translations
unmapCallback	CallbackList

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
foreground	Pixel	
helpCallback	CallbackList	

highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
unitType	unsigned_char	PIXELS
userData	caddr_t	

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	<EnterWindow>: Enter() <FocusIn>: FocusIn() Shift<Btn1Down>,<Btn1Up>: pick() <Btn1Down>: Arm()

<Btn1Up>: Activate()
<Key>F1: Help()
<Key>Return: Return()
<Key>KP_Enter: Return()
Shift<Btn2Down>,<Btn2Motion>: resize()
Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize()
Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize()
Shift<Btn3Down>,<Btn3Motion>: move()
Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move()
Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	X Type	Default
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).
notify		This method is sent to the dialogue in response to a user event (typically a Btn1Down).

XmLabel

Serpent Name

XmLabel

include_file: Xm/Label.h
class: LabelWidgetClass
widget_type: widget

Description

The XmLabel widget is an instantiable widget and is also used as a superclass for other button widgets, such as PushButton and ToggleButton. The XmLabel widget does not accept any button or key input, and the help callback is the only callback defined. XmLabel also receives enter and leave events. It can contain either text or a Pixmap; its text is a compound string.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

Label Resource Set

Name	X Type
accelerator	String
acceleratorText	String
alignment	unsigned_char
fontList	FontList

labelInsensitivePixmap	Pixmap
labelPixmap	Pixmap
labelString	String
labelType	unsigned_char
marginBottom	short
marginHeight	short
marginLeft	short
marginRight	short
marginTop	short
marginWidth	short
mnemonic	char
recomputeSize	Boolean
stringDirection	StringDirection

Primitive Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightOnEnter	Boolean	false
highlightPixmap	Pixmap	
highlightThickness	short	0
shadowThickness	short	2
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
traversalOn	Boolean	false
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Core Resource Set

accelerators	Accelerators
ancestorSensitive	Boolean

background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <EnterWindow>: Enter() <LeaveWindow>: Leave()	

Methods

Name	Parameter	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmList

Serpent Name

XmList

include_file: Xm/List.h
class: ListWidgetClass
widget_type: widget

Description

The XmList widget allows a user to select one or more items from a group of choices. Items are selected from the list in a variety of ways, with both the pointer and the keyboard. XmList operates on an array of strings that are defined by the application. Each string becomes an item in XmList, with the first string becoming the item in position 1, the second string becoming the item in position 2, and so on.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	parentWidget
selectedX	Position	0
selectedY	Position	0
widget	int	

List

Name	X Type
automaticSelection	Boolean
browseSelectionCallback	CallbackList
defaultActionCallback	CallbackList
doubleClickInterval	int
extendedSelectionCallback	CallbackList

fontList	FontList
itemCount	int
items	StringTable
listMarginHeight	Dimension
listMarginWidth	Dimension
listSpacing	short
multipleSelectionCallback	CallbackList
selectedItemCount	int
selectedItems	StringTable
selectionPolicy	unsigned_char
singleSelectionCallback	CallbackList
stringDirection	StringDirection
visibleItemCount	int

ScrolledList

Name	X Type
horizontalScrollBar	Widget
listSizePolicy	unsigned_char
scrollBarDisplayPolicy	unsigned_char
scrollBarPlacement	unsigned_char
scrolledWindowMarginHeight	Dimension
scrolledWindowMarginWidth	Dimension
spacing	Dimension
verticalScrollBar	Widget

Primitive Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightOnEnter	Boolean	false
highlightPixmap	Pixmap	

highlightThickness	short	0
shadowThickness	short	2
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
traversalOn	Boolean	false
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>:move() Button1<Motion>: ListButtonMotion() Shift Ctrl ~Meta<Btn1Down>: ListShiftCtrlSelect()	

Shift Ctrl ~Meta<Btn1Up>: ListShiftCtrlUnSelect()
Shift Ctrl ~Meta<KeyDown>space: ListKbdShiftCtrlSelect()
Shift Ctrl ~Meta<KeyUp>space: ListKbdShiftCtrlUnSelect()
Shift Ctrl ~Meta<KeyDown>Select: ListKbdShiftCtrlSelect()
Shift Ctrl ~Meta<KeyUp>Select: ListKbdShiftCtrlUnSelect()
Shift ~Ctrl ~Meta<Btn1Down>: ListShiftSelect()
Shift ~Ctrl ~Meta<Btn1Up>: ListShiftUnSelect()
Shift ~Ctrl ~Meta<KeyDown>space: ListKbdShiftSelect()
Shift ~Ctrl ~Meta<KeyUp>space: ListKbdShiftUnSelect()
Shift ~Ctrl ~Meta<KeyDown>Select: ListKbdShiftSelect()
Shift~Ctrl ~Meta<KeyUp>Select: ListKbdShiftUnSelect()
Ctrl ~Shift ~Meta<Btn1Down>: ListCtrlSelect()
Ctrl ~Shift ~Meta<Btn1Up>: ListCtrlUnSelect()
Ctrl ~Shift ~Meta<KeyDown>space: ListKbdCtrlSelect()
Ctrl ~Shift ~Meta<KeyUp>space: ListKbdCtrlUnSelect()
Ctrl ~Shift ~Meta<KeyDown>Select: ListKbdCtrlSelect()
Ctrl ~Shift ~Meta<KeyUp>Select: ListKbdCtrlUnSelect()
~Shift ~Ctrl ~Meta<Btn1Down>: ListElementSelect()
~Shift ~Ctrl ~Meta<Btn1Up>: ListElementUnSelect()
~Shift ~Ctrl ~Meta<KeyDown>space: ListKbdSelect()
~Shift ~Ctrl ~Meta<KeyUp>space: ListKbdUnSelect()
~Shift ~Ctrl ~Meta<KeyDown>Select: ListKbdSelect()
~Shift ~Ctrl ~Meta<KeyUp>Select: ListKbdUnSelect()
Shift Ctrl ~Meta<Key>Up: ListShiftCtrlPrevElement()
Shift Ctrl ~Meta<Key>Down: ListShiftCtrlNextElement()
Shift ~Ctrl ~Meta<Key>Up: ListShiftPrevElement()
Shift ~Ctrl ~Meta<Key>Down: ListShiftNextElement()
~Shift Ctrl ~Meta<Key>Up: ListCtrlPrevElement()
~Shift Ctrl ~Meta<Key>Down: ListCtrlNextElement()
~Shift ~Ctrl ~Meta<Key>Up: ListPrevElement()
~Shift ~Ctrl ~Meta<Key>Down: ListNextElement()
<Enter>: ListEnter()
<Leave>: ListLeave()
<FocusIn>: ListFocusIn()
<FocusOut>: ListFocusOut()
<Unmap>: PrimitiveUnmap()
Shift<Key>Tab: PrimitivePrevTabGroup()
Ctrl<Key>Tab: PrimitiveNextTabGroup()
<Key>Tab: PrimitiveNextTabGroup()
<Key>Home: PrimitiveTraverseHome()

Methods

Name	Parameter	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.

resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmMainWindow

Serpent Name

xmMainWindow

include_file: Xm/MainW.h
class: MainWindowWidgetClass
widget_type: widget
check_routine: check_MainW

Description

The XmMainWindow widget provides a standard layout for the primary window of an application. This layout includes an MenuBar, an CommandWindow, a work region, and ScrollBars. Any or all of these areas are optional. The work region and prog in the XmMainWindow behave identically to the work region and ScrollBars in the ScrolledWindow widget. (The user can think of the XmMainWindow as an extended ScrolledWindow with an optional MenuBar and optional CommandWindow.)

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

MainWindow

Name	X Type
commandWindow	Widget

mainWindowMarginHeight	Dimension
mainWindowMarginWidth	Dimension
menuBar	Widget
showSeparator	Boolean

ScrolledWindow Resource Set

Name	X Type
clipWindow	Widget
horizontalScrollBar	Widget
scrollBarDisplayPolicy	unsigned_char
scrollBarPlacement	unsigned_char
scrolledWindowMarginHeight	Dimension
scrolledWindowMarginWidth	Dimension
scrollingPolicy	unsigned_char
spacing	int
verticalScrollBar	Widget
visualPolicy	unsigned_char
workWindow	Widget

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type	Default
insertPosition	OrderProc	

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
Translations Translations	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()	

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.

resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmMenubar

Serpent Name

XmMenubar

include_file: Xm/RowColumn.h
class: RowColumnWidgetClass
widget_type: widget

Description

The XmMenuBar widget is a specially configured RowColumn widget created with a convenience routine. It is used to build a pulldown menu.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

RowColumn

Name	X Type
adjustLast	Boolean
adjustMargin	Boolean
entryAlignment	unsigned_char
entryBorder	short
entryCallback	CallbackList
entryClass	WidgetClass

isAligned	Boolean
isHomogeneous	Boolean
labelString	String
mapCallback	CallbackList
marginHeight	Dimension
marginWidth	Dimension
menuAccelerator	String
menuHelpWidget	Widget
menuHistory	Widget
mnemonic	char
numColumns	short
orientation	unsigned_char
packing	unsigned_char
popupEnabled	Boolean
radioAlwaysOne	Boolean
radioBehavior	Boolean
resizeHeight	Boolean
resizeWidth	Boolean
rowColumnType	unsigned_char
spacing	short
subMenuId	Widget
unmapCallback	CallbackList
whichButton	unsigned_int

RowColumn Special Menu

Name	X Type
menuCursor	String

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	

highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
Translations	Translations	<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move()

Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmMenuShell

Serpent Name XmMenuShell

include_file: Xm/MenuShell.h
class: MenuShellWidgetClass
widget_type: override

Description The XmMenuShell widget is a custom OverrideShell widget. An OverrideShell widget bypasses the window manager when displaying itself. It is designed specifically to contain Popup or Pulldown MenuPanes.

Attributes
Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
parent	Widget	
method	MethodName	

Shell

Name	X Type	Default
Name		
allowShellResize	Boolean	
createPopupChildProc	Boolean	
geometry	caddr_t	
overrideRedirect	Boolean	
popdownCallback	CallbackList	
popupCallback	CallbackList	
saveUnder	Boolean	

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations Translations:	Shift<Btn2Down>,<Btn2Motion>:resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <BtnDown>: ClearTraversal() <Key>Escape: MenuShellPopdownDone() <BtnUp>: MenuShellPopdownDone()	

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.

XmMessageBox

Serpent Name

XmMessageBox

include_file: Xm/MessageB.h
class: MessageBoxWidgetClass
widget_type: widget

Description

The XmMessageBox widget is a dialogue class widget used for creating simple message dialogues. Convenience dialogues based on XmMessageBox are provided for several common interaction tasks, including giving information, asking questions, and reporting errors.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
isComposite	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

MessageBox

Name	X Type
cancelCallback	CallbackList
cancelLabelString	String
defaultButtonType	unsigned_char
dialogType	unsigned_char

helpLabelString	String
messageAlignment	unsigned_char
messageString	String
minimizeButtons	Boolean
okCallback	CallbackList
okLabelString	String
symbolPixmap	Pixmap

BulletinBoard Resource Set

Name	X Type
allowOverlap	Boolean
autoUnmanage	Boolean
buttonFontList	FontList
cancelButton	Widget
defaultButton	Widget
defaultPosition	Boolean
dialogStyle	unsigned_char
dialogTitle	String
focusCallback	tCallbackList
labelFontList	FontList
mapCallback	CallbackList
marginHeight	short
marginWidth	short
noResize	Boolean
resizePolicy	unsigned_char
shadowType	unsigned_char
stringDirection	StringDirection
textFontList	FontList
textTranslations	Translations
unmapCallback	CallbackList

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	

bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	

translations	Translations:	<EnterWindow>:Enter() <FocusIn>: FocusIn() <Btn1Down>: Arm() <Btn1Up>: Activate() <Key>F1: Help() <Key>Return: Return() <Key>KP_Enter: Return() Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmMessageDialog

Serpent Name

XmMessageDialog

include_file Xm/MessageB.h
class: xmMessageBoxWidgetClass
widget_type: widget

Description

The XmMessageDialog widget is a MessageBox created with a convenience routine. This convenience routine creates a MessageBox parented to a dialogue shell.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
deactivate	Boolean	false
isComposite	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	Widget	
MessageBox	CallbackList	
cancelLabelString	String	
defaultButtonType	unsigned_char	
dialogType	unsigned_char	
helpLabelString	String	
messageAlignment	unsigned_char	
messageString	String	
minimizeButtons	Boolean	

okCallback	CallbackList
okLabelString	String
symbolPixmap	Pixmap

BulletinBoard Resource Set

Name	X Type
allowOverlap	Boolean
autoUnmanage	Boolean
buttonFontList	FontList
cancelButton	Widget
defaultButton	Widget
defaultPosition	Boolean
dialogStyle	unsigned_char
dialogTitle	String
focusCallback	CallbackList
labelFontList	FontList
mapCallback	CallbackList
marginHeight	short
marginWidth	short
noResize	Boolean
resizePolicy	unsigned_char
shadowType	unsigned_char
stringDirection	StringDirection
textFontList	FontList
textTranslations	Translations
unmapCallback	CallbackList

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black

highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	EnterWindow>:Enter() <FocusIn>: FocusIn() Shift<Btn1Down>,<Btn1Up>: pick() <Btn1Down>: Arm() <Btn1Up>: Activate() <Key> F1: Help()

```
<Key>Return:  Return()
<Key>KP_Enter:  Return()
Shift<Btn2Down>,<Btn2Motion>:  resize()
Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>:  resize()
Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>:  resize()
Shift<Btn3Down>,<Btn3Motion>:  move()
Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>:  move()
Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>:  move()
```

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmOption

Serpent Name

XmOption

include_file: Xm/RowColumn.h
class: RowColumnWidgetClass
widget_type: widget
convenience_routine: CreateOptionMenu

Description

The XmOption widget provides the application with the means for obtaining the widget ID for the internally created CascadeButtonGadget. Once the application has obtained the widget ID, it has the ability to adjust the visuals for the CascadeButtonGadget.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
isComposite	Boolean	true
manage	Boolean	true
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

RowColumn

Name	X Type
adjustLast	Boolean
adjustMargin	Boolean

entryAlignment	unsigned_char
entryBorder	short
entryCallback	CallbackList
entryClass	WidgetClass
isAligned	Boolean
isHomogeneous	Boolean
labelString	String
mapCallback	CallbackList
marginHeight	Dimension
marginWidth	Dimension
menuAccelerator	String
menuHelpWidget	Widget
menuHistory	Widget
mnemonic	char
numColumns	short
orientation	unsigned_char
packing	unsigned_char
popupEnabled	Boolean
radioAlwaysOne	Boolean
radioBehavior	Boolean
resizeHeight	Boolean
resizeWidth	Boolean
rowColumnType	unsigned_char
spacing	short
subMenuId	Widget
unmapCallback	CallbackList
whichButton	unsigned_int

RowColumn Special Menu

Name	X Type
menuCursor	String

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	

width		Dimension
x		Position
y		Position
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmPanedWindow

Serpent Name

XmPanedWindow

include_file: Xm/PanedW.h
class: PanedWindowWidgetClass
widget_type: widget

Description

The XmPanedWindow widget is a composite widget that lays out children in a vertically tiled format. Children appear in top-to-bottom fashion, with the child that is inserted first appearing at the top of XmPanedWindow and the child inserted last appearing at the bottom. XmPanedWindow will grow to match the width of its widest child and all other children are forced to this width. The height of XmPanedWindow will be equal to the sum of the heights of all of its children, the spacing between them, and the size of the top and bottom margins.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

PanedWindow

Name	X Type
marginHeight	short

marginWidth	short
refigureMode	Boolean
sashHeight	Dimension
sashIndent	Position
sashShadowThickness	int
sashWidth	Dimension
separatorOn	Boolean
spacing	int

PanedWindow Constraint

Name	X Type
allowResize	Boolean
maximum	int
minimum	int
skipAdjust	Boolean

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	

background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <Btn1Down>: arm() <Btn1Up>: activate()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmPopup

Serpent Name XmPopup

include_file: Xm/RowColumn.h
class: RowColumnWidgetClass
widget_type: widget

Description XmPopUp is a convenience routine which creates a RowColumn widget configured as an XmPopup menu.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
isComposite	Boolean	true
manage	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

RowColumn

Name	X Type
adjustLast	Boolean
adjustMargin	Boolean
entryAlignment	unsigned_char
entryBorder	short
entryCallback	CallbackList

entryClass	WidgetClass
isAligned	Boolean
isHomogeneous	Boolean
labelString	String
mapCallback	CallbackList
marginHeight	Dimension
marginWidth	Dimension
menuAccelerator	String
menuHelpWidget	Widget
menuHistory	Widget
mnemonic	char
numColumns	short
orientation	unsigned_char
packing	unsigned_char
popupEnabled	Boolean
radioAlwaysOne	Boolean
radioBehavior	Boolean
resizeHeight	Boolean
resizeWidth	Boolean
rowColumnType	unsigned_char
spacing	short
subMenuId	Widget
unmapCallback	CallbackList
whichButton	unsigned_int

RowColumn Special Menu

Name	X Type
menuCursor	String

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	

helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize()	

```
Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>:  resize()
Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>:  resize()
Shift<Btn3Down>,<Btn3Motion>:  move()
Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>:  move()
Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>:  move()
```

Methods

Name	Parameters	Description
`move`	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
`resize`	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
`pick`	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmPulldown

Serpent Name

XmPulldown

include_file: /RowColumn.h
class: RowColumnWidgetClass
widget_type: widget

Description

The XmPulldown widget is a convenience routine that creates a RowColumn widget configured as an Popup menu.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
isComposite	Boolean	true
manage	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

RowColumn

Name	X Type
adjustLast	Boolean
adjustMargin	Boolean
entryAlignment	unsigned_char
entryBorder	short
entryCallback	CallbackList

entryClass	WidgetClass
isAligned	Boolean
isHomogeneous	Boolean
labelString	String
mapCallback	CallbackList
marginHeight	Dimension
marginWidth	Dimension
menuAccelerator	String
menuHelpWidget	Widget
menuHistory	Widget
mnemonic	char
numColumns	short
orientation	unsigned_char
packing	unsigned_char
popupEnabled	Boolean
radioAlwaysOne	Boolean
radioBehavior	Boolean
resizeHeight	Boolean
resizeWidth	Boolean
rowColumnType	unsigned_char
spacing	short
subMenuId	Widget
unmapCallback	CallbackList
whichButton	unsigned_int

RowColumn Special Menu

Name	X Type
menuCursor	String

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	

helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize()	

```
Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>:  resize()
Shift<Btn3Down>,<Btn3Motion>:  move()
Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>:  move()
Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>:  move()
```

Methods

Name	Parameters	Description
`move`	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
`resize`	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
`pick`	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmPushButton

Serpent Name

XmPushButton

include_file: /PushB.h
class: PushButtonWidgetClass
widget_type: widget

Description

The XmPushButton widget issues commands within an application. It consists of a text label or Pixmap surrounded by a border shadow. When XmPushButton is selected, the shadow moves to give the appearance that it has been pressed in. When XmPushButton is not selected, the shadow moves to give the appearance that it is out.

Attributes

Serpent

Name	X Type	Default
after	Widget	
allowUserMove	Boolean	
allowUserResize	Boolean	
managedWhenCreated	Boolean	
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

PushButton

Name	X Type	Default
activateCallback	CallbackList	Sixcallback
armCallback	CallbackList	
armColor	Pixel	

armPixmap	Pixmap
disarmCallback	CallbackList
fillOnArm	Boolean
showAsDefault	short

Label Resource Set

Name	X Type
accelerator	String
acceleratorText	String
alignment	unsigned_char
fontList	FontList
labelInsensitivePixmap	Pixmap
labelPixmap	Pixmap
labelString	String
labelType	unsigned_char
marginBottom	short
marginHeight	short
marginLeft	short
marginRight	short
marginTop	short
marginWidth	short
mnemonic	char
recomputeSize	Boolean
stringDirection	StringDirection

Primitive Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightOnEnter	Boolean	false
highlightPixmap	Pixmap	

highlightThickness	short	0
shadowThickness	short	2
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
traversalOn	Boolean	false
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() <Btn1Down>:Arm()<Btn1Up>: Activate()Disarm() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() <Key>Return: ArmAndActivate() <Key>space: ArmAndActivate()	

<EnterWindow>: Enter()
<LeaveWindow>: Leave()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
notify		This method is sent to the dialogue in response to a user event (typically a Btn1Down).
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmQuestionDialog

Serpent Name

XmQuestionDialog

include_file: Xm/MessageB.h
class: MessageBoxWidgetClass
widget_type: widget

Description

The XmQuestionDialog widget is a MessageBox created with a convenience routine. This dialogue is used to get an answer to a question from a user. The dialogue box comes with three buttons: OK, Cancel, and Help. The default symbol is a question mark.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
deactivate	Boolean	false
isComposite	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	
MessageBox	CallbackList	
cancelLabelString	String	
defaultButtonType	unsigned_char	
dialogType	unsigned_char	
helpLabelString	String	
messageAlignment	unsigned_char	
messageString	String	

minimizeButtons	Boolean
okCallback	CallbackList
okLabelString	String
symbolPixmap	Pixmap

BulletinBoard Resource Set

Name	X Type
allowOverlap	Boolean
autoUnmanage	Boolean
buttonFontList	FontList
cancelButton	Widget
defaultButton	Widget
defaultPosition	Boolean
dialogStyle	unsigned_char
dialogTitle	String
focusCallback	CallbackList
labelFontList	FontList
mapCallback	CallbackList
marginHeight	short
marginWidth	short
noResize	Boolean
resizePolicy	unsigned_char
shadowType	unsigned_char
stringDirection	StringDirection
textFontList	FontList
textTranslations	Translations
unmapCallback	CallbackList

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
foreground	Pixel	
helpCallback	CallbackList	

highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	<EnterWindow>: Enter() <FocusIn>: FocusIn() Shift<Btn1Down>,<Btn1Up>: pick() <Btn1Down>: Arm()

<Btn1Up>: Activate()
<Key>F1: Help()
<Key>Return: Return()
<Key>KP_Enter: Return()
Shift<Btn2Down>,<Btn2Motion>: resize()
Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize()
Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize()
Shift<Btn3Down>,<Btn3Motion>: move()
Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move()
Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	X Type	Default
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmRowColumn

Serpent Name XmRowColumn

include_file: /RowColumn.h
class: RowColumnWidgetClass
widget_type: widget

Description The widget is a general purpose XmRowColumn manager capable of containing any widget type as a child. In general, XmRowColumn requires no special knowledge of how its children function and provides nothing beyond support for several different layout styles. However, XmRowColumn can be configured as a menu, in which case, it expects only certain children and it configures to a particular layout. The menus supported are: MenuBar, Pulldown, or Popup MenuPanes and OptionMenu.

Attributes
Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
isComposite	Boolean	true
managedWhenCreated	Boolean	true
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

RowColumn

Name	X Type
adjustLast	Boolean
adjustMargin	Boolean
entryAlignment	unsigned_char
entryBorder	short
entryCallback	CallbackList
entryClass	WidgetClass
isAligned	Boolean
isHomogeneous	Boolean
labelString	String
mapCallback	CallbackList
marginHeight	Dimension
marginWidth	Dimension
menuAccelerator	String
menuHelpWidget	Widget
menuHistory	Widget
mnemonic	char
numColumns	short
orientation	unsigned_char
packing	unsigned_char
popupEnabled	Boolean
radioAlwaysOne	Boolean
radioBehavior	Boolean
resizeHeight	Boolean
resizeWidth	Boolean
rowColumnType	unsigned_char
spacing	short
subMenuId	Widget
unmapCallback	CallbackList
whichButton	unsigned_int

RowColumn Special Menu

Name	X Type
menuCursor	String

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	

height		Dimension
mappedWhenManaged		Boolean
screen		Screen
sensitive		Boolean
width		Dimension
x		Position
y		Position
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>:resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>:resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>:resize() Shift<Btn3Down>,<Btn3Motion>:move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>:move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>:move() <Unmap>: MenuUnmap() <FocusIn>: MenuFocusIn() <FocusOut>:MenuFocusOut() <EnterWindow>: MenuEnter() <Key>Left: MenuGadgetTraverseLeft() <Key>Right: MenuGadgetTraverseRight() <Key>Up: MenuGadgetTraverseUp() <Key>Down: MenuGadgetTraverseDown()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmScale

Serpent Name

XmScale

include_file: /Scale.h
class: ScaleWidgetClass
widget_type: widget

Description

The XmScale widget is used by an application to indicate a value from within a range of values, and it allows the user to input or modify a value from the same range. XmScale has an elongated, rectangular region similar to that of ScrollBar. Inside this region is a slider that indicates the current value of XmScale. The user can also modify the value of XmScale by moving the slider within the rectangular region. XmScale can also include a set of labels located outside the scale region. These can indicate the relative value at various positions along the scale.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

Separator

Name	X Type
decimalPoints	short

dragCallback	CallbackList	
fontList	FontList	
highlightOnEnter	Boolean	
highlightThickness	short	
maximum	int	
minimum	int	
orientation	unsigned_char	
processingDirection	unsigned_char	
scaleHeight	Dimension	
scaleWidth	Dimension	
showValue	Boolean	
titleString	String	
traversalOn	Boolean	
value	int	
valueChangedCallback	CallbackList	scale_callback

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
shadowThickness	short	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <Btn1Down>: Arm() <Btn1Up>: Activate() <EnterWindow>: Enter() <FocusIn>: FocusIn()	

Methods

Name	Parameters	Description
value_changed		This method is sent to the dialogue in response to a user event (typically moving a slider).
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.

resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmScreenObject

Serpent Name XmScreenObject

Description The XmScreenObject widget allows for the detection of screen display IDs: Display size includes height and width. Display type includes color or black & white.

Attributes

Serpent

Name	X Type
color	Boolean
display	int
height	Dimension
screen	Screen
width	Dimension

XmScrollBar

Serpent Name

XmScrollBar

include_file: Xm/ScrollBar.h
class: xmScrollBarWidgetClass
widget_type: widget

Description

The XmScrollBar widget gives the user a means of viewing data that would not otherwise fit into the available space. The Xmscrollbar is typically located adjacent to some viewing region. The scrollbar consists of two rectangular regions—a large one called the scroll region and a small one called the slider—and two arrows. When the slider is moved, directly or through the arrows, the viewable data is scrolled through the viewing region.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

ScrollBar

Name	X Type
decrementCallback	CallbackList
dragCallback	CallbackList
increment	int

incrementCallback	CallbackList	
initialDelay	int	
maximum	int	
minimum	int	
orientation	unsigned_char	
pageDecrementCallback	CallbackList	
pageIncrement	int	
pageIncrementCallback	CallbackList	
processingDirection	unsigned_char	
repeatDelay	int	
showArrows	Boolean	
sliderSize	int	
toBottomCallback	CallbackList	
toTopCallback	CallbackList	
value	int	
valueChangedCallback	CallbackList	scroll_callback

Primitive Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightOnEnter	Boolean	false
highlightPixmap	Pixmap	
highlightThickness	short	0
shadowThickness	short	2
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
traversalOn	Boolean	false
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL
accelerators	Accelerators	
ancestorSensitive	Boolean	

background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move this widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).
value_changed		This method is sent to the dialogue in response to a user event (typically moving a slider).

XmScrolledWindow

Serpent Name XmScrolledWindow

include_file: Xm/ScrolledW.h
class: xmScrolledWindowWidgetClass
widget_type: widget

Description The XmScrolledWindow widget combines one or more ScrollBar widgets and a viewing area to implement a visible window onto some other (usually larger) data display. The visible part of the window can be scrolled through the larger display by the use of ScrollBars.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	

ScrolledWindow Resource Set

Name	X Type
clipWindow	Widget
horizontalScrollBar	Widget
scrollBarDisplayPolicy	unsigned_char
scrollBarPlacement	unsigned_char
scrolledWindowMarginHeight	Dimension

scrolledWindowMarginWidth	Dimension
scrollingPolicy	unsigned_char
spacing	int
verticalScrollBar	Widget
visualPolicy	unsigned_char
workWindow	Widget

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1

colormap		Colormap	
depth		Cardinal	
destroyCallback		CallbackList	
height		Dimension	
mappedWhenManaged		Boolean	
screen		Screen	
sensitive		Boolean	
width		Dimension	1
x		Position	
y		Position	
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()	

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move this widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize this widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmSeparator

Serpent Name XmSeparator

include_file: Xm/Separator.h
class: XmSeparatorWidgetClass
widget_type: widget

Description The XmSeparator widget is a primitive widget that separates items in a display. Several different line drawing styles are provided, as well as horizontal and vertical orientation.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	serpdef
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	
Separator	short	
orientation	unsigned_char	
separatorType	unsigned_char	

Primitive Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
foreground	Pixel	

helpCallback	CallbackList	
highlightColor	Pixel	black
highlightOnEnter	Boolean	false
highlightPixmap	Pixmap	
highlightThickness	short	0
shadowThickness	short	2
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
traversalOn	Boolean	false
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()	

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmText

Serpent Name

XmText

include_file: Xm/Text.h
class: TextWidgetClass
widget_type: widget
check_routine: check_MText

Description

The XmText widget provides a single and multi-line text editor for customizing both user and program interfaces. It can be used for single-line string entry, forms entry with verification procedures, and full-window editing. It provides an application with a consistent editing system for textual data. The screen's textual data adjusts to the application writer's needs.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
sendBuffer	Boolean	
selectedX	Position	0
selectedY	Position	0
widget	int	

Text

Name	X Type
activateCallback	CallbackList
autoShowCursorPosition	Boolean

cursorPosition	TextPosition
editable	Boolean
editMode	int
focusCallback	CallbackList
losingFocusCallback	CallbackList
marginHeight	short
marginWidth	short
maxLength	int
modifyVerifyCallback	CallbackList
motionVerifyCallback	CallbackList
topPosition	TextPosition
value	String
valueChangedCallback	CallbackList

Text Input

Name	X Type
pendingDelete	Boolean
selectionArray	Pointer
selectThreshold	int

Text Output

Name	X Type
blinkRate	int
columns	short
cursorPositionVisible	Boolean
fontList	FontList
resizeHeight	Boolean
resizeWidth	Boolean
rows	short
wordWrap	Boolean

Text Scrolled Text

Name	X Type
scrollHorizontal	Boolean

scrollLeftSide	Boolean
scrollTopSide	Boolean
scrollVertical	Boolean

Primitive Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightOnEnter	Boolean	false
highlightPixmap	Pixmap	
highlightThickness	short	0
shadowThickness	short	2
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_Pixmap
traversalOn	Boolean	
Techdef		false
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	

height		Dimension
mappedWhenManaged		Boolean
screen		Screen
sensitive		Boolean
width		Dimension
x		Position
y		Position
translations	Translations:	Shift<Key>Tab: tab() Ctrl<Key>Tab:next-tab-group() <Key>Tab:next-tab-group() <Key>Up:traverse-prev() <Key>Down:traverse-next() <Key>Home:traverse-home() Ctrl<Key>Right:forward-word() Shift<Key>Right: key-select(right) <Key>Right:forward-character() Ctrl<Key>Left:backward-word() Shift<Key>Left:key-select(left) <Key>Left:backward-character() Shift<Key>Delete: delete-previous-word() <Key>Delete:delete-previous-character() Shift<Key>Linefeed:delete-next-word() <Key>Linefeed:delete-next-character() Shift<Key>F13:delete-next-word() <Key>F13:delete-next-character() Shift<Key>BackSpace:delete-previous-word() <Key>BackSpace:delete-previous-character() <Key>Return:activate() send() ~Ctrl <Key>:self-insert() Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <Btn1Down>: grab-focus() Button1<PtrMoved>: extend-adjust() <Btn1Up>: extend-end() <Btn2Down>: secondary-start() Button2<PtrMoved>: secondary-adjust() Ctrl<Btn2Up>: move-to() secondary-end-and-kill() <Btn2Up>: copy-to() secondary-end() <ClientMessage>: secondary-stuff() remote-kill-selection() <LeaveWindow>: leave()

<FocusIn>: focusIn()
<FocusOut>: focusOut()
<Unmap>: unmap()

Methods

Name	Parameter	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
send	value	This method is returned when the send_buffer flag is set to true by the dialogue, or when there is a translation table action.
tab		This method is sent to the dialogue in response to a shifted tab.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmToggleButton

Serpent Name

XmToggleButton

include file: ToggleB.h
class xmToggleButtonWidgetClass
widget_type: widget

Description

The XmToggleButton widget presents a choice of values (such as yes or no) to the user. When the user selects a toggle button, its value remains in use until another choice is toggled. Usually this widget consists of an indicator (square or diamond) with either text or a pixmap to its right. However, it can also consist of just text or a pixmap without the indicator.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	
ToggleButton	CallbackList	
disarmCallback	CallbackList	
fillOnSelect	Boolean	
indicatorOn	Boolean	
indicatorType	unsigned_char	
selectColor	Pixel	
selectPixmap	Pixmap	

set	Boolean	
spacing	short	
valueChangedCallback	CallbackList	toggle_callback
visibleWhenOff	Boolean	

Label Resource Set

Name	X Type
accelerator	String
acceleratorText	String
alignment	unsigned_char
fontList	FontList
labelInsensitivePixmap	Pixmap
labelPixmap	Pixmap
labelString	String
labelType	unsigned_char
marginBottom	short
marginHeight	short
marginLeft	short
marginRight	short
marginTop	short
marginWidth	short
mnemonic	char
recomputeSize	Boolean
stringDirection	StringDirection

Primitive Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
foreground	Pixel	
helpCallback	CallbackList	
highlightColor	Pixel	black
highlightOnEnter	Boolean	false
highlightPixmap	Pixmap	

highlightThickness	short	0
shadowThickness	short	2
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
traversalOn	Boolean	false
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	Shift<Btn1Down>,<Btn1Up>: pick() Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move() <Btn1Down>: Arm() <Btn1Up>: Select() Disarm() <Key>Return: ArmAndActivate() <Key>space: ArmAndActivate() <EnterWindow>: Enter() <LeaveWindow>: Leave()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location, and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).
toggle	set	This method sends the new value of the toggle button to the user when the button is pressed.

XmTopLevelShell

Serpent Name

XmTopLevelShell

include_file X11/Shell.h
class: applicationShellWidgetClass
widget_type: shell

Description

The XmTopLevelShell widget is the standard user shell. It is used for normal top-level windows and mediates the interaction between the widgets and the window manager.

Attributes
TopLevelShell

Name	X Type
iconic	Boolean
iconName	String
VendorShell	unsigned_char
keyboardFocusPolicy	unsigned_char
mwmDecorations	int
mwmFunctions	int
mwmInputMode	int
mwmMenu	String
shellUnitType	unsigned_char
WMShell	int
iconMask	Pixmap
iconPixmap	Pixmap
iconWindow	Window
iconX	int
iconY	int
initialState	int
input	Boolean

maxAspectX	int
maxAspectY	int
maxHeight	int
maxWidth	int
minAspectX	int
minAspectY	int
minHeight	int
minWidth	int
title	char_star
transient	Boolean
waitForWm	Boolean
widthInc	int
windowGroup	XID
wmTimeout	int

Shell Resource Set

Name	X Type
allowShellResize	Boolean
createPopupChildProc	Boolean
geometry	caddr_t
overrideRedirect	Boolean
popdownCallback	caddr_t
popupCallback	caddr_t
saveUnder	Boolean

Composite Resource Set

Name	X Type
insertPosition	OrderProc
accelerators	Accelerators
ancestorSensitive	Boolean
background	Pixel
backgroundPixmap	Pixmap
borderColor	Pixel
borderPixmap	Pixmap

borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	Shift<Btn2Down>,<Btn2Motion>: resize() Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize() Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize() Shift<Btn3Down>,<Btn3Motion>: move() Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move() Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	Parameters	Default
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize this widget with the mouse and sends its x and y location and new width and height to the dialogue.

XmWarningDialog

Serpent Name XmWarningDialog

include_file: Xm/MessageB.h
class: xmMessageBoxWidgetClass
widget_type: widget

Description The XmWarningDialog widget is a MessageBox created with a convenience routine. This dialogue is used to warn a user of the consequences of some action. The dialogue box comes with three buttons: OK, Cancel, and Help. The default symbol is an exclamation point.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
deactivate	Boolean	false
isComposite	Boolean	true
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	
MessageBox	CallbackList	
cancelLabelString	String	
defaultButtonType	unsigned_char	
dialogType	unsigned_char	
helpLabelString	String	
messageAlignment	unsigned_char	

messageString	String
minimizeButtons	Boolean
okCallback	CallbackList
okLabelString	String
symbolPixmap	Pixmap

BulletinBoard Resource Set

Name	X Type
allowOverlap	Boolean
autoUnmanage	Boolean
buttonFontList	FontList
cancelButton	Widget
defaultButton	Widget
defaultPosition	Boolean
dialogStyle	unsigned_char
dialogTitle	String
focusCallback	CallbackList
labelFontList	FontList
mapCallback	CallbackList
marginHeight	short
marginWidth	short
noResize	Boolean
resizePolicy	unsigned_char
shadowType	unsigned_char
stringDirection	StringDirection
textFontList	FontList
textTranslations	Translations
unmapCallback	CallbackList

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
foreground	Pixel	

helpCallback	CallbackList	
highlightColor	Pixel	black
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	<EnterWindow>: Enter() <FocusIn>: FocusIn() Shift<Btn1Down>,<Btn1Up>: pick()

<Btn1Down>: Arm()
<Btn1Up>: Activate()
<Key>F1: Help()
<Key>Return: Return()
<Key>KP_Enter: Return()
Shift<Btn2Down>,<Btn2Motion>: resize()
Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize()
Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize()
Shift<Btn3Down>,<Btn3Motion>: move()
Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move()
Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	Parameters	Description
move	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
resize	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
pick	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

XmWorkingDialog

Serpent Name XmWorkingDialog

include_file: Xm/MessageB.h
class: xmMessageBoxWidgetClass
widget_type: widget

Description The XmWorkingDialog widget is a MessageBox created with a convenience routine. This dialogue is used to notify a user about a time-consuming operation in progress. The dialogue box comes with three buttons: OK, Cancel, and Help. The default symbol is an hourglass.

Attributes

Serpent

Name	X Type	Default
allowUserMove	Boolean	false
allowUserResize	Boolean	false
deactivate	Boolean	false
isComposite	Boolean	false
method	MethodName	
parent	Widget	
selectedX	Position	0
selectedY	Position	0
widget	int	
MessageBox	CallbackList	
cancelLabelString	string	
defaultButtonType	unsigned_char	
dialogType	unsigned_char	
helpLabelString	String	
messageAlignment	unsigned_char	

messageString	String
minimizeButtons	Boolean
okCallback	CallbackList
okLabelString	String
symbolPixmap	Pixmap

BulletinBoard Resource Set

Name	X Type
allowOverlap	Boolean
autoUnmanage	Boolean
buttonFontList	FontList
cancelButton	Widget
defaultButton	Widget
defaultPosition	Boolean
dialogStyle	unsigned_char
dialogTitle	String
focusCallback	CallbackList
labelFontList	FontList
mapCallback	CallbackList
marginHeight	short
marginWidth	short
noResize	Boolean
resizePolicy	unsigned_char
shadowType	unsigned_char
stringDirection	StringDirection
textFontList	FontList
textTranslationsa	Translations
unmapCallback	CallbackList

Manager Resource Set

Name	X Type	Default
bottomShadowColor	Pixel	
bottomShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
foreground	Pixel	

helpCallback	CallbackList	
highlightColor	Pixel black	
highlightPixmap	Pixmap	
shadowThickness	short	
topShadowColor	Pixel	
topShadowPixmap	Pixmap	UNSPECIFIED_PIXMAP
unitType	unsigned_char	PIXELS
userData	caddr_t	NULL

Composite Resource Set

Name	X Type
insertPosition	OrderProc

Core Resource Set

Name	X Type	Default
accelerators	Accelerators	
ancestorSensitive	Boolean	
background	Pixel	
backgroundPixmap	Pixmap	
borderColor	Pixel	
borderPixmap	Pixmap	
borderWidth	Dimension	1
colormap	Colormap	
depth	Cardinal	
destroyCallback	CallbackList	
height	Dimension	
mappedWhenManaged	Boolean	
screen	Screen	
sensitive	Boolean	
width	Dimension	
x	Position	
y	Position	
translations	Translations:	<EnterWindow>: Enter() <FocusIn>: FocusIn() Shift<Btn1Down>,<Btn1Up>: pick()

<Btn1Down>: Arm()
<Btn1Up>: Activate()
<Key>F1: Help()
<Key>Return: Return():
<Key>KP_Enter: Return()
Shift<Btn2Down>,<Btn2Motion>: resize()
Shift<Btn2Down>,<Btn2Motion>,<Leave>,<Btn2Up>: resize()
Shift<Btn2Down>,<Btn2Motion>,<Btn2Up>: resize()
Shift<Btn3Down>,<Btn3Motion>: move()
Shift<Btn3Down>,<Btn3Motion>,<Leave>,<Btn3Up>: move()
Shift<Btn3Down>,<Btn3Motion>,<Btn3Up>: move()

Methods

Name	Parameters	Description
`move`	x, y	This method allows the user to move the widget with the mouse and sends the widget's new x and y location to the dialogue.
`resize`	x, y, width, height	This method allows the user to resize the widget with the mouse and sends its x and y location and new width and height to the dialogue.
`pick`	selectedX, selectedY	This method allows the user to select a point on the widget with the mouse and sends the location of the point to the dialogue in response to a user event (typically a shifted Btn1Down).

Index

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS
None

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for Public Release
Distribution Unlimited

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

4. PERFORMING ORGANIZATION REPORT NUMBER(S
CMU/SEI-91-UG-5

5. MONITORING ORGANIZATION REPORT NUMBER(S)
CMU/SEI-91-UG-5

6a. NAME OF PERFORMING ORGANIZATION
Software Engineering Institute

6b. OFFICE SYMBOL (if applicable)
SEI

7a. NAME OF MONITORING ORGANIZATION
SEI Joint Program Office

6c. ADDRESS (City, State and ZIP Code)
Carnegie Mellon University
Pittsburgh PA 15213

7b. ADDRESS (City, State and ZIP Code)
ESD/AVS
Hanscom Air Force Base, MA 01731

8a. NAME OFFUNDING/SPONSORING ORGANIZATION
SEI Joint Program Office

8b. OFFICE SYMBOL (if applicable)
ESD/AVS

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F1962890C0003

8c. ADDRESS (City, State and ZIP Code)
Carnegie Mellon University
Pittsburgh PA 15213

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT NO.
63752F	N/A	N/A	N/A

11. TITLE (Include Security Classification)
Serpent: Slang Reverence Manual

12. PERSONAL AUTHOR(S)
SEI User Interface Project

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final	FROM TO	May 1991	~350

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse of necessary and identify by block number)
Serpent, UIMS, user interface management system, user interface generators, Slang, dialogue

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Serpent is a user interface management system (UIMS) that supports the development and implementation of user interfaces, providing an editor to specify the user interface and a runtime system that enables communication between the application and the end user. This manual describes the model, syntax, and semantics of the Slang dialogue language, the language within Serpent used for the specification of user interfaces. Readers should be familiar with general UIMS concepts, have a working knowledge of programming languages, and understand the concepts described in *Serpent Overview* and Serpent: System Guide.

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ■ SAME AS RPT ☐ DTIC USERS ■

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified, Unlimited Distribution

22a. NAME OF RESPONSIBLE INDIVIDUAL
John S. Herman, Capt, USAF

22b. TELEPHONE NUMBER (Include Area Code)
(412) 268-7630

22c. OFFICE SYMBOL
ESD/AVS (SEI JPO)

ABSTRACT —continued from page one, block 19

CMU/SEI-91-UG-1	Serpent Overview
CMU/SEI-91-UG-2	Serpent: System Guide
CMU/SEI-91-UG-3	Serpent: Saddle User's Guide
CMU/SEI-91-UG-4	Serpent: Dialogue Editor User's Guide
CMU/SEI-91-UG-5	Serpent: Slang Reference Manual
CMU/SEI-91-UG-6	Serpent: C Application Developer's Guide
CMU/SEI-91-UG-7	Serpent: Ada Application Developer's Guide
CMU/SEI-91-UG-8	Serpent: Guide to Adding Toolkits